# MuxProofs: Succinct Arguments for Machine Computation from Vector Lookups

Zijing DiLucas XiaWilson NguyenNirvan TyagiStanford UniversityStanford UniversityStanford UniversityStanford University

**Abstract.** Proofs for machine computation prove the correct execution of arbitrary programs that operate over fixed instruction sets (e.g., RISC-V, EVM, Wasm). A standard approach for proving machine computation is to prove a universal set of constraints that encode the full instruction set at each step of the program execution. This approach incurs a proving cost per execution step on the order of the total sum of instruction constraints for all of the instructions in the set, despite each step of the program only executing a single instruction. Existing proving approaches that avoid this universal cost per step (and incur only the cost of a single instruction's constraints per step) either fail to provide zero-knowledge or rely on recursive proof composition for which security relies on the heuristic instantiation of the random oracle.

We present new protocols for proving machine execution that resolve these limitations, enabling prover efficiency on the order of only the executed instructions while achieving zero-knowledge and avoiding recursive proofs. Our core technical contribution is a new primitive that we call a succinct vector lookup argument which enables a prover to build up a machine execution "on-the-fly". We propose succinct vector lookups for both univariate polynomial and multivariate polynomial commitments in which vectors are encoded on cosets of a multiplicative subgroup and on subcubes of the boolean hypercube, respectively. We instantiate our proofs for machine computation by integrating our vector lookups with existing efficient, succinct non-interactive proof systems for NP.

# Contents

| 1 | Introduction  | 3                           |
|---|---|-----------------------------|
| 2 | Technical Overview         2.1       Strategy: Computation Commitments from Machine Commitments   | 7<br>8<br>10<br>12          |
| 3 | Preliminaries         3.1       Notation         3.2       Proof and Argument Systems         3.3       Useful PolyIOPs   | <b>14</b><br>14<br>15<br>18 |
| 4 | Succinct Vector Lookup  | 18                          |
| 5 | Succinct Arguments for Unrolled Machine Execution from Vector Lookups5.1Mux-PLONK: Adapting the PLONK PolyIOP to Machine Execution5.2Extensions to Other PolyIOPs | <b>23</b><br>24<br>30       |
| 6 | Evaluation6.1Succinct Vector Lookup Evaluation6.2Machine Computation Evaluation   | <b>31</b><br>31<br>32       |
| A | Useful PolyIOPs   | 37                          |
| B | Generic Zero-Knowledge Compiler for Univariate PolyIOPs   | 39                          |
| С | Multivariate Succinct Vector Lookup   | 41                          |
| D | Mux-HyperPLONK Construction   | 45                          |
| E | Mux-Marlin Construction   | 49                          |
| F | Plookup-based Succinct Vector Lookup         F.1       Vector Permutation   | <b>63</b><br>63<br>68       |

# 1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) [Kil92, Mic94, GW11, BCCT12] enable efficient verification of NP statements. Improving prover efficiency is a key challenge in the design of SNARKs and a pathway to increase their practically deployment. In this work, we improve the prover efficiency for an important class of statements known as *machine computation* [BFR<sup>+</sup>13, BCTV14b].

In machine computation, statements are defined by the output of a program operating over a predefined fixed instruction set. A program maintains some state including an instruction pointer which determines the next instruction to execute from the instruction set. The result of an instruction execution step is an updated state and instruction pointer pointing to the next instruction to be executed.

A starting motivation for our goal of improving prover time for machine computations is another class of statements in which structure can be leveraged for prover efficiency: disjunctions [CDS94, AOS02]. A disjunctive statement consists of a set of clauses, each of which is itself an NP statement. It is satisfied if there exists a satisfying witness for at least one of the clauses. Many privacy-preserving systems rely on *zero-knowledge* proofs for disjunctive statements in which the clause that is satisfied must remain hidden. A standard approach to proving the validity of a disjunctive statement in zero-knowledge is by simply encoding the clauses into a constraint system that includes the constraints for each individual clause as well as constraints for a disjunction over validity of all clauses; this constraint system over the full set of clauses is sometimes referred to as a *universal* constraint system. Any compatible zero-knowledge SNARK for NP can be used with the universal constraint system to produce a succinct proof. Unfortunately, given no shared structure between the clauses, the universal constraint system has size equal to the sum of the individual clause constraint encodings, and prover time scales accordingly. Prior work has shown how to do better in some cases [GK15, HK20, BMRS21, ACF21, GGHK22, GHKS22], where most recently Goel et al. show how to build SNARKs for disjunctions with NP clauses in which prover time scales with  $\tilde{O}(C + \ell)$  computation where C is the constraint size of a single clause and  $\ell$  is the number of clauses in the disjunction. This is in contrast to a  $\tilde{O}(C\ell)$  cost of the universal constraint system approach.

In this work, we would like to obtain similar prover time gains by taking advantage of the similar structure found in machine computation. Machine computation resembles a disjunction as the prover would like to prove at each step that the new program state is the result of applying one of the valid instructions in the instruction set. Indeed, it is more complex than a simple disjunction as the prover needs to additionally prove that the correct instruction is executed at each step and the intermediate program state between steps is consistent; further, the prover must do this over a sequence of many execution steps. That said, the high level goal of constructing a prover that scales only with the size of executed instructions rather than the sum of executions of a universal instruction set is similar. We target a prover time of  $\tilde{O}((n+\ell)C)$  versus the prover time of  $\tilde{O}(n\ell C)$  achieved through universal constraint systems where n is the number of executed instructions,  $\ell$  is the number of instructions in the instruction set, and C is the constraint size of a single instruction.

Proofs for correct execution of machine computation have received significant attention with active projects working to build proof systems for prominent instruction sets including EVM [Pol,zkS], RISC-V [RIS], and WebAssembly [zkW]. These proofs have already been deployed to improve the scalability of blockchain auditing, in particular, with respect to audit of smart contract execution. Now, instead of requiring auditors to execute smart contracts locally to determine and verify a new blockchain state, auditors can simply verify a succinct proof of correct machine computation for the instruction set to which the smart contract is compiled. The task of producing such a proof can be outsourced to any untrusted prover. Importantly, the proving time for producing such a proof must be manageable as it will determine the contract execution throughput that the system will be able to support. As we discuss below, deployed systems such as zkEVM [Pol] do not provide zero-knowledge (despite the misnomer), in part, due to prover efficiency reasons. Nevertheless, zero-knowledge is an important property for these applications and will be necessary to realize next-generation systems that support private smart contract execution [BCG<sup>+</sup>20, XCZ<sup>+</sup>22].

**Prior approaches to succinct proofs for machine computation.** There have been two overarching approaches to proving correct execution of machine computation. The first is through the use of *incrementally-verifiable computation*.

(IVC) [Val08] in which each instruction step is proved in sequence building on a proof for the correct execution of the program up to that point. The second approach first "unrolls" the complete program execution and proves it as a single constraint system. Figure 1 provides a summary.

Incremental proof systems for machine computation. Ben-Sasson et al. [BCTV14a] demonstrate the ability to build proofs for machine execution from IVC using recursive proofs [BCCT13] in which the constraint system for each step verifies one instruction step and recursively verifies a SNARK for the previous step. This work uses a universal constraint system encoding the full instruction set at each step. This general approach can be instantiated with state-of-the-art approaches for achieving IVC [BGH19, BCMS20, BDFG21, BCL<sup>+</sup>21, KST22], which avoid direct verification of recursive proofs and therefore achieve lower recursive overhead. However, this strategy will incur computation on the order of the size of the universal constraint system at each step ( $\tilde{O}(\ell C)$  per step), as opposed to just the size of the executed instruction constraints ( $\tilde{O}(C)$  per step).

Instead, to obviate the universal constraint system, an alternate strategy would be to commit to constraint systems for each instruction in the instruction set, e.g., in a Merkle tree commitment. At each step, the prover would open up the commitment to the instruction to be executed for the step, prove the instruction execution, and recursively verify a proof for the previous step. In concurrent work, SuperNova [KS22] and Protostar [BC23] refine this high-level blueprint building on the state-of-the-art recursion techinques [KST22, BCL<sup>+</sup>21] and further employing techniques in offline memory checking [BEG<sup>+</sup>91, SAGL18, LNS20] to remove asymptotic dependence on the number of instructions in the instruction set when opening the instruction commitment. In this way, SuperNova and Protostar build proofs for machine computation using IVC that achieve  $\tilde{O}((n + \ell)C)$  prover cost (formalized as *non-uniform IVC* [KS22]).

A drawback of all these approaches that fall under the IVC strategy is that they rely on recursively proving computations that query random oracles. For example, in each step, the prover may need to prove the execution of a recursive verifier that queries a random oracle in its decision process. Outside of recent exploratory work [CCS22, CCG<sup>+</sup>23] in specialized models, we do not have secure constructions of SNARKs that prove computations which query random oracles. Thus, in practice, the security of such constructions is based on a heuristic assumption. In particular, the assumption, informally, states these constructions remain secure if the random oracle is instantiated with a particular concrete hash function. This is necessary to encode the recursive computation in a manner suitable for existing SNARKs. As such, these constructions have not been shown to be secure in the random oracle model.

As we describe next, an alternate strategy avoids IVC (and its associated heuristic assumption) by unrolling and proving the full program execution in its entirety.

<u>Unrolled proof systems for machine computation</u>. Unrolled proof systems for universal constraint systems incur cost on the size of the universal constraint system per instruction unrolled ( $\tilde{O}(n\ell C)$ ) simply by repeating the universal constraint system for each execution step [BCTV14b, BCG<sup>+</sup>18, BCG<sup>+</sup>19]. Other unroll approaches including Pantry [BFR<sup>+</sup>13], Buffet [WSR<sup>+</sup>15], and vRAM [ZGK<sup>+</sup>18] avoid the use of universal constraint systems and achieve prover computation that we desire on the order only of the executed instructions ( $\tilde{O}((n + \ell)C)$ ).

However, the unroll approach is not able to provide full zero-knowledge of program execution—at the very least, it must leak some upper bound on the number of execution steps. Prior unroll proof systems that achieve prover computation on the order of only executed instructions leak even more: Pantry [BFR+13] and Buffet [WSR+15] both require program-specific preprocessing in which the full program description must be revealed to the verifier, while vRAM [ZGK+18] avoids program-specific preprocessing but reveals the number of times each instruction is executed. Indeed, deployed systems such as zkEVM [Pol] take an unroll approach but do not provide zero-knowledge. Intuitively, providing zero-knowledge for unrolled executions without incurring universal constraint costs is challenging; the key issue is that it cannot be known ahead of time which instruction will be executed at each execution step.

**Our approach using succinct vector lookups.** In this work, we propose the first unrolled proof system for machine computation that supports zero-knowledge (beyond an upper bound on execution length) while also incurring prover computation of  $\tilde{O}((n + \ell)C)$  (see below for comparison to concurrent work). In comparison to prior proof systems that are able to achieve this prover complexity: Pantry [BFR+13], Buffet [WSR+15], and vRAM [ZGK+18] do not provide zero-knowledge, and SuperNova [KS22] and Protostar [BC23] rely on IVC techniques which require a heuristic

| Protocols     |   | Prover computation?    | Execution leakage?       | ROM-secure? |
|---------------|---|------------------------|--------------------------|-------------|
| IVC w/ UC     | [BCTV14a]   | $\tilde{O}(n\ell C)$   | no leakage               | Ν           |
| IVC w/ Exe    | SuperNova [KS22], Protostar [BC23]  | $\tilde{O}((n+\ell)C)$ | no leakage               | N           |
| Unroll w/ UC  | [BCTV14b], Arya [BCG <sup>+</sup> 18], [BCG <sup>+</sup> 19], Mirage [KPPS20] | $\tilde{O}(n\ell C)$   | instruction upper bound  | Y           |
| Unroll w/ Exe | Pantry [BFR <sup>+</sup> 13], Buffet [WSR <sup>+</sup> 15]                    | $\tilde{O}((n+\ell)C)$ | full execution           | Y           |
|               | vRAM [ZGK+18]   | $\tilde{O}((n+\ell)C)$ | instruction multiplicity | Y           |
|               | Sublonk [CGG+23] (Concurrent work)  | $\tilde{O}(nC)$        | instruction upper bound  | Y           |
|               | Mux-PLONK, Mux-Marlin (This work)   | $\tilde{O}((n+\ell)C)$ | instruction upper bound  | Y           |
|               | Mux-HyperPLONK (This work)  | $O((n+\ell)C)$         | instruction upper bound  | Y           |

Figure 1: Summary of strategy and characteristics of machine execution proof protocols. UC refers to using universal circuit constraints and Exe refers to using constraints just for executed instructions. The asymptotic prover time is given in terms of the number of executed instructions n, the number of instructions in the instruction set  $\ell$ , and the constraint size of a single instruction C. Execution leakage refers to aspects of the program execution that are revealed to the verifier. The final column refers to security in the random oracle model. IVC constructions with recursive proof composition rely on a heuristic security step instead of a random oracle model security analysis. The  $\tilde{O}$  notation hides polylogarithmic factors; in particular, the prover time for Mux-HyperPLONK is strictly  $O((n + \ell)C)$  and does not have polylogarithmic factors.

instantiation of the random oracle for recursion.

To do this, our main technical contributions are new succinct proof systems for *vector lookups*. A vector lookup allows a committer to prove that a commitment to a list of vectors contains only vectors that exist in a reference table of vectors, i.e., that every vector was "looked up" from some index in the reference table. Generically, a vector lookup can be constructed from any element lookup proof system. The generic construction starts off by committing to each position of the vectors in a separate commitment. Then, it homomorphically combines the commitments into a single table via a linear combination with a random verifier challenge. Finally, it runs the element lookup protocol with respect to the random table [GW20]. This transformation can be applied to any suitable element lookup [BCG<sup>+</sup>18, GW20, ZBK<sup>+</sup>22, PK22, ZGK<sup>+</sup>22, GK22, EFG22] but results in verification that is linear in the vector size. In contrast, our new vector lookups admit succinct verification running in time logarithmic in the vector size which will be important for our machine computation application.

Equipped with our new vector lookup, we proceed to construct an unrolled proof system for machine computation. Our approach combines vector lookup arguments with proof systems for NP constraint systems that are compiled from a common information-theoretic abstraction known as a *polynomial interactive oracle proof* (polyIOP) where computation is encoded as a vector of constraints within a polynomial commitment in a preprocessing step. In our approach, the vector of constraints representing each instruction in the instruction set are encoded together in a table that represents the available instructions. Then a vector lookup is used to construct a polynomial commitment on-the-fly that represents the unrolled machine execution, "looking up" the constraints for the executed instructions.

We instantiate this approach with three existing polyIOPs for NP, PLONK [GWC19], Marlin [CHM<sup>+</sup>20], and HyperPLONK [CBBZ23]. PLONK and Marlin are both *univariate* polyIOPs: when combined with our univariate vector lookup CosetLkup and a suitable univariate polynomial commitment scheme (e.g., Marlin-KZG [CHM<sup>+</sup>20, KZG10]), the resulting proof systems for machine computation, Mux-PLONK and Mux-Marlin, admit constant proof size at the expense of quasilinear proving time  $\tilde{O}((n + \ell)C)$ . We also build on the multivariate polyIOP, HyperPLONK, which when combined with our multivariate vector lookup SubcubeLkup and a suitable multivariate polynomial commitment scheme (e.g., Brakedown [GLS<sup>+</sup>23] or Orion [XZS22, CBBZ23]) results in Mux-HyperPLONK with an efficient linear-time ( $O((n + \ell)C)$ ) prover and sublinear proof size.

Minimal instruction sets (e.g., TinyRAM [BCTV14b] or RISC-V) have instruction set size  $\ell$  of 30-50 instructions and instruction constraint size C on the order of 128 constraints (in the case of enforcing 64-bit computation modulo a prime). Our evaluation estimates indicate our protocols incur less than  $2.5 \times$  overhead on top of comparable zeroknowledge proof systems for the same computation size. Given our protocols reduce the computation size by a factor of  $\ell$  (i.e.,  $30-50 \times$ ) over unrolled zero-knowledge proof systems for universal constraints, our cost accounting indicates our protocols reduce proving time by up to  $9 \times$  in this setting. Further, in industry, there is an ongoing trend away from minimial instruction sets towards richer "virtual instruction sets". In this setting, our protocols demonstrate their fullest potential. Take, for example, the Ethereum virtual machine (EVM) instruction set which includes custom instructions for Keccak hashing and ECDSA signature verification. In a rich instruction set, ECDSA verification can be represented as a single instruction with size C of 1.5 million constraints <sup>1</sup>. In contrast, compiling ECDSA verification for a minimal instruction set greatly increases the number of executed instructions n; ECDSA verification expands to around 5 million RISC-V instructions <sup>2</sup>. Due to our protocols' succinctness and prover efficiency with respect to C and  $\ell$ , they are especially well suited for the task of proving machine computation for rich instruction sets that include many complex instructions.

Summary of contributions. We summarize our contributions as follows:

- Succinct vector lookup arguments: We present two new vector lookup arguments that provide succinct verification and proof size with respect to vector size, lookup size, and table size. One argument, CosetLkup, encodes vectors within univariate polynomial commitments over cosets of multiplicative subgroups. The other, SubcubeLkup, encodes vectors within multivariate polynomial commitments over subcubes of the boolean hypercube.
- *MuxProofs protocols for machine computation*: We demonstrate the modularity of our approach by combining our vector lookup arguments with three different polyIOPs for NP (with different prover properties) to build zero-knowledge unrolled proof systems for machine computation that only incur prover cost on the order of executed instructions (avoiding universal constraint costs per execution step).
- *Implementation and evaluation*: We implement our univariate vector lookup CosetLkup and evaluate its efficiency. Verifier time is vastly improved over the naive linear combination approach with roughly equal verifier times at vector size 8 and 60× faster for vector size 2<sup>10</sup>. We further perform a thorough cost accounting and estimate performance of our proofs for machine computation.

Along the way, we also contribute the following results of independent interest:

- *Generic zero-knowledge compiler for univariate polyIOPs*: We formalize common techniques for adding zeroknowledge to a sound polyIOP within a generic compiler. To do this, we introduce a *domain admissibility* property for polyIOPs that restricts how polynomial oracles may depend on witness elements and we restrict polynomial evaluations on oracles to particular polynomial identity tests.
- Index-private variant of Marlin polyIOP: Index privacy is a stronger notion of zero-knowledge in which the relation index is also hidden from the verifier. The notion was proposed by Boneh et al. where we identify that Marlin was incorrectly claimed to be index-private [BNO21]. As part of our construction for Mux-Marlin, we propose and prove secure an index-private variant of the Marlin proof system, recovering security for the motivating hiding functional commitments construction of Boneh et al [BNO21].

**Concurrent work.** There exist a number of concurrent works targeting improvements in prover time for proving correct machine execution. The work most closely related to MuxProofs is Sublonk [CGG<sup>+</sup>23] which achieves similar asymptotic prover time and also uses succinct vector lookups (referred to as segment lookups). Sublonk builds a univariate vector lookup from the cq element lookup protocol [EFG22] which admits a prover time independent of the table size  $\ell$ ,  $\tilde{O}(nC)$ , as opposed to  $\tilde{O}((n + \ell)C)$  in this work. Campanelli et al. also propose a succinct vector lookup based on cq (referred to as matrix lookups) [CFF<sup>+</sup>23]. Their protocol is more concretely efficient than that of Sublonk and also includes a zero-knowledge analysis. Even so, constant factors of the cq-based vector lookup are higher than our univariate vector lookup CosetLkup. In the machine computation application, we typically expect the number of executed instructions in a program *n* to eclipse the instruction set size  $\ell$  and so CosetLkup will outperform in this case. Further, Sublonk applies their vector lookup to the PLONK polyIOP for a *layered branching circuit* computation model; their treatment does not directly address certain challenges of the machine computation model such as variable-length execution (e.g., their preprocessing work is dependent on the full execution length). Lastly, both of these works only

<sup>&</sup>lt;sup>1</sup>https://github.com/0xPARC/circom-ecdsa

<sup>&</sup>lt;sup>2</sup>https://github.com/risc0/risc0/tree/v0.16.0/examples/ecdsa

| Protocol                             | Proof size Prover computation |                       | mputation                         | Verifier                               | computation        | U/M                            |   |
|--------------------------------------|-------------------------------|-----------------------|-----------------------------------|--|--------------------|--------------------------------|---|
| Plookup [GW20] + LC                  | $(5+m)\mathbb{G}$             | $9\mathbb{F}$         | $O((n+\ell)\cdot m)\mathbb{G}$    | $\tilde{O}((n+\ell)\cdot m)\mathbb{F}$ | 2P                 | $O(m)\mathbb{G}$               | U |
| cq [ <b>EFG22</b> ] + LC             | $(8+m)\mathbb{G}$             | $3\mathbb{F}$         | $O(n \cdot m)  \mathbb{G}$        | $\tilde{O}(n\cdot m)\mathbb{F}$        | 5P                 | $O(m)  \mathbb{G}$             | U |
| Segment lookup [CGG <sup>+</sup> 23] | $20\mathbb{G}$                | $6\mathbb{F}$         | $	ilde{O}(n \cdot m)  \mathbb{G}$ | $	ilde{O}(n \cdot m)  \mathbb{F}$      | 23P                | -                              | U |
| Matrix lookup [CFF+23]               | $16\mathbb{G}$                | $2\mathbb{F}$         | $O(n \cdot m)  \mathbb{G}$        | $	ilde{O}(n \cdot m)  \mathbb{F}$      | 13P                | -                              | U |
| CosetLkup (this work)                | $28\mathbb{G}$                | $31\mathbb{F}$        | $O((n+\ell)\cdot m)\mathbb{G}$    | $\tilde{O}((n+\ell)\cdot m)\mathbb{F}$ | 2P                 | -                              | U |
| SubcubeLkup (this work)              | $O(\log((n+\ell)$             | $\cdot m))\mathbb{F}$ | -                                 | $O((n+\ell)\cdot m)\mathbb{F}$         | $O(\log((n - C)))$ | $(+\ell)\cdot m))\mathbb{F}/H$ | М |

Figure 2: Comparison of properties of vector lookup proof systems for vectors of length m, tables of  $\ell$  vectors, and lookups of n vectors. The LC annotation denotes the generic linear combination approach to transform any field lookup with linearly-homomorphic commitments into a vector lookup [GW20]. The final column indicates whether the lookup operates over a univariate polynomial encoding (U) or a multivariate polynomial encoding (M). Univariate protocols are instantiated with the Marlin-KZG polynomial commitment [CHM<sup>+</sup>20] over a bilinear pairing group. For cost analysis,  $\mathbb{G}$  denotes group element/multiplication,  $\mathbb{F}$  denotes scalar field element/operation, H denotes a hash operation, and P denotes a pairing operation. Our multivariate protocol is instantiated with a linear-prover polynomial commitment with a logarithmic proof size and verifier time (e.g., Orion [XZS22, CBBZ23]). The  $\tilde{O}$  notation hides polylogarithmic factors. We highlight that SubcubeLkup does not have polylogarithmic factors in the prover time.

build succinct univariate vector lookups. We further propose SubcubeLkup, a succinct multivariate vector lookup, enabling a strictly linear-time prover. We summarize the state of vector lookups in Table 2.

Jolt [AST23] proposes the use of the Lasso lookup for structured large tables [STW23b] to encode and lookup executions from full instruction input-output tables (e.g., for 64-bit RISC-V, tables of size  $2^{128}$ ). In contrast, MuxProofs encodes instruction constraints for looking up which instruction to execute at each step. These are orthogonal (but possibly complementary) applications of lookups for machine computation.

Another line of work achieves the same asymptotic prover efficiency (with very good constants) but does not provide succinctness; proof size and verification time grow linearly in nC [GHK23, YHH<sup>+</sup>23]. Most of the work described so far considers C as the upper bound of instruction constraint size for all instructions in the instruction set, thus incurring overhead when instructions are of varying sizes. Yang et al. consider "tight" machine computation in which the prover only incurs cost on the constraint size of the executed instructions [YHH<sup>+</sup>24]. Instead of leaking the upper bound on the number of executed constraints.

SuperNova [KS22] and Protostar [BC23] achieve the same asymptotic prover time as MuxProofs but take the IVC approach with heuristic security. IVC incurs prover overhead to recursively verify a folding proof: this entails using cycles of elliptic curves and non-native arithmetic constraints [KS23,NBS23]. However, even with these overheads, IVC has been shown to be more prover-efficient than monolithic (i.e., unrolled) proof systems for many settings [NDC<sup>+</sup>24]. Our multivariate protocol Mux-HyperPLONK may offer a promising alternative, as it can be instantiated with prover-efficient polynomial commitments (e.g., Brakedown [GLS<sup>+</sup>23] or Orion [XZS22,CBBZ23]) that do not provide required homomorphism for efficient folding in SuperNova and Protostar.

A previous version of this paper presented a different univariate succinct vector lookup protocol based on the Plookup element lookup [GW20]. It provides the same asymptotics as CosetLkup but with worse constant factors. For completeness, we include the construction in Appendix F.

# 2 Technical Overview

A standard approach to constructing succinct zero knowledge proof systems employs *holography* in which the claimed computation to be proved is encoded within a *computation commitment* in an initial preprocessing phase [CHM<sup>+</sup>20, Set20]. After checking the validity of the computation commitment once—a non-succinct operation that can take time linear in the size of the computation—the verifier can verify any number of proofs for the computation succinctly. Unfortunately in machine execution, the description of the unrolled executed computation of a program (i.e., the sequence of executed instructions) is dependent on the program and program inputs. Thus, a different computation commitment and verifier check would be required for each different program execution. Not only does this approach not result in succinct verification but it is also not amenable to zero-knowledge: the executed computation description

may leak information about the program and its inputs.

We describe below an overview of our strategy for constructing the first zero-knowledge argument for unrolled machine execution with prover-efficiency on the order of the executed instructions that avoids recursive proving techniques. We describe two main technical contributions (Sections 2.2 and 2.3, respectively). The first contribution is a new building block, a succinct vector lookup argument, for efficiently proving correspondence of vector encodings between two polynomials. The second contribution is to show how to compose vector lookup arguments with holographic polyIOPs to realize succinct and prover-efficient proofs for machine computation.

#### 2.1 Strategy: Computation Commitments from Machine Commitments

Despite the executed computation being program-dependent, there exists structure in the computation that we can take advantage of. Namely, the set of possible instructions that can be executed is fixed ahead of time as a description of the "machine" the program runs on (e.g., a RISC-V CPU has a fixed instruction set). In our work, during preprocessing, the machine description (i.e., instruction set) is encoded within a *machine commitment*. To prove machine executed instructions) can be computed on-the-fly in such a way that the verifier can succinctly verify correctness of the computation commitment given the machine commitment. Then given the computation commitment, we can largely rely on previous techniques to verify correctness of computation execution. As we describe next, the core insight of our work is a new way to encode computation descriptions to enable efficient proofs for the relation between the executed computation commitment and machine commitment.

**Modeling machine execution.** First, we provide an introduction to our model of machine execution. We say a machine description consists of  $\ell$  instructions each of which are represented as a computation over an input state  $(inst_{in}, mem_{in})$  and produce an output state  $(inst_{out}, mem_{out})^3$ . The output state  $(inst_{out}, mem_{out})$  is passed as input to the next instruction. There are two parts to the running state. First, the instruction pointer  $inst \in [\ell]$  specifies which of the  $\ell$  instructions to run next. We assume that the instruction computation checks that the instruction pointer in the input state is correct. Second, the memory mem contains all other state including program inputs, program description, program counter, and external memory. As such, in applying an instruction computation to move from  $(inst_{in}, mem_{in})$  to  $(inst_{out}, mem_{out})$ , our modeling of an instruction computation captures two possibly distinct functionality: (1) The instruction functionality applying changes to external memory (e.g., storing the sum of two values in the case of an "add" instruction), and (2) the control logic functionality determining the next instruction to run (e.g., changing the program counter according to inputs and reading the program description to determine the next instruction pointer).

In practice, it will not be desirable to pass the full memory as described into each instruction computation. *Offline memory checking* techniques [BEG<sup>+</sup>91] enable a verifier to efficiently check a prover maintains memory correctly by performing a small amount of work per memory access (e.g., a Merkle path check or a multiset hash update) [SAGL18, KPPS20, LNS20, GHK23, YH23]. Offline memory checking is not a contribution of this work, and any of these existing techniques can be employed with MuxProofs. In the remainder of the paper, we will consider *mem* as a small digest (e.g., a Merkle root or a multiset hash) and appropriate witnesses are provided for checking memory accesses within the instruction computation.

Encoding a machine commitment as a polynomial. We now return to our goal of encoding a machine description as a machine commitment in a useful manner. Among prior proof systems that employ holography [GWC19, CHM<sup>+</sup>20, CBBZ23], a predominant approach to encoding the computation is as a vector (or small number of vectors) containing elements of a field  $\mathbb{F}$ . The vector, say of length m, is then encoded as the evaluation points of a polynomial over some specified domain. Some proof systems encode as a univariate polynomial  $f \in \mathbb{F}^{\leq m}[X]$  of degree m where f is interpolated over evaluations of a canonical ordered subgroup  $\mathbb{H} \subseteq \mathbb{F}$  [GWC19, CHM<sup>+</sup>20]. Others encode as a  $(\log m)$ -multivariate polynomial  $f \in \mathbb{F}[X_{[\log m]}]$  where f is the multilinear polynomial interpolated over evaluations of

<sup>&</sup>lt;sup>3</sup>There are different models for computation. For example, if modeled using circuit satisfiability, an instruction circuit would take in  $(inst_{in}, mem_{in}, inst_{out}, mem_{out})$  as well as possibly some other witness inputs such that the circuit is satisfied if and only if  $(inst_{out}, mem_{out})$  is a valid application of the instruction computation to  $(inst_{in}, mem_{in})$ .

the boolean hypercube  $\{0,1\}^{\log m}$  [Set20, CBBZ23]. This preprocessing of computation commitments as polynomials is used by a popular class of proof systems known as *polynomial interactive oracle proofs* (polyIOPs) [BFS20]. We will model our machine commitment in the same way. For now, let us focus on the univariate polynomial setting; we will revisit the multivariate polynomial setting which admits various tradeoffs shortly.

Say each instruction can be described by a vector of field elements of size m. By packing the instruction vectors into a larger vector of size  $\ell m$ , we can encode the full instruction set into the evaluations of a polynomial t over a subgroup  $\mathbb{H}$  of size  $|\mathbb{H}| = \ell m$ . Looking forward, a key insight to enable our efficient proof techniques is the manner in which we perform this encoding. In particular, we encode each instruction over a size-m coset of  $\mathbb{H}$  that admits useful structure. This will allow us to prove more granular properties at the level of certain instructions rather than being limited to simply proving properties about the full instruction set. More precisely, say  $\mathbb{H} = \langle \omega \rangle$  is generated by generator  $\omega \in \mathbb{F}$ :  $\mathbb{H} = \{1, \omega, \omega^2, \dots, \omega^{\ell m - 1}\}$ . Then, we define a multiplicative subgroup  $\mathbb{V} \leq \mathbb{H}$  of size  $|\mathbb{V}| = m$  where  $\mathbb{V}$  is generated by  $\gamma = \omega^{\ell}$ :

$$\mathbb{V} = \left\{ 1, \gamma = \omega^{\ell}, \gamma^2 = \omega^{2\ell}, \dots, \gamma^{m-1} = \omega^{(m-1)\ell} \right\}$$

Further, we define the  $\ell$  cosets of  $\mathbb{V}$  in  $\mathbb{H}$  as

$$\forall i \in [0,\ell), \quad \omega^i \mathbb{V} = \left\{ \omega^i, \omega^i \gamma = \omega^{\ell+i}, \omega^i \gamma^2 = \omega^{2\ell+i}, \dots, \omega^i \gamma^{m-1} = \omega^{(m-1)\ell+i} \right\}$$

In this way, we interpolate polynomial t for the machine commitment such that the evaluations on coset  $\omega^i \mathbb{V}$  are set to the vector of field elements that describe the computation for the  $i^{th}$  instruction.

**Building an executed computation commitment via a lookup argument.** Now given a polynomial t that encodes the set of  $\ell$  instructions as a machine commitment, our goal is to produce a computation commitment polynomial for the unrolled execution. An unrolled execution consists of applying n instruction computations in sequence where n is the number of execution steps until program termination.

At a high level, we want to be able to produce a polynomial f interpolated over a subgroup  $\mathbb{G}$  (where  $|\mathbb{G}| = mn$ and generator  $\mathbb{G} = \langle \mu \rangle$ ) that encodes the n executed instructions. Analogous to our encoding of  $\ell$  instructions from the instruction set in the machine commitment polynomial t, we encode the n executed instructions in f as evaluations over the n cosets of  $\mathbb{V}$  in  $\mathbb{G}$ . More precisely, each coset of f should correspond to some instruction encoded over a coset of t:  $\forall j \in [n] \; \exists i \in [\ell] \; \text{s.t.} \; f(\mu^j \mathbb{V}) = t(\omega^i \mathbb{V}).$ 

This type of relation can be abstracted as a *vector lookup*: we would like to prove that polynomial f faithfully "looks up" vectors encoded in the instruction table polynomial t. That is, only valid instructions are encoded. The standard approach for vector lookups build on lookup protocols for individual field elements [BCG<sup>+</sup>18, GW20, ZBK<sup>+</sup>22, PK22, ZGK<sup>+</sup>22, GK22, EFG22]. An element lookup allows proving the simpler relation that every evaluation of a polynomial  $f_1$  over  $\mathbb{G}$  exists in the evaluations of the table polynomial  $t_1$  on  $\mathbb{H}$ . To build a vector lookup from an element lookup, one may encode each position  $i \in [m]$  of the instruction vectors within a different polynomial, resulting in polynomials  $[f_i]_{i \in [m]}$  and  $[t_i]_{i \in [m]}$ . If the polynomials are committed using a linearly-homomorphic commitment scheme, the verifier can sample a random challenge  $\beta \leftarrow \mathbb{F}$  to randomly combine the position commitments. The prover and verifier jointly compute (commitments to) to polynomials  $\hat{f} = \sum_{i \in [m]} \beta^i \cdot f_i$  and  $\hat{t} = \sum_{i \in [m]} \beta^i \cdot t_i$ , and perform an element lookup with respect to  $\hat{f}$  and  $\hat{t}$  [GW20].

This approach to vector lookups incurs verification cost and proof size that is at least linear in the vector size m. Recall in the machine computation application, we are proposing encoding instruction constraints within a vector. At the very least, the simplest instructions encoding 64-bit arithmetic requires constraints on the order of 128; in richer instruction sets, such as EVM, instructions can be much larger (e.g., 1.5 million for ECDSA verification or 20000 for SHA256). Linear scaling in m of the vector lookup is at best a significant overhead and at worst is a prohibitive road block to richer instruction sets. Our first key technical contribution is the construction of new *succinct* vector lookup arguments for univariate and multivariate polynomials (Section 2.2) where proof size and verfier cost is succinct in n,  $\ell$ , and vector size m.

The vector lookup proves the computation commitment f indeed includes encodings of valid instructions. Ideally,

we would be able to directly apply an existing proof system to f to prove the validity of the executed computation. However, there are two additional hurdles to overcome (Section 2.3). First, f is constructed as a stitching together of the computation encodings for each of the individual n executed instructions. It is not necessarily the case (and in fact not the case for existing proof systems) that a direct stitching together of the "local" instruction computation encodings results in a valid computation encoding for the "global" sequence of instructions; it may be the case that some global structure is required in the computation commitment. Nevertheless, we provide a protocol for adapting fto f' to recover the global structure required in three existing polyIOPs, PLONK [GWC19], Marlin [CHM<sup>+</sup>20], and HyperPLONK [CBBZ23].

Lastly, applying a polyIOP directly to f' would not quite meet our succinctness goal. Recall, the verifier input to each instruction computation is  $(inst_{in}, mem_{in}, inst_{out}, mem_{out})$ . Thus, to verify the full executed computation, the verifier will need  $[inst_j, mem_j]_j^n$  where the statement for the  $j^{th}$  instruction is  $(inst_j, mem_j, inst_{j+1}, mem_{j+1})$ . Instead, to enable succinctness, the verifier will hold only the input state to the first instruction  $(inst_0, mem_0)$  and the output state of the last instruction  $(inst_n, mem_n)$ . We provide a protocol to prove the wellformedness of the intermediate instruction states, i.e., that the output state from instruction j is the same as the input state to instruction j + 1.

#### 2.2 Contribution: Succinct Vector Lookup Arguments

The constructions we propose, CosetLkup for the univariate polynomial case and SubcubeLkup for the multivariate polynomial case, both derive from the following technical lemma of Haböck [Hab22, Section 3.4] (reformulated in [BC23, Section 4.4]):

**Lemma 1** (informal). Suppose  $\left[ [f_{i,j}]_{j \in [m]} \right]_{i \in [n]}$  and  $\left[ [t_{i,j}]_{j \in [m]} \right]_{i \in [\ell]}$  are sequences of element vectors in field  $\mathbb{F}$ . Then, the vector lookup relation  $\left\{ \left\{ f_{i,j} \right\}_{j \in [m]} \right\}_{i \in [n]} \subseteq \left\{ [t_{i,j}]_{j \in [m]} \right\}_{i \in [\ell]}$  holds if and only if there exists a sequence of field elements  $[c_i]_{i \in [\ell]}$  such that the following equality holds over the rational function field  $\mathbb{F}(X,Y)$ :

$$\sum_{i \in [n]} 1/(X + \sum_{j \in [m]} f_{i,j}Y^j) = \sum_{i \in [\ell]} c_i/(X + \sum_{j \in [m]} t_{i,j}Y^j).$$

To achieve this equality, the field elements  $[c_i]_{i \in [\ell]}$  are set to the counts that vector  $t_i$  appears in f. Intuitively, this lemma represents the logarithmic derivative of the polynomial equality  $\prod_{i \in [n]} (X + \sum_{j \in [m]} f_{i,j}Y^j) = \prod_{i \in [\ell]} (X + \sum_{j \in [m]} t_{i,j}Y^j)^{c_i}$ . Our protocols check this equality by evaluating the rational functions on random verifier challenges  $\alpha, \beta \leftarrow \mathbb{F}^2$ . The challenge then is proving this equality succinctly to a verifier given the vector encodings.

$$\sum_{i \in [n]} 1/(\alpha + \sum_{j \in [m]} \beta^j \cdot f_{i,j}) = \sum_{i \in [\ell]} c_i/(\alpha + \sum_{j \in [m]} \beta^j \cdot t_{i,j}).$$

Consider again the univariate polynomial encoding of f with vectors encoded within cosets. We will build up to two polynomials  $U_f$  and  $U_t$  that encode the left and right sides of the equality expression above, respectively. Without loss of generality, consider the left side of the equality dealing with f where  $f_{i,j} = f(\mu^i \gamma^j)$ . That is,  $U_f$  will encode as its evaluations over  $\mathbb{G}$ :

$$\left[ \left[ U_f(\mu^i \gamma^j) = 1/(\alpha + \sum_{k \in [m]} \beta^k \cdot f(\mu^i \gamma^k)) \right]_{i \in [n]} \right]_{j \in [n]}$$

Given  $U_f$  and an analogously-encoded  $U_t$ , the final equality is checked using a polynomial identity test known as a sum check, in which we compare the sum of  $U_f$  over  $\mathbb{G}$  and  $U_t$  over  $\mathbb{H}$  (see Appendix A).

Now let us provide some details on how  $U_f$  is built up. The main challenge is proving that the summation in the denominator of  $U_f$  correctly encodes the elements of each vector. To do this, first consider the following helper polynomials  $I_f$  and  $S_f$ . Polynomial  $S_f$  directly encodes the claim summation for each vector within the coset for that vector. Polynomial  $I_f$  encodes the m powers-of- $\beta$  in each coset.

$$\left[\left[I_f(\mu^i\gamma^j) = \beta^j\right]_{i \in [n]}\right]_{j \in [m]}, \qquad \left[\left[S_f(\mu^i\gamma^j) = \sum_{k \in [m]} \beta^k \cdot f(\mu^i\gamma^k)\right]_{i \in [n]}\right]_{j \in [m]}$$

The encodings of  $I_f$  and  $S_f$  are proved to the verifier again using standard polynomial identities. In this case, zero test protocols check that the following identities hold over some subgroup (see Appendix A):

- $I_f(1) = 1$ : The first element of  $I_f$  is anchored to equal to 1.
- (I<sub>f</sub>(γX) − β · I<sub>f</sub>(X))(X − γ<sup>m−1</sup>) = 0 over V: The term (I<sub>f</sub>(γX) − β · I<sub>f</sub>(X)) enforces the next element in the coset V (generated by γ) is equal to β times the previous element. The last term (X − γ<sup>m−1</sup>) excludes the last element which would carry-over to the first element: β<sup>m−1</sup> · β ≠ 1. Since the first element was anchored to 1 and the last element is excluded, this sets the evaluations of V to be equal to the powers of β.
- (I<sub>f</sub>(μX) − I<sub>f</sub>(X)) · Z<sub>μ<sup>n-1</sup>V</sub>(X) = 0 over G: The first term (I<sub>f</sub>(μX) − I<sub>f</sub>(X)) enforces the next element in G is equal to the previous element in G. The second term Z<sub>μ<sup>n-1</sup>V</sub>(X) excludes the last coset where Z is the "vanishing polynomial" that evaluates to 0 on μ<sup>n-1</sup>V. These checks ensure that the j<sup>th</sup> element of every coset is the same. Since we know the powers of β are encoded in coset V, this check enforces that the same powers are propagated to the other cosets in G.

To prove the wellformedness of  $S_f$ , we introduce another helper polynomial  $B_f$  which encodes the partial summations of  $S_f$  and builds up to the claimed summation inductively:

$$\left| \left[ B_f(\mu^i \gamma^j) = \sum_{k \in [j]} \left( \beta^k \cdot f(\mu^i \gamma^k) - \frac{S_f(\mu^i \gamma^j)}{m} \right) \right]_{i \in [n]} \right|_{j \in [n]}$$

Then the polynomial identities to complete the wellformedness verification of  $S_f$  are as follows:

- $S_f(\gamma X) = S_f(X)$  over G: Within a coset, the same claimed summation is encoded throughout, i.e., all of the evaluations over a coset are a constant (the respective summation).
- $B_f(\gamma X) = B_f(X) + I_f(\gamma X) \cdot f(\gamma X) \frac{S_f(X)}{m}$  over G: The inductive statement enforces that the next element in the partial summation sums the previous partial summation with the contribution of the next element in the coset, namely  $\beta^k \cdot f(\mu^j \gamma^k)$  where the next power of  $\beta$  is encoded within  $I_f$ . Lastly, the normalized claimed summation,  $\frac{S_f(X)}{m}$ , is subtracted for every element in the coset. If the claimed sum is correct, then these subtractions will exactly cancel out with the true sum over the full coset.

Finally, given  $S_f$ , one last polynomial identity is used to prove the form of  $U_f$ :  $U_f(X) \cdot (\alpha + S_f(X)) = 1$  over  $\mathbb{G}$ . Our use of coset encodings and of the subgroup generator  $\gamma$  to traverse cosets was critical in creating polynomial identities that can efficiently check this structure. Section 4 (Figure 5) presents the full CosetLkup protocol. All together, when instantiated with the Marlin-KZG polynomial commitment scheme [CHM<sup>+</sup>20], CosetLkup admits a quasilinear prover, a constant-size proof, and constant pairing (and logarithmic field operations) verifier (see Figure 2).

**Extending to the multivariate setting.** Multivariate polyIOPs offer certain benefits over their univariate counterparts. By encoding vectors as multilinear polynomials over evaluations of the boolean hypercube, polynomial identities can be checked via the linear-time and concretely efficient sum check protocol [LFKN92] avoiding the quasilinear proving time in the univariate setting resulting from the need to perform Fast Fourier Transforms (FFTs). These benefits come at the expense of larger logarithmic-sized (or larger depending on the polynomial commitment) proofs and verifier time. As shown before, the choice of encoding of vectors within the polynomial is important in enabling a succinct argument. Before, in the univariate setting, we chose to encode vectors within cosets of a multiplicative subgroup. For the multivariate setting, we encode vectors within subcubes of the boolean hypercube. Consider  $(\log(mn))$ -variate polynomial f encoding n vectors of length m over the  $(\log(mn))$ -dimension boolean hypercube  $\{0,1\}^{\log(mn)}$  The trailing  $\log n$  bits select the vector and the leading  $\log m$  bits select the vector position. Thus, for  $[[f_{i,j}]_{j \in [m]}]_{i \in [n]}: [[f(j,i) = f_{i,j}]_{j \in \{0,1\}^{\log m}}]_{i \in \{0,1\}^{\log n}}.$  The table polynomial t is encoded analogously over boolean hypercube  $\{0,1\}^{\log(m\ell)}$ . We present a succinct vector lookup protocol SubcubeLkup for this subcube vector encoding over multivariate polynomials. It follows the same blueprint described above building polynomials  $I_f, S_f, B_f, U_f$ (respectively,  $U_t$ , etc.) and completes by checking the equality of Lemma 1 via a sum check over  $U_f$  and  $U_t$ . Our key trick to achieving succinctness in the univariate setting was using the coset substructure and defining polynomial identities that traverse a coset using the subgroup generator  $\gamma$ .

Recently, Diamond and Posen proposed new techniques that enable traversing subcubes of the boolean hypercube in an analogous manner [DP23]. They define a shift operator  $\widetilde{shft}_b(f)$  that takes a  $\mu$ -variate polynomial f, is parameterized by a subcube size b, and outputs a polynomial shifted by the subcube. That is, for all  $i \in \{0,1\}^{\mu-b}$  and for all  $j \in \{0,1\}^{b}$ , it holds that  $shft_b(f)(j,i) = f(bin_b(int_b(j) + 1 \mod 2^b), i)$  where  $int_b$  and  $bin_b$  map back and forth integers  $[2^b]$  and boolean vectors  $\{0,1\}^b$ . Using this operator, we can again define succinctly verifiable polynomial identities to check the wellformedness of  $U_f$  and  $U_t$ .

To illustrate this, consider the powers-of- $\beta$  polynomial  $I_f$ . Now  $I_f(X_{\lfloor \log(mn) \rfloor})$  is a  $(\log(mn))$ -variate polynomial encoding the *m* powers of  $\beta$  in the boolean subcubes of its leading  $\log m$  bits:  $\left[ \left[ I_f(j,i) = \beta^{\mathsf{int}_{\log m}(j)} \right]_{j \in \{0,1\}^{\log m}} \right]_{i \in \{0,1\}^{\log m}}$ 

The following polynomial identities checked via multivariate zero tests [Set20, CBBZ23] over  $\{0,1\}^{\log(mn)}$  verify the wellformedness of  $I_f$ :

- $(I_f(X_{\lfloor \log(mn) \rfloor}) 1)(\widetilde{eq}_{\log m}(X_{\lfloor \log m \rfloor}, 0_{\lfloor \log m \rfloor})) = 0$ : The first term checks that  $I_f$  is anchored to 1. The second term  $\widetilde{eq}_{\log m}(X_{\lceil \log m \rceil}, 0_{\lceil \log m \rceil})$  enforces this check only for the first position  $(0_{\lceil \log m \rceil})$  of each subcube. The polynomial  $\tilde{eq}_{\log m}$  takes in two boolean vectors of length  $\log m$  and outputs 1 if they are equal and 0 otherwise.
- $\left(\widetilde{shft}_{\log m}(\widetilde{I}_b)(X_{\lfloor \log(md_b) \rfloor}) \beta \cdot \widetilde{I}_b(X_{\lfloor \log(md_b) \rfloor})\right) \cdot (1 \widetilde{eq}_{\log m}(X_{\lfloor \log m \rfloor}, 1_{\lfloor \log m \rfloor})) = 0$ : The first term enforces • the next element in the subcube (generated by  $\widetilde{shft}$ ) is equal to  $\beta$  times the previous element. The last term excludes the last element of the subcube  $(1_{\lceil \log m \rceil})$  preventing carry-over:  $\beta^{m-1} \cdot \beta \neq 1$ . As before, since the first element was anchored to 1 and the last element is excluded, this sets the evaluations of every subcube to be equal to the powers of  $\beta$ .

The full details of SubcubeLkup are given in Appendix C (Figure 21). When instantiated with an appropriate polynomial commitment scheme, it enables an efficient linear-time prover with sublinear proof size and verifier (see Figure 2).

#### **Contribution: Vector Lookups for Machine Execution** 2.3

We described our high level strategy of dynamically creating a computation commitment to only executed instructions through a vector lookup argument on the machine commitment table of valid instructions. There are two further challenges to overcome to apply existing polyIOP-based proof systems to this computation commitment: (1) the computation commitment may require some global structure that is lost by stitching together computation commitments for individual instructions, and (2) the NP statement for this computation commitment is not necessarily succinct.

Recovering global structure of the computation commitment. To illustrate the issue of global structure, let us examine the structure of a computation commitment for a specific polyIOP, PLONK [GWC19]. PLONK is a polyIOP for the "Plonkish" arithmetization which is a natural encoding of computation in NP with a circuit-like structure [STW23a]. Consider a simplified example of Plonkish for an arithmetic circuit with m gates. The computation trace is encoded as a vector of wire values  $z \in \mathbb{F}^{3m}$ :

$$z = \left[ (z_0^{(l)}, z_0^{(r)}, z_0^{(o)}), (z_1^{(l)}, z_1^{(r)}, z_1^{(o)}), \dots, (z_{m-1}^{(l)}, z_{m-1}^{(r)}, z_{m-1}^{(o)}) \right].$$

We denote the ordering of vector z such that  $(z_{3i}, z_{3i+1}, z_{3i+2}) = (z_i^{(l)}, z_i^{(r)}, z_i^{(o)})$  correspond to the left, right, and output wires of gate  $i \in [m]$ , respectively. The computation is encoded by two vectors,  $sel \in \mathbb{F}^m$  and  $\sigma \in \mathbb{F}^{3m}$ . The selector vector sel specifies the gate type by encoding 1 at index i if gate i is an addition gate and 0 for a multiplication gate. The copy vector  $\sigma$  specifies the connections of wires between gates by encoding a permutation of the indices [3m] (i.e.  $\{\sigma_i\}_{i \in [3m]} = \{i \in [3m]\}$ ). The copy vector is constructed such that wires that are connected have permuted indices. A computation trace satisfies a computation encoding if: • *Gate constraints*:  $\forall i \in [m], sel_i \cdot (z_i^{(l)} + z_i^{(r)}) + (1 - sel_i) \cdot z_i^{(l)} \cdot z_i^{(r)} = z_i^{(o)}.$ 

- Copy constraints:  $\forall i \in [3m], z_i = z_{\sigma_i}$ .

In the univariate polyIOP PLONK [GWC19],  $\bar{sel} \in \mathbb{F}^m[X]$  and  $\bar{\sigma} \in \mathbb{F}^{3m}[X]$  are interpolated as polynomials fixing their evaluations over subgroups of appropriate size. Commitments to these polynomials form the computation commitment.

In the machine computation setting, each instruction  $i \in [\ell]$  in the instruction set is represented by a pair of vectors  $[sel_i \in \mathbb{F}^m, \sigma_i \in \mathbb{F}^{3m}]_{i \in [\ell]}$ . Following our vector lookup strategy, these vectors are encoded in table polynomials tseland  $t\sigma$  where each vector is encoded in cosets of V within  $\mathbb{H} = \langle \omega \rangle$ . Assume, for simplicity, that *tsel* and  $t\sigma$  use the same evaluation subgroups  $\mathbb{V}$  and  $\mathbb{H}$  (where  $|\mathbb{V}| = 3m$  and  $|\mathbb{H}| = 3m\ell$ ), e.g., that *tsel* repeats each gate selector three times to pad out. For a machine computation that executes n instructions  $[inst_{i \in [\ell]}]_{i \in [n]}$ , we define two new polynomials sel and  $\bar{\sigma}$  that encode the executed instruction vectors in the cosets  $\mathbb{V}$  within  $\mathbb{G} = \langle \mu \rangle$  (where  $|\mathbb{G}| = 3mn$ ). The form of these polynomials is proved exactly using a vector lookup as demonstrated before:

$$\left[\bar{sel}(\mu^{i}\mathbb{V}) = tsel(\omega^{inst_{i}}\mathbb{V})\right]_{i \in [n]}, \qquad \left[\bar{\sigma}(\mu^{i}\mathbb{V}) = t\sigma(\omega^{inst_{i}}\mathbb{V})\right]_{i \in [n]}$$

If we examine the resulting selector polynomial sel defined over  $\mathbb{G}$ , we find that it fits the form needed for the PLONK polyIOP. The *m* gate selections for each of the *n* executed instructions are all encoded within sel.

However, now consider the resulting copy polynomial  $\bar{\sigma}$  defined over G. Recall that the PLONK polyIOP expects a permutation over |G| = 3mn encoded within  $\bar{\sigma}$ . This is not the case for  $\bar{\sigma}$ ; the global permutation structure is damaged. It is true that each instruction vector encodes a permutation over [3m], and thus by the vector lookup, the evaluations of each coset of  $\mathbb{V}$  in  $\bar{\sigma}$  encode a permutation over [3m]. Fortunately, we can recover the global permutation over [3mn] by offsetting the permutation encoded in each coset  $i \in [n]$  over [3m] by 3mi. Define the offset copy polynomial  $\bar{\sigma}'$  and the offset polynomial s over G:

$$\left[\bar{\sigma}'(\mu^i \mathbb{V}) = 3mi + \bar{\sigma}(\mu^i \mathbb{V})\right]_{i \in [n]}, \qquad \left[s(\mu^i \mathbb{V}) = 3mi\right]_{i \in [n]}$$

The prover uses the following polynomial identities to succinctly prove that the offset was performed correctly:

- s(X) = 0 over  $\mathbb{V}$ : The first coset is anchored to evaluate to 0; there is no offset needed.
- (3m+s(X) − s(µX))(Z<sub>µn-1V</sub>(X)) = 0 over G: The first term enforces the next element in G is equal to the previous element offset by an additional 3m. The second term excludes enforcing an additional offset from the last coset to the first coset. Since we know zero is encoded in coset V, these checks enforce that each coset in sequence offsets by an additional 3m.
- $\bar{\sigma}'(X) = \bar{\sigma}(X) + s(X)$  over G: The offset copy polynomial adds the correct encoded offsets.

With the recovered global structure, the prover has now on-the-fly generated sel and  $\bar{\sigma}'$  which encode the executed machine computation and proved their correctness with respect to the machine commitments tsel and  $t\sigma$ . The polyIOP PLONK can be applied directly.

**Compressing the NP statement of the computation commitment.** Our last challenge is to compress the statement for the unrolled computation commitment to allow for succinct verification. Naively, the statement for the unrolled computation commitment consists of the statements for each executed instruction:  $[(inst_{in,i}, mem_{in,i}, inst_{out,i}, mem_{out,i})]_{i \in [n]}$ . Not only does this prevent succinct verification, but it also prevents zero-knowledge of intermediate program execution state. We address this by observing that the verifier does not need to have the intermediate program state; it is sufficient for the verifier to simply check that the intermediate program state is passed correctly between instructions. That is, that  $(inst_{out,i}, mem_{out,i}) = (inst_{in,i+1}, mem_{in,i+1})$  for all  $i \in [n]$ . Then the verifier only need hold the starting state  $(inst_{in,0}, mem_{in,0})$  and the ending state  $(inst_{out,n}, mem_{out,n})$ .

In PLONK, the statement and witness for the unrolled computation are encoded together in a polynomial z (referred to as the extended witness) defined over  $\mathbb{G}$ . In which, the statement and witness for each instruction are encoded within a corresponding coset of  $\mathbb{V}$  in  $\mathbb{G}$ : for instruction  $i \in [n]$ , the evaluation of  $z(\mu^i \mathbb{V}) = (inst_{in,i}, mem_{in,i}, inst_{out,i}, mem_{out,i}, w_i)$  for a corresponding witness vector  $w_i$ . More precisely, we define a subgroup  $\mathbb{V}_x = \langle \psi \rangle \subset \mathbb{V}$  where  $|\mathbb{V}_x| = 2 \cdot (|inst| + |mem|)$  whose cosets will encode the statement for each executed instruction. Further, we define subgroup  $\mathbb{V}_{in} \subset \mathbb{V}_x$  and it's single coset  $\mathbb{V}_{out}$  such that  $\mathbb{V}_{in} \cup \mathbb{V}_{out} = \mathbb{V}_x$  and  $|\mathbb{V}_{in}| = |\mathbb{V}_{out}| = |\mathbb{V}_x|/2$ . These will encode the input and output states of each executed instruction:  $\forall i \in [n]$ ,

$$\left[z(\mu^{i}\mathbb{V}_{in}) = \left[\operatorname{inst}_{in,i}, \operatorname{mem}_{in,i}\right]\right]_{i \in [n]} \quad \left[z(\mu^{i}\mathbb{V}_{out}) = \left[\operatorname{inst}_{out,i}, \operatorname{mem}_{out,i}\right]\right]_{i \in [n]}$$

Thus, proving consistency of intermediate program states, again, reduces to proving polynomial identities over certain cosets. Namely, for  $i \in [1, n]$ ,  $z(\mu^i \mathbb{V}_{in}) = z(\mu^{i-1} \mathbb{V}_{out})$ . The above coset equality constraints can be written as the polynomial identity:  $z(X) = z(\mu^{-1}\psi X)$  over  $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$  where  $\mathbb{G}_{in} = \bigcup_{j=0}^{n-1} \mu^j \mathbb{V}_{in}$ . The full details of proving input-output correspondence and our full adaption of PLONK to machine computation, Mux-PLONK, is given in Section 5.

**Extending to other polyIOPs.** The general recipe that we described using vector lookups to build Mux-PLONK is quite modular. We demonstrate the generality by extending two other polyIOPs for use with machine computation, each offering various tradeoffs.

We build Mux-HyperPLONK building off of the multivariate HyperPLONK polyIOP [CBBZ23]. As motivated earlier, multivariate polyIOPs enable linear-time provers and use of the efficient sum check protocol [LFKN92] for proving polynomial identities. In Mux-HyperPLONK, we employ our new multivariate succinct vector lookup SubcubeLkup and show how to translate the PLONK global permutation recovery and input-output consistency checks to the multivariate setting. Details are given in Appendix D.

Next, we build Mux-Marlin building off the univariate Marlin polyIOP [CHM<sup>+</sup>20]. Whereas, Mux-PLONK and Mux-HyperPLONK encode instructions using the Plonkish arithmetization, Marlin uses a rank-1 constraint system (R1CS) arithmetization which offers encoding tradeoffs [STW23a]. The full details are given in Appendix E. An interesting open question is defining formally the properties needed of a polyIOP to work with our vector lookup "compiler" for machine execution. One aspect of this is the global structure of the computation commitment. Another aspect, for zero-knowledge, is that the polyIOP must enable zero-knowledge of the computation commitment, a property not typically required as the computation is public preprocessed information. It turns out that Marlin [CHM<sup>+</sup>20] does not provide this property. In Appendix E, we construct a variant of Marlin to recover this property, and, by doing so, also recover the security of a construction for hiding functional commitments [BNO21].

All together, we build three new succinct proof systems for unrolled machine execution. They all have prover costs that scale quasilinearly (linearly in the case of Mux-HyperPLONK) with the constraints of the executed instructions (see cost overview in Figure 1) and avoid the heuristic security assumptions required of IVC approaches.

## **3** Preliminaries

#### 3.1 Notation

<u>Sets and vectors.</u> For a positive numbers m and n with m < n, let [m, n] denote the vector  $[m, \ldots, n-1]$  and [n] be shorthand for [0, n]. We use  $[\cdot]$  and  $(\cdot)$  to denote ordered vectors,  $\{\cdot\}$  to denote sets, and  $\{\!\{\cdot\}\!\}$  to denote a multiset. Any of these operators can be expanded via a subscript, i.e.,  $[a_i]_{i=1}^n = [a_1, \ldots, a_n]$ .

<u>Fields</u>, groups, and polynomials. Define  $\mathbb{F}$  to be a scalar field of large prime order p. We will use  $\mathbb{H}$  and  $\mathbb{V}$  (and various subscripts) to denote multiplicative subgroups of  $\mathbb{F}^*$ . We may denote a generator  $\gamma$  for a subgroup as  $\mathbb{H} = \langle \gamma \rangle$ . We will also denote subgroups to be subgroups of each other, say  $\mathbb{V}$  is a subgroup of  $\mathbb{H}$ , denoted  $\mathbb{V} \leq \mathbb{H}$ . We require that all multiplicative groups we use are FFT-friendly and have smooth sizes [BCG<sup>+</sup>13], i.e., are a power of two. That is, we want  $2^L | p - 1$  for some large integer L, so each divisor of  $2^L$  (every power of 2 less than  $2^L$ ) gives exactly one subgroup whose order is the divisor by Lagrange's theorem. Many common curves support these properties including BN382 and BLS12-381 [AHG22].

When  $\mathbb{V}$  is a subgroup of  $\mathbb{H}$ , we will make use of the cosets of  $\mathbb{V}$  in  $\mathbb{H}$ . A coset of  $\mathbb{V}$  is defined by a field element offset  $a \in \mathbb{F}$  as  $\{av : v \in \mathbb{V}\}$ , which we may denote as  $a\mathbb{V}$ . For multiplicative subgroup  $\mathbb{H} = \langle \omega \rangle, \mathbb{V}$  where  $\mathbb{V} \leq \mathbb{H}$ ,  $|\mathbb{H}| = nm$ , and  $|\mathbb{V}| = m$ , then  $[\omega^i \mathbb{V}]_{i \in [n]}$  forms the *n* distinct cosets of  $\mathbb{V}$  in  $\mathbb{H}$ .

Let  $\mathbb{F}^{\leq d}[X_{[\mu]}]$  be the set of  $\mu$ -variate polynomials in indeterminate  $X_0, \ldots, X_{\mu-1}$  with coefficients in  $\mathbb{F}$  with degree less than or equal to d. Similarly, let  $\operatorname{Func}^{\mathbb{F}}[X_{[\mu]}]$  be the set of  $\mu$ -variate functions over  $\mathbb{F}$ . We use  $X_{[\mu]}$  as shorthand for expanding  $X_0, \ldots, X_{\mu-1}$ . For polynomials  $f \in \mathbb{F}[X_{[\mu]}]$  and some evaluation domain D, we use f(D) as shorthand for expanding the vector  $[f(d)]_{d \in D}$ .

<u>Univariate polynomials.</u> For an arbitrary set S, let the vanishing polynomial for S be  $Z_S(X) = \prod_{s \in S} (X - s)$  such that it evaluates to 0 for  $s \in S$ . A Lagrange polynomial  $L_{x,S}$  is a polynomial of degree |S| - 1 that evaluates to zero on  $S \setminus \{x\}$  and has  $L_{x,S}(x) = 1$ . For cosets of a multiplicative group  $\mathbb{V}$  in  $\mathbb{H}$ , both the vanishing polynomial  $Z_{\mathbb{V}}$  and the Lagrange polynomial  $L_{x,\mathbb{V}}$  for  $x \in \mathbb{V}$  have efficiently computable forms. The vanishing polynomial takes the form  $Z_{\omega^i \mathbb{V}}(X) = X^m - \omega^{im}$ . The Lagrange polynomial takes the form  $L_{x,\mathbb{V}}(X) = \frac{c_x Z_{\mathbb{V}}(X)}{X-x}$  where  $c_x$  is the Lagrange

constant for x defined to be  $\frac{1}{\prod_{y \in \mathbb{V}, x \neq y} x - y}$ . Note that  $c_x, \forall x \in \mathbb{V}$  can be precomputed in  $O(|\mathbb{V}|)$  time. In particular, for  $\mathbb{V} = \langle \beta \rangle$  and  $x = \beta^i$ , then  $c_x = (\beta^i/|\mathbb{V}|)$ .

<u>Multivariate polynomials</u>. We will often use the boolean hypercube as an evaluation domain where  $\{0,1\}^{\mu}$  denotes a boolean hypercube of dimension  $\mu$ . Assume expansion over the hypercube is done in a canonical order, e.g.,  $[a(i)]_{i \in \{0,1\}^{\mu}} = [a(0,\ldots,0),\ldots,a(1,\ldots,1)]$ . Define  $bin_{\mu} : [2^{\mu}] \to \{0,1\}^{\mu}$  for mapping integers to their boolean representation, and analogously  $int_{\mu} : \{0,1\}^{\mu} \to [2^{\mu}]$  for mapping boolean vectors to their integer representation. Define the equality polynomial  $\widetilde{eq}_{\mu}(X_{[\mu]}, Y_{[\mu]}) = \prod_{i \in [\mu]} X_i Y_i + (1 - X_i)(1 - Y_i)$  that on inputs  $X_{[\mu]}, Y_{[\mu]}$  over the boolean hypercube  $\{0,1\}^{\mu}$  outputs 1 if X = Y, else outputs 0. Define the shift operator  $\widetilde{shft}_b(f)$  that takes a polynomial  $f \in \mathbb{F}[X_{[\mu]}]$ , is parameterized by a subcube  $b \leq \mu$ , and outputs a polynomial shifted by the subcube. That is, for all  $i \in \{0,1\}^{\mu-b}$  and for all  $j \in \{0,1\}^{b}$ , it holds that  $\widetilde{shft}_b(f)(j,i) = f(bin_b(int_b(j) + 1 \mod 2^b), i)$ . The shifted polynomial  $\widetilde{shft}_b(f)$  admits an efficient evaluation algorithm using a sumcheck of size b, a query to f, and an evaluation of an O(b) size polynomial. We refer the reader to Diamond and Posen's full definition of  $\widetilde{shft}_b(f)$  [DP23, Section 4.3].

For every function  $f : \{0,1\}^{\mu} \to \mathbb{F}$ , there is a unique multilinear polynomial  $\tilde{f} \in \mathbb{F}[X_{[\mu]}]$  (called the multilinear extension (MLE) of f) such that for all  $i \in \{0,1\}^{\mu}$ ,  $f(i) = \tilde{f}(i)$ . The multilinear extension takes the form  $\tilde{f}(X_{[\mu]}) = \sum_{i \in \{0,1\}^{\mu}} f(i) \cdot \tilde{eq}_{\mu}(i, X_{[\mu]})$ .

## 3.2 Proof and Argument Systems

Our approach to constructing succinct non-interactive arguments of knowledge (SNARKs) will be to first build an information-theoretic proof system called a *polynomial interactive oracle proof* (polyIOP) [BFS20], a generalization of interactive oracle proofs [BCS16, RRR16], which themselves combine aspects of interactive proofs [Bab85, GMR85] and probabilistically checkable proofs [BFLS91, AS92]. We review the formalism and provide a self-contained definition following the treatment of [CBBZ23] and [BNO21].

Interactive arguments of knowledge. We begin by describing *interactive arguments of knowledge* for *indexed relations*. Such a protocol is run between three parties: an indexer, a prover, and a verifier. It consists of an initial non-interactive preprocessing phase run by an indexer to produce encoded parameters followed by an interactive online phase between a prover and verifier. An indexed relation R [CHM<sup>+</sup>20] is defined over triples (i, x, w) where *i* is called the *index*, *x* is called the *statement*, and *w* is called the *witness*. Indexed relations allow for capturing preprocessing in succinct arguments in which the verifier's input is split into two parts for offline and online phases. For example, in an indexed relation for a satisfiable boolean circuit, the index corresponds to the circuit description, the statement corresponds to assignment of public input wires, and the witness corresponds to assignment of private input wires.

An interactive argument of knowledge system  $\Pi$  is a tuple of algorithms  $\Pi = (\text{Setup, Index, P, V})$  for an indexed relation R. The algorithms are defined as follows:

- $gp \leftarrow$  Setup( $\lambda$ ): The setup algorithm takes a security parameter  $\lambda$  and outputs the global parameters gp.
- (vp, pp) ← Index(gp, i): The deterministic indexing algorithm takes as input the index i and outputs an index-specific set of verifier parameters vp and prover parameters pp. Importantly, the index algorithm does not depend on the statement or witness.
- P(pp,x,w) ↔ V(vp,x): Proving knowledge of a witness is an interactive protocol run between a prover and a verifier. We model the interactive protocol by defining a stateful algorithm for each party that takes an incoming message and a current state, and outputs an outgoing message to be passed to the next algorithm: (st<sub>P</sub>, m<sub>out</sub>) ← P.Round(st<sub>P</sub>, m<sub>in</sub>). The verifier algorithm additionally outputs a decision in {accept,cont}: (st<sub>V</sub>, m<sub>out</sub>, dec) ← V.Round(st<sub>V</sub>, m<sub>in</sub>). If the verifier accepts, the output message is parsed as m<sub>out</sub> ∈ {0,1} indicating whether verification succeeded. The state for the prover algorithm is initialized with the prover parameters, statement, and witness, (st<sub>P</sub>, m<sub>out</sub>) ← P.Init(pp, x, w), and the verifier algorithm is initialized with the verifier parameters and statement, st<sub>V</sub> ← V.Init(vp, x). The prover initialization algorithm produces the first message.

The formalism can also be applied to relations that are not indexed, i.e., consist of statement-witness pairs. In this case,

| $\underline{\langle P(pp, x, w, aux_{P}) \leftrightarrow V(vp, x, aux_{V}) \rangle_n}$   | $\frac{\text{Oracle POLY}(g, [i_j]_j^m, \alpha)}{}$                                       |  |  |
|--|---|--|--|
| Parse oracles $[(f_i, d_i)]_{i=0}^{\ell} \leftarrow vp$ ; $[(f_i, d_i)]_{i=\ell}^{\ell+m} \leftarrow x$<br>$vp \leftarrow [d_i]_{i=0}^{\ell}$ ; $x \leftarrow [d_i]_{i=\ell}^{\ell+m}$ ; $ctr \leftarrow \ell+m$ | Require $g \in \mathbb{F}^1[X_1, \dots, X_m]$<br>Require $\forall_j^m \ i_j \in [0, ctr]$ |  |  |
| $(st_{P}, m) \leftarrow P.Init(pp, x, w, aux_{P})$   | Return $g(f_{i_0}(\alpha), \dots, f_{i_{m-1}}(\alpha))$                                   |  |  |
| $st_V \leftarrow V.Init(vp, x, aux_V)$   |   |  |  |
| Repeat <i>n</i> times:   |   |  |  |
| $(st_{V}, m, dec) \leftarrow V.Round \xrightarrow{POLY} (st_{V}, m)$   |   |  |  |
| If $dec = \texttt{accept}$ then return $m$   |   |  |  |
| $(st_{P}, m) \leftarrow P.Round(st_{P}, m)$  |   |  |  |
| Parse oracles $[(f_i, d_i)]_{i=ctr}^{ctr+\ell} \leftarrow m$   |   |  |  |
| $m \leftarrow [d_i]_{i=ctr}^{ctr+\ell} \ ; \ ctr \leftarrow ctr + \ell$  |   |  |  |
| Return 0   |   |  |  |

Figure 3: Interactive proving protocol between prover and verifier. The highlighted code is included for polynomial interactive oracle proof protocols. The oracles are parsed as polynomial and degree bound pairs where the polynomial is used to respond to oracle queries and the degree bound is passed along to the verifier.

| Game SOUND $\mathcal{A}_{\Pi,R,X,n}(\lambda)$   | Game $\operatorname{ZK}_{\Pi,R,S,n}^{\mathcal{A},b}(\lambda)$  |  |
|---|--|--|
| $\overline{gp \leftarrow \Pi.Setup(\lambda)}$   | $\boxed{gp_1 \leftarrow \ \ \Pi.Setup(\lambda); \ (gp_0, st_{S}) \leftarrow \ \ S.Setup(\lambda)}$   |  |
| $(i, x, st_{\mathcal{A}}) \leftarrow \mathcal{A}_1(gp)$   | $(i, x, w, st_{\mathcal{A}}) \leftarrow \mathcal{A}_1(gp_b)$   |  |
| $(vp, pp) \leftarrow \Pi.Index(gp, i)$  | $(vp, pp) \leftarrow \Pi.Index(gp_b, i)$   |  |
| $w \leftarrow X^{\mathcal{A}_1, \mathcal{A}_2}(gp, i, x)$   | $vw_1 \leftarrow View\langle \Pi.P(pp, x, w, \bot) \leftrightarrow \Pi.V(vp, x, \bot) \rangle_n$   |  |
| $ \left( \left\langle \mathcal{A}_2(pp, \mathbf{x}, \bot, st_{\mathcal{A}}) \leftrightarrow \Pi.V(vp, \mathbf{x}) \right\rangle_n \right) $ | $vw_0 \leftarrow S.SimView(i, x, st_S)$  |  |
| Return $\bigwedge \left( (i, x, w) \notin R \right)$  | $\left  \begin{array}{c} \text{Return } \Lambda \left( \begin{array}{c} \mathcal{A}_2(vw_b, st_{\mathcal{A}}) \end{array} \right) \right. \right.$ |  |
| ( · · · · · / /   | $(i, x, w) \in R$  |  |

Figure 4: Knowledge soundness (left) and honest-verifier zero knowledge (right) security games for interactive argument systems.

the setup algorithm outputs the prover and verifier parameters directly.

We also define the following properties for an interactive argument of knowledge:

<u>Completeness</u>. An argument system is *complete* if given a tuple  $(i, x, w) \in \mathsf{R}$ , a prover can convince a verifier. A proof system  $\Pi$  with n rounds of interaction has *perfect completeness* for  $\mathsf{R}$  if  $\forall (i, x, w) \in \mathsf{R}$  and choice of security parameter  $\lambda$ ,

$$\Pr\left[ \begin{array}{c|c} \left\langle \mathsf{P}(pp, x, w) \leftrightarrow \mathsf{V}(vp, x) \right\rangle_n = 1 & \left| \begin{array}{c} gp \leftarrow \mathsf{s} \, \mathsf{Setup}(\lambda) \\ (vp, pp) \leftarrow \mathsf{Index}(gp, i) \end{array} \right] = 1. \end{array} \right]$$

<u>Knowledge soundness.</u> An argument system is knowledge-sound or is an argument of knowledge if whenever a prover is able to produce a valid proof for an index and statement (i, x), it must be that the prover "knows" some witness w such that  $(i, x, w) \in \mathbb{R}$ . This is modeled via an extractor algorithm X that can learn the witness given oracle access to the prover. Here, oracle access X<sup>P</sup> means the extractor has black-box access to each "next-message" round algorithm that defines P by passing in arbitrary state (in particular, the extractor can rewind the prover by passing in previous state). If the adversary running time is unbounded, it is known as a proof of knowledge. We define security via the pseudocode game SOUND in Figure 4. An *n*-round protocol  $\Pi$  is considered knowledge-sound if there exists an extractor X such that for all  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  the following advantage probability is negligible in  $\lambda$ : Adv<sup>sound</sup><sub> $\Pi, R, X, n, \mathcal{A}$ </sub> ( $\lambda$ ) = Pr [SOUND<sup> $\mathcal{A}_{\Pi, R, X, n}$ </sup>( $\lambda$ ) = 1].

Zero knowledge. An argument system is honest-verifier zero-knowledge (HVZK) if the interactive protocol does not leak any information to the verifier besides membership in the relation. We define security via the pseudocode game ZK in Figure 4 in which an adversary is tasked with distinguishing between an honest-verifier interaction with a prover with knowledge of a valid witness and a simulated interaction without a witness. In the pseudocode, View denotes the view of the verifier consisting of the transcript of prover and verifier messages.

An *n*-round protocol  $\Pi$  is zero-knowledge if there exists a simulator S such that for all  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , the following advantage probability is negligible in  $\lambda$ :

$$\mathsf{Adv}_{\mathsf{\Pi},\mathsf{R},\mathsf{S},n,\mathcal{A}}^{\mathrm{zk}}(\lambda) = \left| \Pr \Big[ \mathrm{ZK}_{\mathsf{\Pi},\mathsf{R},\mathsf{S},n}^{\mathcal{A},1}(\lambda) = 1 \Big] - \Pr \Big[ \mathrm{ZK}_{\mathsf{\Pi},\mathsf{R},\mathsf{S},n}^{\mathcal{A},0}(\lambda) = 1 \Big] \right|.$$

<u>Public coin</u>. An interactive argument is considered *public coin* if all of the verifier messages are uniform random samples from some predefined challenge space. For public-coin, interactive oracle protocols (defined below), View used in zero-knowledge formalism consists of  $(r, a_1, \ldots, a_q)$  where r is the verifier randomness and  $a_1, \ldots, a_q$  are responses to verifier's q oracle queries. Importantly, the verifier messages should not depend on the results of oracles it may have access to.

**Polynomial interactive oracle proofs (PolyIOPs).** We next introduce polynomial interactive oracle proofs (polyIOPs) as a special case of an interactive proof of knowledge. A polyIOP allows polynomial oracles to be made available to the verifier as part of the verifier parameters, the statement, and the prover messages. Oracles specify a degree bound of the polynomial and can be queried by the verifier at arbitrary points. In our protocol descriptions, we will denote an oracle for polynomial  $f \in \mathbb{F}^{\leq d}[X_{[\mu]}]$  as  $[\![f]\!]^{\leq d}$  (dropping the superscript if clear from context).

The security of polyIOPs are defined the same as for interactive arguments; we provide pseudocode for the handling of polynomial oracles in Figure 3. We additionally require the following properties for polynomial oracles:

<u>Oracle degree admissibility</u>. Every oracle provided to the verifier is accompanied by a degree bound for the corresponding polynomial. A polyIOP is *degree admissible* if the degree bounds of the polynomials the indexer and prover provide as part of the oracle description correctly bound the polynomials used to instantiate the oracle.

<u>Virtual oracles for linear combinations</u>. In our formalization of polyIOPs, we will also allow for the verifier to make queries to virtual oracles which are queries to polynomials that are linear combinations of oracles the verifier has received [BCG<sup>+</sup>19,GWC19]. More formally, if a verifier has oracles for polynomials  $f_1, \ldots, f_m \in \mathbb{F}[X]$ , then they may make queries to virtual oracle  $g(f_1, \ldots, f_m) \in \mathbb{F}[X]$  where  $g \in \mathbb{F}^1[X_1, \ldots, X_m]$  (see Figure 3).

<u>Domain-restricted admissibility</u>. We define a further restricted form of polyIOPs where every polynomial oracle is accompanied by an *evaluation domain*. This idea generalizes the treatment of [CBBZ23] which defines a restriction of multivariate polyIOPs to sum-checks over the boolean hypercube. A polyIOP is *domain admissible* if the verifier never requests evaluation queries for oracles (or virtual oracles derived from an oracle) at any point within the union of all restricted domains. Convenient properties for polyIOPs emerge when the witness of the prover is encoded as evaluations of the polynomial on the evaluation domain (such encodings are common in existing polyIOPs [GWC19, CHM<sup>+</sup>20]).

**PolyIOP compilation.** PolyIOPs are a useful information-theoretic proof system for abstracting and proving the security of protocols. There exist a number of standard techniques for compiling sound polyIOPs into protocols with additional properties [CHM<sup>+</sup>20, BFS20, CBBZ23].

Zero-knowledge compiler: Using our introduced notion of domain admissibility, we propose a new zero-knowledge compiler for univariate polyIOPs. A sound domain-admissible polyIOP whose witness contains only specified evaluations of oracle polynomials on the restricted evaluation domain and whose verifier only performs zero tests over evaluation subdomains compiles to a zero-knowledge sound polyIOP by careful application of a bounded independence argument to the polynomial oracles [BCR<sup>+</sup>19, CHM<sup>+</sup>20]. The full compiler details are provided in Appendix B. A similar zero-knowledge compiler exists for multivariate polyIOPs that we will also make use of. Here, the verifier is restricted to perform sum checks over evaluation subdomains (boolean hypercubes) [CBBZ23].

<u>Oracle instantiation compiler</u>. Polynomial oracles are instantiated with polynomial commitments provided by the prover. If the polynomial commitment scheme is additively homomorphic, the compilation can support virtual oracles for linear combinations. An oracle-admissable, knowledge-sound polyIOP compiles to an interactive argument of knowledge if the polynomial commitment scheme has witness-extended emulation. A zero-knowledge polyIOP compiles to a zero-knowledge interactive argument if the polynomial commitment is hiding and provides zero-knowledge evaluation [CHM<sup>+</sup>20, Theorem 8.1-8.4].

<u>Non-interaction compiler</u>. Further, if the polyIOP is public-coin and the above hold, it can be compiled to a zeroknowledge non-interactive argument of knowledge (zkNARK) using the Fiat-Shamir transform in the random oracle model. As evidenced by recent work, care should be taken when applying the transform to avoid so-called "weak Fiat-Shamir" attacks [DMWG23]. Tighter knowledge soundness bounds have been shown for applying Fiat-Shamir to related multi-round protocols [AFK22] and for providing the stronger adaptive soundness notion of simulation extractability [FKMV12, GKK<sup>+</sup>22, DG23]. We leave further analysis of the non-interactive variant of our protocol to future work.

#### 3.3 Useful PolyIOPs

Here we enumerate some useful polyIOPs—zero test, sum check, product check, and multiset check—that we will make use of in building higher level protocols. We note that all of these protocols are domain admissible and may be compiled to zero-knowledge. We defer the full presentation of the protocols to Appendix A.

# 4 Succinct Vector Lookup

We propose two constructions for succinct vector lookup. One in which the vectors are encoded within univariate polynomials and the second in which the vectors are encoded within multivariate polynomials. Both constructions follow the same high level blueprint and their security is derived from the following main technical lemma from Haböck [Hab22]:

Lemma 2. Let  $\mathbb{F}$  be a field with  $\operatorname{char}(\mathbb{F}) > \max(d_0, d_1)$ . Suppose  $\{\{f_{i,j}\}_{j \in [m]}\}_{i \in [d_0]}$  and  $\{\{t_{i,j}\}_{j \in [m]}\}_{i \in [d_1]}$  are sequences of element vectors in  $\mathbb{F}$ . Then,  $\{\{f_{i,j}\}_{j \in [m]}\}_{i \in [d_0]} \subseteq \{\{t_{i,j}\}_{j \in [m]}\}_{i \in [d_1]}$  if and only if there exists a sequence of field elements  $[c_i]_{i \in [d_1]}$  such that

$$\sum_{i \in [d_0]} 1/(X - \sum_{j \in [m]} f_{i,j}Y^j) = \sum_{i \in [d_1]} c_i/(X - \sum_{j \in [m]} t_{i,j}Y^j).$$

where equality holds over the rational function field  $\mathbb{F}(X, Y)$ .

In this section, we present the univariate polynomial vector encoding construction, CosetLkup, in which vectors are encoded on coset evaluation domains. The multivariate construction, SubcubeLkup, encodes vectors on boolean subcube evaluation domains; it is deferred to Appendix C.

Recall the univariate polynomial encoding for vectors from Section 2.2. Given a table of vectors  $[[t_{i,j}]_{j\in[m]}]_{i\in[d_1]}$  and a list of claimed looked up vectors  $[[f_{i,j}]_{j\in[m]}]_{i\in[d_0]}$ , we consider polynomial encodings  $t \in \mathbb{F}^{md_1}[X]$  and  $f \in \mathbb{F}^{md_0}[X]$  as follows. Consider two subgroups  $\mathbb{H}_0 = \langle \omega_0 \rangle \leq \mathbb{F}$  and  $\mathbb{H}_1 = \langle \omega_1 \rangle \leq \mathbb{F}$  such that  $|\mathbb{H}_0| = md_0$  and  $|\mathbb{H}_1| = md_1$ . Further consider the shared subgroup  $\mathbb{V} \leq \mathbb{H}_0$  (and  $\mathbb{V} \leq \mathbb{H}_1$ ) such that  $|\mathbb{V}| = m$  and  $\mathbb{V} = \langle \gamma = \omega_0^{d_0} = \omega_1^{d_1} \rangle$ . The vectors are encoded as the evaluations of each coset of  $\mathbb{V}$  in  $\mathbb{H}_0$  and  $\mathbb{H}_1$  respectively:

$$\left[f(\omega_0^i \mathbb{V}) = [f_{i,j}]_{j \in [m]}\right]_{i \in [d_0]}, \qquad \left[t(\omega_1^i \mathbb{V}) = [t_{i,j}]_{j \in [m]}\right]_{i \in [d_1]}$$

Given this encoding, we consider the following vector lookup relation for univariate polynomials:

$$\mathsf{R}_{\mathsf{vlkup}} = \left\{ \bot, (\llbracket f \rrbracket, \llbracket t \rrbracket), (f, t) : \left\{ f(\omega_0^i \mathbb{V}) \right\}_{i \in [d_0]} \subseteq \left\{ t(\omega_1^i \mathbb{V}) \right\}_{i \in [d_1]} \right\}$$

We walk through the main points of our construction in Section 2.2. The full construction is provided in Figure 5.

**Security.** We prove the completeness, knowledge soundness, and zero knowledge of CosetLkup in the following two theorems. The zero-knowledge of CosetLkup is achieved through our zero-knowledge compiler (see Appendix B) observing that CosetLkup is domain-restriction admissible and reduces to zero test polynomial identities.

Theorem 3. CosetLkup for R<sub>vlkup</sub> (Figure 5) is complete and knowledge sound with negligible error.

*Proof.* CosetLkup for  $\mathsf{R}_{\mathsf{vlkup}}$  (Figure 5) has completeness error  $\frac{|\mathbb{H}_0| + |\mathbb{H}_1|}{|\mathbb{F}| - |\mathbb{H}_0| - |\mathbb{H}_1|}$  and for any adversary  $\mathcal{A}$  against knowledge

| Rullium = 4 | $ \left( \perp, \left( \mathbb{H}_0 = \langle \omega_0 \rangle, \mathbb{H}_1 = \langle \omega_1 \rangle, \mathbb{V} = \langle \gamma = \omega_0^{d_0} = \omega_1^{d_1} \rangle, \llbracket f \rrbracket, \llbracket t \rrbracket \right), (f, t) \right) \  \  \right) $ |
|-------------|--|
|             | $\left( : \left\{ f(\omega_0^i \mathbb{V}) \right\}_{i \in [d_0]} \subseteq \left\{ t(\omega_1^i \mathbb{V}) \right\}_{i \in [d_1]} \right)$   |

 $\mathsf{CosetLkup}.\mathsf{P}(\bot,(\mathbb{H}_0,\mathbb{H}_1,\mathbb{V},\llbracket f \rrbracket,\llbracket t \rrbracket),(f,t)) \leftrightarrow \mathsf{CosetLkup}.\mathsf{V}(\bot,(\mathbb{H}_0,\mathbb{H}_1,\mathbb{V},\llbracket f \rrbracket,\llbracket t \rrbracket))$ 

(1) P computes, sends, and proves the wellformedness of the count polynomial c that encodes the counts  $[c_i]_{i \in [d_1]}$  where  $c_i$  is the number of times the vector  $t(\omega_1^i \mathbb{V})$  appears in f.

(a) P sends c defined over  $\mathbb{H}_1$  setting the evaluation to be constant in each coset:  $\{\{c(\omega_1^i \gamma^j) = c_i\}_{i \in [d_1]}\}_{j \in [m]}$ .

(b) P and V engage in  $\text{ZeroTest}(\mathbb{H}_1,\mathbb{H}_1)$  to prove every coset of  $\mathbb{V}$  in  $\mathbb{H}_1$  encodes a constant value:  $c(\gamma X) = c(X)$  over  $\mathbb{H}_1$ .

(2) V sends random challenges  $(\alpha, \beta) \in (\mathbb{F} \setminus (\mathbb{H}_0 \cup \mathbb{H}_1))^2$ .

(3) P computes and sends the position-indexing powers-of- $\beta$  polynomial  $I_b(X)$  for  $b \in \{0,1\}$  and proves its well-formedness:

(a) P computes and sends  $I_b$  defined over  $\mathbb{H}_b$  setting the evaluation of the  $j^{th}$  element of each coset to be j-th power-of- $\beta$ ,  $\beta^j$ : { $\{I_b(\omega_b^i \gamma^j) = \beta^j\}_{i \in [d_b]}\}_{j \in [m]}$ 

(b) P and V engage in ZeroTest( $\mathbb{V}, \mathbb{H}_b$ ) to prove  $L_{1,\mathbb{V}}(X)(I_b(X) - 1) = 0$  over  $\mathbb{V}$ .

(c) P and V engage in  $\operatorname{ZeroTest}(\mathbb{V}, \mathbb{H}_b)$  to prove  $(I_b(\gamma X) - \beta \cdot I_b(X))(X - \gamma^{m-1}) = 0$  over  $\mathbb{V}$ .

(d) P and V engage in ZeroTest( $\mathbb{H}_b, \mathbb{H}_b$ ) to prove  $(I_b(X) - I_b(\omega_b X))Z_{\omega,d_b^{-1}\mathbb{V}}(X) = 0$  over  $\mathbb{H}_b$ .

(4) P computes and sends the summation polynomial  $S_b(X)$  for  $b \in \{0,1\}$  and proves its well-formedness:

(a) P computes and sends  $S_b$  defined over  $\mathbb{H}_b$  setting the evaluation to be constant in each coset  $\omega_b^i \mathbb{V}$  for  $i \in [d_b]$  to summation of the values in the coset multiplied by powers of  $\beta$ . Let  $p_0 = f$  and  $p_1 = t$ :

$$\{\{S_b(\omega_b^i \gamma^j) = \sum_{k \in [m]} \beta^k \cdot p_b(\omega_b^i \gamma^k)\}_{i \in [d_b]}\}_{j \in [m]}$$

(b) P and V engage in ZeroTest( $\mathbb{H}_b, \mathbb{H}_b$ ) to prove every coset of  $\mathbb{V}$  in  $\mathbb{H}_b$  encodes a constant:  $S_b(\gamma X) = S_b(X)$  over  $\mathbb{H}_b$ .

(5) P sends the induction polynomial  $B_b(X)$  for  $b \in \{0,1\}$  and proves well-formedness. Let  $p_0 = f$  and  $p_1 = t$ :

(a) P computes and sends  $B_b$  defined over  $\mathbb{H}_b$  that accumulates the normalized summation:

$$\{\{B_b(\omega_b^i\gamma^j) = \sum_{k \in [j]} \left(\beta^k \cdot p_b(\omega_b^i\gamma^k) - (S_b(\omega_b^i\gamma^j))/m\right)\}_{i \in [d_b]}\}_{j \in [m]}.$$

(b) P and V engage in ZeroTest( $\mathbb{H}_b, \mathbb{H}_b$ ) to prove induction  $B_b(\gamma X) = (B_b(X) + I_b(\gamma X) \cdot p_b(\gamma X)) - S_b(X)/m$  over  $\mathbb{H}_b$ .

(6) P computes and sends the inverse polynomial  $U_b(X)$  for  $b \in \{0,1\}$  and proves its well-formedness:

(a) P computes and sends  $U_b$  defined over  $\mathbb{H}_b$  setting the evaluation of each coset  $\omega_b^i \mathbb{V}$  for  $i \in [d_b]$  to the inverse of the summation and random challenge  $\alpha$  as appears in the denominator of the Haböck lemma. Let  $p_0 = f$  and  $p_1 = t$ :

$$\{\{U_b(\omega_b^i \gamma^j) = 1/(\alpha - \sum_{k \in [m]} \beta^k p_b(\omega_b^i \gamma^k))\}_{i \in [d_b]}\}_{j \in [m]}$$

(b) P and V engage in ZeroTest( $\mathbb{H}_b$ ,  $\mathbb{H}_b$ ) to prove inversion:  $U_b(X) \cdot (\alpha - S_b(X)) = 1$  over  $\mathbb{H}_b$ .

(7) P proves summations of  $U_0$  and  $c \cdot U_1$  over  $\mathbb{H}_0$  and  $\mathbb{H}_1$ , respectively, are equal. Let  $u_0 = U_0$  and  $u_1 = c \cdot U_1$ . For  $b \in \{0, 1\}$ :

(a) P interpolates and sends polynomials  $T_b$  over  $\mathbb{H}_b$  such that:

 $\{T_b(\omega_b^i) = \sum_{k=i}^{|\mathbb{H}_b|-1} u_0(\omega_b^k)\}_{i \in |\mathbb{H}_b|}.$ 

(b) P and V engage in ZeroTest( $\mathbb{H}_b, \mathbb{H}_b$ ) to prove  $(X - \omega_b^{|\mathbb{H}_b|-1})(T_b(\omega_b X) + u_b(X) - T_b(X)) = 0$  over  $\mathbb{H}_b$ . (c) P and V engage in ZeroTest( $\mathbb{H}_b, \mathbb{H}_b$ ) to prove  $L_{\omega_h^{|\mathbb{H}_b|-1}, \mathbb{H}_b}(X)(T_b(X) - u_b(X)) = 0$  over  $\mathbb{H}_b$ .

(d) P and V engage in ZeroTest( $\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1$ ) to prove  $L_{1,\mathbb{H}_0}(X)(T_0(X) - T_1(X)) = 0$  over  $\mathbb{H}_0$ .

Figure 5: Vector lookup argument in which vectors are encoded as evaluations over coset domains in a univariate polynomial.

soundness, we provide an extractor X using X<sub>zt</sub>, an extractor for ZeroTest, such that

$$\begin{split} & \mathsf{Adv}^{sound}_{\mathsf{CosetLkup,vlkup,X,\mathcal{A}}}(\lambda) \leq 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_0,|\mathbb{H}_0|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero},X,\mathcal{A}}}(\lambda) \\ & + 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_1,|\mathbb{H}_1|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero},X_{\mathsf{zt}},\mathcal{A}}}(\lambda) + 6\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_0,\mathbb{H}_0,2|\mathbb{H}_0|),\mathsf{R}_{\mathsf{zero},X_{\mathsf{zt}},\mathcal{A}}}(\lambda) \\ & + \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_0,\mathbb{H}_0\cup\mathbb{H}_1,|\mathbb{H}_0|+\max(|\mathbb{H}_0|+|\mathbb{H}_1|)),\mathsf{R}_{\mathsf{zero},X_{\mathsf{zt}},\mathcal{A}}}(\lambda) \\ & + 7\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_1,\mathbb{H}_1,3|\mathbb{H}_1|),\mathsf{R}_{\mathsf{zero},X_{\mathsf{zt}},\mathcal{A}}}(\lambda) + \frac{|\mathbb{H}_0|+|\mathbb{H}_1|}{|\mathbb{F}|-|\mathbb{H}_0|-|\mathbb{H}_1|}. \end{split}$$

<u>Completeness</u>. For  $b \in \{0,1\}$ , The honest prover first interpolates  $c, I_b, S_b, B_b$  as indicated in steps 1(a), 3(a), 4(a), 5(a), 6(a). Then in steps 1(b), 3(b)-(d), 4(b), 5(b), 6(b) where the structures of these polynomials are tested, the verifier will succeed based on the completeness of zero tests.

As mentioned in Remark 4, the fraction might be undefined and so are the evaluations of  $U_b$  if  $\alpha, \beta$  are such that  $(\alpha - \sum_{j \in [m]} f_{i,j}\beta^j) = 0$  or  $(\alpha - \sum_{j \in [m]} t_{i,j}\beta^j) = 0$ . By the Schwartz-Zippel lemma, either happens with probability at most  $\frac{|\mathbb{H}_0|+|\mathbb{H}_1|}{|\mathbb{F}|}$ . To achieve perfect completeness, we can apply the technique in Remark 4.

Assuming  $\{(f(\omega_0^i \mathbb{V}))\}_{i \in [d_0]} \subseteq \{(t(\omega_1^i \mathbb{V}))\}_{i \in [d_1]}$ , we can apply Lemma 2, by treating index *i* as the coset index and index *j* as the element index in coset *i*. Then there exists a sequence of field elements (the multiplicity of each vector)  $[c_i]_{i \in [d_1]}$  such that when evaluating at  $\alpha, \beta$ 

$$\sum_{i \in [d_0]} \frac{1}{\alpha - \sum_{j \in [m]} f(\omega_0^i \gamma^j) \beta^j} = \sum_{i \in [d_1]} \frac{c_i}{\alpha - \sum_{j \in [m]} t(\omega_1^i \gamma^j) \beta^j}$$

If  $U_b$ 's are properly defined, then by the construction of polynomials  $c, u_0, u_1$  (constant within cosets), this is equivalent to

Į.

$$\sum_{i=0}^{\mathbb{H}_0|-1} u_0(\omega_0^i) = \sum_{i=0}^{|\mathbb{H}_1|-1} u_1(\omega_1^i)$$
(1)

The honest prover then interpolates  $T_b$  that encodes the accumulated sum of  $u_b$  across subgroup  $\mathbb{H}_b$  in reserve direction. Step 7(b) checks that the sum is encoded in decreasing order, i.e.,  $T_b(\omega_b^0) = \sum_{k=0}^{|\mathbb{H}_b|-1} u_b(\omega_b^k)$ . Since  $\mathbb{H}_0$  and  $\mathbb{H}_1$  overlaps at  $\omega_0^0 = \omega_1^0$  and Equation 1 holds, the verifier will succeed based on the completeness of zero tests. The completeness error is hence at most the probability that  $U_b$ 's are undefined.

<u>Knowledge Soundness.</u> We bound the advantage of adversary  $\mathcal{A}$  by bounding the advantage of each of a series of game hops [BR06]. We define  $G_0 = \text{SOUND}_{\text{CosetLkup}, R_{\text{vlkup}}, X}^{\mathcal{A}}(\lambda)$ . The inequality above follows from the following claims that we will justify:

(1)

$$\begin{split} |\Pr[G_0=1] - \Pr[G_1=1]| &\leq \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_0,|\mathbb{H}_0|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_1,|\mathbb{H}_1|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_0,|\mathbb{H}_0|+1),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_1,|\mathbb{H}_1|+1),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{H}_0,\mathbb{H}_0,|\mathbb{H}_0|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{H}_0,\mathbb{H}_0,|\mathbb{H}_0|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \end{split}$$

(2)

$$\begin{split} |\Pr[G_1 = 1] - \Pr[G_2 = 1]| &\leq \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0, |\mathbb{H}_0| + 1), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_1, |\mathbb{H}_1| + 1), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

$$\begin{split} &+ \mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{H}_0,\mathbb{H}_0,2|\mathbb{H}_0|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{H}_1,\mathbb{H}_1,2|\mathbb{H}_1|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \end{split}$$

(3)

$$\begin{split} &|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \leq \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0, 2|\mathbb{H}_0|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_1, 2|\mathbb{H}_1|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_1, |\mathbb{H}_1|+1), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(4)

$$\begin{split} |\Pr[G_3 = 1] - \Pr[G_4 = 1]| &\leq \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0, 2|\mathbb{H}_0|), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{H}_1, \mathbb{H}_1, 3|\mathbb{H}_1|), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{H}_0, |\mathbb{H}_0| + 1), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{H}_1, \mathbb{H}_1, 2|\mathbb{H}_1| + 1), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{H}_1, \mathbb{H}_0, \mathbb{H}_0 + \max(|\mathbb{H}_0| + |\mathbb{H}_1|)), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(5)  $|\Pr[G_4 = 1] - \Pr[G_5 = 1]| \le \frac{|\mathbb{H}_0| + |\mathbb{H}_1|}{|\mathbb{F}| - |\mathbb{H}_0| - |\mathbb{H}_1|}$ (6)  $\Pr[G_5 = 1] = 0$ 

The plan for the soundness proof is as follows: Claim 1 argues that polynomial  $I_b$ 's are constructed properly. Claim 2 argues that  $S_0$  and  $S_1$  encode hashes of each coset of f and t, respectively. Claim 3 argues that  $u_b$  encode the summand of the logarithmic derivatives evaluated at  $(\alpha, \beta)$ . Claim 4 argues that the logarithmic derivative evaluations are equal, claim 5 argues that if so then  $\{(f(\omega_0^i \mathbb{V}))\}_{i \in [d_0]} \subseteq \{(t(\omega_1^i \mathbb{V}))\}_{i \in [d_1]}$ . Lastly, Claim 6 argues that the constructed extractor always succeeds for an accepting verifier.

Claim 1: For the first step, we show that polynomial  $I_b$  encodes powers of  $\beta$  over cosets, i.e.,  $I_b(\omega_b^i \gamma^j) = \beta^j, \forall i \in [d_b], j \in [m]$ .

- $L_{1,\mathbb{V}}(X)(I_b(X)-1) = 0$  over  $\mathbb{V}$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}}(\mathbb{V},\mathbb{H}_b,|\mathbb{H}_b|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda)$ : Checks base case that  $I_b(1) = 1$ .
- $(I_b(\gamma X) \beta \cdot I_b(X))(X \gamma^{m-1}) = 0$  over  $\mathbb{V}$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{V}, \mathbb{H}_b, |\mathbb{H}_b|+1), \mathbb{R}_{\operatorname{Zero}}, \mathbb{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks inductive step that for all  $j \in [m-1]$ ,  $I_b(\gamma^j) = \beta \cdot I_b(\gamma^{j-1}) = \beta^j$ .
- $(I_b(X) I_b(\omega_b X))Z_{\omega_b^{d_b-1}\mathbb{V}}(X) = 0$  over  $\mathbb{H}_b$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_b, \mathbb{H}_b, |\mathbb{H}_b| + |\mathbb{V}|), \mathbb{R}_{\operatorname{Zero}}, \mathbb{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks  $I_b(\omega_b^i\mathbb{V}) = I_b(\omega_b^j\mathbb{V}), \forall i, j$ . Since from the previous check we have that  $I_b(\omega_b^0\mathbb{V})$  encodes the powers-of- $\beta$ , this check ensures that every coset  $\omega_b^i\mathbb{V}$  with  $i \in [d_b]$  encodes the powers-of- $\beta$ .

 $G_1$  employs the zero test extractor  $X_{zt}$  to check the above tests and aborts if the extractor fails. The claimed probability bound follows from a series of hybrids bounding each hybrid by the soundness advantages for the zero tests as claimed above.

*Claim 2:* In the second claim, we argue that  $S_b$  encode hashes of each coset of  $p_0 = f$  and  $p_1 = t$ .

- $S_b(\gamma X) = S_b(X)$  over  $\mathbb{H}_b$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_b, \mathbb{H}_b, |\mathbb{H}_b|), \mathbb{R}_{\operatorname{zero}}, \mathbb{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks  $\forall i \in [d_b], S_b(\omega_b^i \mathbb{V})$  encodes some constant value.
- $B_b(\gamma X) = (B_b(X) + I_b(\gamma X) \cdot p_b(\gamma X)) \frac{S_b(X)}{m}$  over  $\mathbb{H}_b$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_b, \mathbb{H}_b, 2|\mathbb{H}_b|), \mathbb{R}_{\operatorname{Zero}}, \mathbb{X}_{\operatorname{zt}}, \mathcal{A}(\lambda)$ : Checks  $\forall i \in [d_b], \sum_{j \in [m]} I_b(X) p_b(X) = \sum_{j \in [m]} \beta^j p_b(X) = \sum_{j \in [m]} \frac{S_b(X)}{m}$ . Since  $S_b(X)$  is constant over  $\omega_b^i \mathbb{V}$ , we have that  $S_b(\omega_b^i \mathbb{V})$  must encode the hash:  $S_b(\omega_b^i \mathbb{V}) = \sum_{j \in [m]} \beta^j p_b(\omega_b^i \gamma^j)$ .

Again,  $G_2$  employs the zero test extractor  $X_{zt}$  to check the above tests and aborts if the extractor fails. The claimed

probability bound follows from a series of hybrids bounding each hybrid by the soundness advantages for the zero tests as claimed above, doubling for f and t.

*Claim 3:* In the third claim, we want to prove that  $u_b$  encodes the summand of the logarithmic derivatives evaluated at  $(\alpha, \beta)$ .

- $U_b(X) \cdot (\alpha S_b(X)) = 1$  over  $\mathbb{H}_b$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_b, \mathbb{H}_b, 2|\mathbb{H}_b|), \operatorname{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Given  $S_b$  encodes hashes of  $p_b(\omega_b^i \mathbb{V})$  and is constant over cosets, it checks  $U_b(\omega_b^i \mathbb{V}) = \frac{1}{\alpha \sum_{k \in [m]} \beta^k p_b(\omega_b^i \gamma^k)}$ . Note that it is also constant over cosets.
- $c(\gamma X) = c(X)$  over  $\mathbb{H}_1$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_1, \mathbb{H}_1, |\mathbb{H}_1|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$ : Checks  $\forall i \in [d_1], c(\omega_1^i \mathbb{V})$  encodes some constant value  $c_i$ .

Then by construction, we have

$$u_0(\omega_0^i) = \frac{1}{\alpha - \sum_{k \in [m]} \beta^k f(\omega_0^i \gamma^k)}, \omega_0^i \in \mathbb{H}_0$$
$$u_1(\omega_1^i) = \frac{c_i}{\alpha - \sum_{k \in [m]} \beta^k t(\omega_1^i \gamma^k)}, \omega_1^i \in \mathbb{H}_1$$

G<sub>3</sub> employs the zero test extractor X<sub>zt</sub> to check the above tests and aborts if the extractor fails.

*Claim 4:* In the fourth claim, we prove logarithmic derivatives are equal when evaluated at  $(\alpha, \beta)$ .

- $L_{\omega_b^{|\mathbb{H}_b|-1},\mathbb{H}_b}(X)(T_b(X)-u_b(X)) = 0$  over  $\mathbb{H}_b$  with advantage  $\mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathrm{sound}}(\mathbb{H}_0,\mathbb{H}_0,2|\mathbb{H}_0|),\mathbb{R}_{\mathsf{Zero}},\mathsf{x}_{\mathsf{zt}},\mathcal{A}}(\lambda)$  for b=0 and  $\mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathrm{sound}}(\mathbb{H}_1,\mathbb{H}_1,3|\mathbb{H}_1|),\mathbb{R}_{\mathsf{Zero}},\mathsf{x}_{\mathsf{zt}},\mathcal{A}}(\lambda)$  for b=1: Checks base case that  $T_b(\omega_b^{|\mathbb{H}_b|-1}) = u_b(\omega_b^{|\mathbb{H}_b|-1})$ .
- $(X \omega_b^{|\mathbb{H}_b|-1})(T_b(\omega_b X) + u_b(X) T_b(X)) = 0$  over  $\mathbb{H}_b$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_0, \mathbb{H}_0, \mathbb{$
- $L_{1,\mathbb{H}_0}(X)(T_0(X)-T_1(X)) = 0$  over  $\mathbb{H}_0$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_0,\mathbb{H}_0\cup\mathbb{H}_1,|\mathbb{H}_0|+\max(|\mathbb{H}_0|+|\mathbb{H}_1|)),\mathbb{R}_{\operatorname{Zero}},\mathbb{X}_{\operatorname{zt}},\mathcal{A}}(\lambda)$ : Checks that  $\sum_{k=0}^{|\mathbb{H}_0|-1} u_0(\omega_0^k) = \sum_{k=0}^{|\mathbb{H}_1|-1} u_1(\omega_1^k).$

By Claim 3, the last check is equivalent to

$$\sum_{i \in [d_0]} \frac{1}{\alpha - \sum_{k \in [m]} \beta^k f(\omega_0^i \gamma^k)} = \sum_{i \in [d_1]} \frac{c_i}{\alpha - \sum_{k \in [m]} \beta^k t(\omega_1^i \gamma^k)}$$

 $G_4$  employs the zero test extractor  $X_{zt}$  to check the above tests and aborts if the extractor fails. Claim 5: Define a bivariate polynomial of degree  $m \cdot (d_0 + d_1 - 1)$ :

$$\begin{split} g(X,Y) &= \prod_{k \in [d_0]} \left( X + \sum_{j \in [m]} f(\omega_0^k \gamma^j) Y^j \right) \cdot \prod_{\ell \in [d_1]} \left( X + \sum_{j \in [m]} t(\omega_1^\ell \gamma^j) Y^j \right) \\ & \cdot \left( \sum_{k \in [d_0]} \frac{1}{X + \sum_{j \in [m]} f(\omega_0^k \gamma^j) Y^j} - \sum_{\ell \in [d_1]} \frac{c_i}{X + \sum_{j \in [m]} t(\omega_1^\ell \gamma^j) Y^j} \right) \end{split}$$

It is a polynomial since we multiply the fractionals by the common denominator. By claim 4, we have the difference of the sum to be zero at  $(\alpha, \beta)$ , and so  $g(\alpha, \beta) = 0$ . By an application of Schwartz-Zippel lemma (or Reed-Solomon encoding), g must be the zero polynomial except with probability at most  $\frac{|\mathbb{H}_0| + |\mathbb{H}_1|}{|\mathbb{F}| - |\mathbb{H}_0| - |\mathbb{H}_1|}$ . Hence, it must be also the case that

$$\sum_{k \in [d_0]} \frac{1}{X + \sum_{j \in [m]} f(\omega_0^k \gamma^j) Y^j} = \sum_{\ell \in [d_1]} \frac{c_i}{X + \sum_{j \in [m]} t(\omega_1^\ell \gamma^j) Y^j}$$

By Lemma 2, this implies that  $\{(f(\omega_0^i \mathbb{V}))\}_{i \in [d_0]} \subseteq \{(t(\omega_1^i \mathbb{V}))\}_{i \in [d_1]}$ .

Claim 6: Finally, we construct our extractor X that always succeeds on a verifying prover. X employs  $X_{zt}$  to retrieve and output f, t. By Claim 2, in  $G_2$ , if the verifier succeeds,  $X_{zt}$  always succeeds and so our extractor will always succeed.

**Remark 4** (Perfect Completeness). CosetLkup fails to achieve perfect completeness since the chosen randomness may make the denominator  $(\alpha - \sum_{j \in [m]} f_{i,j}\beta^j) = 0$  or  $(\alpha - \sum_{j \in [m]} t_{i,j}\beta^j) = 0$ , so the fraction is undefined. Protostar [BC23] illustrates a technique to recover perfect completeness in exchange for slightly increased soundness error. When the evaluations of  $U_b$  are undefined at  $\omega_b^i \gamma^j$ , the prover sets  $U_b(\omega_b^i \gamma^j) = 0$ . At step 6(b), instead of checking  $U_b(X) \cdot (\alpha - S_b(X)) \stackrel{?}{=} 1$  over  $\mathbb{H}_b$ , verifier instead checks  $(U_b(X) \cdot (\alpha - S_b(X)) - 1)(\alpha - S_b(X)) \stackrel{?}{=} 0$  over  $\mathbb{H}_b$ . Then either  $U_b(\omega_b^i \gamma^j) = \frac{1}{\alpha - \sum_{k \in [m]} p_b(\omega_b^i \gamma^k)\beta^k}$  or  $\alpha - S_b(\omega_b^i \gamma^j) = 0$ . The latter case captures the undefined scenario. The soundness error is raised by the two additional zero tests.

**Theorem 5.** The compiled polyIOP using the compiler in Figure 19 of CosetLkup for  $R_{vlkup}$  (Figure 5) is honest-verifier zero-knowledge.

*Proof.* It is evident from Figure 5 that CosetLkup can be formulated as ZT-Based as it can be grouped into the oracle and randomness sending phase, and the ZeroTest phase, and the V never makes queries outside ZeroTest. Since it is a ZT-Based polyIOP and it is domain-admissible, we can apply the zero knowledge compiler to construct an HVZK protocol.

*k*-Vector Lookup. We can extend the lookup argument to work across *k* pairs of polynomials  $(f_i, t_i)$  for  $i \in [k]$  checking that the same vector lookup applies across all *k* pairs. Our approach follows the multitable approach in [GW20] simply using a random linear combination to construct an expanded hash combining evaluations from all *k* polynomials. To be precise, prover sends *f* as the random linear combination of  $(f_i)_{i \in [k]}$  and *t* as the linear combination of  $(t_i)_{i \in [k]}$  using verifier randomness. Then the prover and verifier engage in CosetLkup using *f* and *t*. The relation is captured as:  $R_{k-vlkup} =$ 

$$\left\{(\bot, [\llbracket f_i \rrbracket, \llbracket t_i \rrbracket]_{i \in [k]}, (f_i, t_i)_{i \in [k]}) : \{(f_i(\omega_0^j \mathbb{V}))_{i \in [k]}\}_{j \in [d_0]} \subseteq \{(t_i(\omega_1^j \mathbb{V}))_{i \in [k]}\}_{j \in [d_1]})\right\}.$$

**Corollary 6.** *k*-CosetLkup for  $R_{k-vlkup}$  is complete and knowledge sound with negligible error, and the compiled polyIOP using the compiler in Figure 19 is honest-verifier zero-knowledge.

#### 5 Succinct Arguments for Unrolled Machine Execution from Vector Lookups

We model a machine execution of a machine with  $\ell$  instructions using  $\ell$  indices  $[i_i]_{i=0}^{\ell-1}$  to an indexed relation R (e.g., rank-1 constraint satisfiability or circuit satisfiability). The index for an instruction takes in a statement x of the form:

$$\mathbf{x} = (\text{inst}_{in}, \text{mem}_{in}, \text{inst}_{out}, \text{mem}_{out}),$$

which can be parsed as two parts. The first part  $(inst_{in}, mem_{in})$  is the "input" to the instruction where  $inst_{in} \in \mathbb{Z}_{\ell}$ specifies which instruction to run and  $mem_{in}$  captures the current memory (or state) of the machine. The second part  $(inst_{out}, mem_{out})$  is the "output" of the instruction specifying the next instruction to run  $(inst_{out})$  and the resulting memory from executing the instruction  $(mem_{out})$ . We require that the indexed relation R enforces  $inst_{in}$  to match the instruction index, i.e., that

$$\forall i \in \mathbb{Z}_{\ell} \ (i_i, (inst_{in}, mem_{in}, inst_{out}, mem_{out}), w) \in \mathsf{R} \Rightarrow inst_{in} = i.$$

In this way, our formal modeling of machine execution ties together the control logic of determining the next instruction to run and the instruction logic of applying changes to memory. In the indexed relations that we consider (rank-1 constraint systems and circuit satisfiability), the index can easily be adjusted to enforce the above by including an equality check against a constant.

Given a set of instruction indices  $[i_i]_{i=0}^{\ell-1}$  that satisfy the above, we define relation  $\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}]$  for *n* steps of unrolled machine computation:

$$\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}] = \left\{ \left( \begin{array}{c} [i_i]_{i=0}^{\ell-1}, \\ (inst_0, mem_0, inst_n, mem_n), \\ \left( [inst_j, mem_j, w_j]_{j=0}^n \right) \end{array} \right) : \bigwedge_{j=0}^{n-1} \left( i_{inst_j}, (inst_j, mem_j, inst_{j+1}, mem_{j+1}), w_j \right) \in \mathsf{R} \right\}$$

In the following sections we build unrolled machine execution proof systems for instructions encoded as rank-1 constraint systems ( $R_{MExe,n}[R_{r1cs}]$ ) derived from the Marlin proof system [CHM<sup>+</sup>20].

**Capturing zero-knowledge of program execution.** Even with a zero-knowledge proof system for the above relation, membership in the relation can leak information about the number of execution steps, the starting and ending instructions, and possibly the program description if it is included in the memory state. An upper bound on the number of execution steps is a fundamental leakage of the unrolled execution proving approach. To mitigate leakage of starting and end instructions, we propose including special instructions for program start and successful return. Lastly, to mitigate leakage of program description, the memory state can be considered in two parts, one that includes the input and output registers that can be revealed to the verifier and another as a hiding commitment to the program description.

#### 5.1 Mux-PLONK: Adapting the PLONK PolyIOP to Machine Execution

PLONK [GWC19] is a polyIOP for NP statements encoded using the following PLONK arithmetization.

**Definition 7.** A PLONK relation is indexed by the tuple  $(\mathbb{F}, sel, \sigma, G, \ell_s, \ell_z, d, d_x)$  where  $sel = [[sel_{i,j}]_{i \in [d]}]_{j \in [\ell_s]}$  is the selector vector, the copy vector  $\sigma : [d\ell_z] \to [d\ell_z]$  is a permutation over  $d\ell_z$ , and  $G : \mathbb{F}[X_{[\ell_s, \ell_z]}]$  is the gate polynomial. The statement  $x \in \mathbb{F}^{d_x}$  and witness  $w \in \mathbb{F}^{d\ell_z - d_x}$  together form an input vector  $z = [[z_{i,j}]_{i \in [d]}]_{j \in [\ell_z]}$  where the following algebraic relation encoding the gate constraints and copy constraints is satisfied:

$$\mathsf{R}_{\mathsf{plonk}} = \begin{cases} \left( \begin{array}{c} (\mathbb{F}, sel, \sigma, G, \ell_s, \ell_z, d, d_x), \\ x, \\ w \end{array} \right) & z = x \parallel w \in \mathbb{F}^{d \times \ell_z} \\ & \bigwedge_{i \in [d]} G([sel_{i,j}]_{j \in [\ell_s]}, [z_{i,j}]_{j \in [\ell_z]}) = 0 \\ & \bigwedge_{i \in [d\ell_z]} z_{\lfloor i/d \rfloor, i \operatorname{mod} d} = z_{\lfloor \sigma(i)/d \rfloor, \sigma(i) \operatorname{mod} d} \end{cases} \end{cases}$$

Here, we take as part of indexing two groups  $\mathbb{H}$  and  $\mathbb{H}_x$  of size d and  $d_x$ , respectively. The statement is encoded as evaluations over domain  $\mathbb{H}_x$  for a polynomial x given to the verifier. For completeness, we give the PLONK polyIOP in Figure 6 along with a modified PLONK relation to fit the univariate polyIOP setting.

There are two sets of polynomials output as part of the computation commitment during indexing: (1) selector polynomials that encode the gate constraints, and (2) permutation polynomials that encode the copy constraints (i.e., which input value should be set equal to each other). As discussed in the technical overview (Section 2.3), in the application to machine execution, the values encoded in each of these polynomials for the computation commitment of a single instruction is instead encoded in a polynomial representing the entire instruction set, i.e., a machine commitment. Each instruction is encoded within a different coset of the machine commitment polynomial's evaluation domain.

Define the following evaluation domains:

- Define  $\mathbb{V} = \langle \gamma \rangle$  as the multiplicative subgroup of size d.
- Define G = ⟨μ⟩ as the multiplicative subgroup of size dn where n is the number of unrolled execution steps. Denote the n cosets of V in G as [μ<sup>i</sup>V]<sup>n-1</sup><sub>i=0</sub>.
- Define H = ⟨ω⟩ as the multiplicative subgroup of size dℓ where ℓ is the number of instructions. Denote the ℓ cosets of V in H as [ω<sup>i</sup>V]<sup>ℓ-1</sup><sub>i=0</sub>.
- Define  $\mathbb{V}_x \leq \mathbb{V}$  as the multiplicative subgroup of size  $d_x$  where  $d/d_x = a$ .

$$\mathsf{R}_{\mathsf{plonk}} = \left\{ \begin{pmatrix} (\mathbb{F}, \mathbb{H}, \mathbb{H}_x, sel, \sigma, G, \ell_s, \ell_z, d, d_x), \\ ([\mathbb{I}x]], \mathbb{K}), \\ (w, x) \end{pmatrix} \xrightarrow{x(\mathbb{H}_x) = x} z = x \parallel w \in \mathbb{F}^{d \times \ell_z} z = x \parallel w \in \mathbb{F}^{d \times \ell_z} \vdots \sum_{\substack{i \in [d] \\ i \in [d]}} G([sel_{i,j}]_{j \in [\ell_s]}, [z_{i,j}]_{j \in [\ell_z]}) = 0 \\ \bigwedge_{i \in [d\ell_z]} z_{\lfloor i/d \rfloor, i \mod d} = z_{\lfloor \sigma(i)/d \rfloor, \sigma(i) \mod d} \right\}$$

 $\mathsf{PLONK}.\mathsf{Setup}(\lambda):\mathsf{Return}\perp$ 

 $\mathsf{PLONK}.\mathsf{Index}(\bot,(\mathbb{F},\mathbb{H},\mathbb{H}_x,sel,\sigma,G,\ell_s,\ell_z,d,d_x))$ 

- (1) For each  $j \in [\ell_s]$ , encode selector polynomial  $sel_j$  setting the following evaluations over  $\mathbb{H} = \langle \omega \rangle$ :  $[sel_j(\omega^i) = sel_{i,j}]_{i \in [d]}$
- (2) For each  $j \in [\ell_z]$ , encode permutation polynomial  $\sigma_j$  setting the following evaluation over  $\mathbb{H}$ :  $\left[\sigma_j(\omega^i) = \sigma(j \cdot \ell_z + i)\right]_{i \in [d]}$ .

 $(3) \text{ Return } \left( pp \leftarrow \left( [sel_j]_{j \in [\ell_s]}, [\sigma_j]_{j \in [\ell_z]}, G \right), vp \leftarrow \left( [[sel_j]]_{j \in [\ell_s]}, [[\sigma_j]]_{j \in [\ell_z]}, G \right) \right).$ 

 $\mathsf{PLONK}.\mathsf{P}(\big([sel_j]_{j \in [\ell_s]}, [\sigma_j]_{j \in [\ell_z]}, G\big), ([\![x]\!], \mathbb{K}), (w, x)) \leftrightarrow \mathsf{PLONK}.\mathsf{V}(\big([[\![sel_j]\!]]_{j \in [\ell_s]}, [[\![\sigma_j]\!]]_{j \in [\ell_z]}, G\big), ([\![x]\!], \mathbb{K})))$ 

- P computes and sends polynomials [z<sub>j</sub>]<sub>j∈[ℓ<sub>z</sub>]</sub> each encoding d elements from the statement x and witness w as evaluations over H. The statement x is encoded in H<sub>x</sub> of z<sub>0</sub>.
- (2) To prove the gate constraints, P and V engage in  $\mathsf{ZeroTest}(\mathbb{H},\mathbb{H}\cup\mathbb{K})$  protocol to show:

$$G([sel_j(X)]_{j \in [\ell_s]}, [z_j(X)]_{j \in [\ell_z]}) = 0 \text{ over } \mathbb{H}$$

(3) To prove the copy constraints, P and V engage in MultisetCheck $(\mathbb{H}, \mathbb{H} \cup \mathbb{K})$  protocol to show:

$$\bigcup_{j \in [\ell_x]} \{\!\!\{(z_j(X), X + j \cdot \ell_z)\}\!\!\} = \bigcup_{j \in [\ell_x]} \{\!\!\{(z_j(X), \sigma_j(X))\}\!\!\} \text{ over } \mathbb{H}.$$

(4) To prove correctness of z, P and V engage in  $\text{ZeroTest}(\mathbb{H}_x, \mathbb{H} \cup \mathbb{K})$  protocol to prove  $z_0(X) - x(X) = 0$  over  $\mathbb{H}_x$ .

# Figure 6: PLONK polyIOP [GWC19].

• Define  $\mathbb{V}_{in} \leq \mathbb{V}_x$  as the multiplicative subgroup of size  $d_x/2$  generated by  $\mu^{\frac{2nd}{d_x}}$  and  $\mathbb{V}_{out} = \mu^{\frac{2nd}{d_x}} \mathbb{V}_{in}$  as the other coset of  $\mathbb{V}_{in}$  in  $\mathbb{V}_x$  encoding the two parts of the machine execution statement.

Using this notation, Figure 7 provides details of the indexer for Mux-PLONK. Table selector polynomials  $[tsel_j]_{j \in [\ell_s]}$ and table permutation polynomials  $[t\sigma_j]_{j \in [\ell_z]}$  encode the full instruction set, encoding each instruction  $i \in [\ell]$  within coset  $\omega^i \mathbb{V}$  of  $\mathbb{H}$ . Figure 8 then provides details of the proving protocol. A vector lookup (Section 4) is employed to lookup the appropriate executed instructions and encode them within new selector polynomials  $[sel_j]_{j \in [\ell_s]}$  and new permutation polynomials  $[\sigma'_j]_{j \in [\ell_z]}$  where now each executed instruction is encoded within coset  $\mu^i \mathbb{V}$  of  $\mathbb{G}$  for  $i \in [n]$ . As discussed in the overview (Section 2.3), the resulting permutation polynomials  $[\sigma'_j]_{j \in [\ell_z]}$  are malformed in that they no longer encode a permutation: all cosets evaluate to  $[d\ell_z]$ . Permutation polynomials  $[\sigma_j]_{j \in [\ell_z]}$  are constructed to offset the evaluations of each coset  $\mu^i \mathbb{V}$  by  $id\ell_z$  to recover the permutation.

Given these vector-lookup constructed (and edited) PLONK index polynomials, we are almost ready to apply PLONK directly to a polynomial x encoding the witnesses of the executed machine computation. The last step is to prove that polynomial x correctly encodes the input-output corresondence of each sequence of instructions. We highlight a technical difficulty that arises in this step.

In proving the correspondence of inputs, we ask the prover to perform a ZeroTest over the set  $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$  where  $\mathbb{G}_{in} = \bigcup_{i=0}^{n-1} \mu^i \mathbb{V}_{in}$ . However, notice that  $Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}(X) = \prod_{i \in [n]} Z_{\mu^i \mathbb{V}_{in}}(X)$  is not succinct and the fastest algorithm to compute this product incurs  $O(|\mathbb{G}_{in} \setminus \mathbb{V}_{in}|\log^2(|\mathbb{G}_{in} \setminus \mathbb{V}_{in}|))$  cost [GK22] which cannot be afforded by the verifier. Instead, we ask the prover to send  $f = Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}$  and prove that f satisfies the properties of a vanishing polynomial. One way is to evaluate f at some random point r and check if it agrees with  $Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}(r)$ . Recall that  $[Z_{\mu^i \mathbb{V}_{in}}(X) = X^{d_x i/2} - \mu^{d_x i/2}]_{i \in [n]}$  since  $\mathbb{V}_{in}$  is of order  $d_x/2$ . To prove wellformedness, the prover creates new polynomials to accumulate the products of  $Z_{\mu^i \mathbb{V}_{in}}(r)$  using induction. Since there are n-1 terms in multiplication, we define group  $\mathbb{G}_n = \langle \mu^d \rangle$  of order n to capture each intermediate result of the multiplication and skip the first element. To be concrete, the prover computes a polynomial h to encode the constant factor  $\mu^{d_x i/2}$  of  $Z_{\mu^i \mathbb{V}_{in}}(r)$  at  $\mu^{di}$ , and a polynomial g to encode the intermediate product up to  $i^{th}$  item in multiplication  $\prod_{k=1}^{i} Z_{\mu^k \mathbb{V}_{in}}(r)$  at  $\mu^{di}$ . Then we use standard

$$\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{plonk}}] = \begin{cases} \left( \left(\mathbb{F}, \mathbb{G}, \mathbb{H}, \mathbb{V}, \mathbb{V}_{x}, \mathbb{V}_{in}, G, \ell_{s}, \ell_{z}, [sel_{i}, \sigma_{i}]_{i \in [\ell]}\right), \\ ([\mathbb{I}x_{0}], [\mathbb{I}x_{n}]), \\ ([\mathrm{inst}_{j}, \mathrm{mem}_{j}, \mathrm{w}_{j}]_{j \in [n]}, x_{0}, x_{n}) \\ x_{0}(\mathbb{V}_{in}) = [\mathrm{inst}_{0}, \mathrm{mem}_{0}] \\ x_{n}(\mathbb{V}_{out}) = [\mathrm{inst}_{n}, \mathrm{mem}_{n}] \\ \vdots \\ \int_{j \in [n]} \left( (\mathbb{F}, \mathbb{V}, \mathbb{V}_{x}, sel_{\mathrm{inst}_{j}}, \sigma_{\mathrm{inst}_{j}}, G), \\ [\mathrm{inst}_{j}, \mathrm{mem}_{j}, \mathrm{inst}_{j+1}, \mathrm{mem}_{j+1}], \\ w_{j} \end{array} \right) \in \mathsf{R}_{\mathsf{plonk}} \end{cases}$$

$$\frac{\mathsf{Mux}\operatorname{PLONK}.\mathsf{Setup}(\lambda): \operatorname{Return} \perp \\ \mathsf{Mux}\operatorname{PLONK}.\mathsf{Index}(\bot, (\mathbb{F}, \mathbb{G}, \mathbb{H}, \mathbb{V}, \mathbb{V}_{x}, \mathbb{V}_{in}, G, \ell_{s}, \ell_{z}, [sel_{i}, \sigma_{i}]_{i \in [\ell]}) \right) \\ (1) \operatorname{Compute the prover parameter polynomials for each instruction index using the PLONK indexer. \\ \left[ \left( \begin{array}{c} pp_{i} \leftarrow ([sel_{i,j}]]_{j \in [\ell_{s}]}, [[\sigma_{i,j}]]_{j \in [\ell_{z}]}, G), \\ vp_{i} \leftarrow (([[sel_{i,j}]]]_{j \in [\ell_{s}]}, [[\sigma_{i,j}]]_{j \in [\ell_{s}]}, G), \\ vp_{i} \leftarrow (([sel_{i,j}]]]_{j \in [\ell_{s}]}, [[\sigma_{i,j}]]_{j \in [\ell_{s}]}, G), \\ vp_{i} \leftarrow ([[sel_{i,j}]]_{j \in [\ell_{s}]}], \sigma_{i,j}, \sigma_{i,j}, \sigma_{i,j}, \sigma_{i,j}] \right) \\ (2) \operatorname{Construct} table selector polynomials [tsel_{j}]_{j \in [\ell_{s}]} and table permutation polynomials [t\sigma_{j}]_{j \in [\ell_{s}]} over \mathbb{H} by setting the evaluations of the cosets [\omega^{i}\mathbb{V}]_{i \in [\ell]}: \\ [[tsel_{j}(\omega^{i}\mathbb{V}) = sel_{i,j}(\mathbb{V})]_{i \in [\ell]}]_{j \in [\ell_{s}]} [[t\sigma_{j}(\omega^{i}\mathbb{V}) = \sigma_{i,j}(\mathbb{V})]_{i \in [\ell]}]_{j \in [\ell_{s}]} \\ (3) \operatorname{Return} (pp \leftarrow ([tsel_{j}]_{j \in [\ell_{s}]}, [t\sigma_{j}]_{j \in [\ell_{s}]}, G), vp \leftarrow (([tsel_{j}]_{j \in [\ell_{s}]}, [[t\sigma_{j}]]_{j \in [\ell_{s}]}, G)) \end{cases}$$

Figure 7: Mux-PLONK: Setup algorithm encoding PLONK computation commitment values for each instruction into cosets of an evaluation domain for the machine commitment.

induction techniques to prove the induction is correct over  $\mathbb{G}_n$  skipping the first element:

- $L_{1,\mathbb{G}_n}(X)(h(X)-1) = 0$  over  $\mathbb{G}_n: h(1) = 1$  as the start of the induction.
- $(\mu^{d_x/2} \cdot h(X) h(\mu^d X))(X \mu^{d(n-1)}) = 0$  over  $\mathbb{G}_n$ : The next element in  $\mathbb{G}_n$  is equal to  $\mu^{d_x/2}$  times the previous element excluding the last one. Since the first element is set to 1, this ensures that each element is set to the next power of  $\mu^{\frac{d_x}{2}}$ .
- $L_{1,\mathbb{G}_n}(X)(g(X)-1) = 0$  over  $\mathbb{G}_n: g(1) = 1$  as the starting of the induction.
- $(X-1)(g(X) g(X/\mu^d) \cdot (r^{d_X/2} h(X)) = 0$  over  $\mathbb{G}_n$ : This enforces that the g's evaluation on the  $i^{th}$  element (in  $\mathbb{G}_n$ ) is equal to  $r^{d_X/2} h(X) = r^{d_X/2} \mu^{d_Xi/2}$  multiplied by g's evaluation on the  $(i-1)^{th}$  element. The check excludes the last and the first one element. Since the first element is set to 1, this ensures that  $i^{th}$  element is set to the accumulated product  $\prod_{k=1}^i Z_{\mu^k \mathbb{V}_{in}}(r)$  up to *i*.

Finally, the prover can prove  $f(r) = g(\mu^{d(n-1)}) = \prod_{i \in [n]} Z_{\mu^i \mathbb{V}_{in}}(r)$  by evaluating g(X) at  $\mu^{d(n-1)}$  using Lagrange polynomial and query f(r).

**Security.** We prove the completeness, knowledge soundness, and zero knowledge of Mux-PLONK in the following two theorems. The zero-knowledge of Mux-PLONK is achieved through our zero-knowledge compiler (see Appendix B) observing that Mux-PLONK is domain-restriction admissible with one exception that we handle explicitly.

**Theorem 8.** Mux-PLONK for  $R_{MExe,n}[R_{plonk}]$  (Figure 8) is complete and knowledge sound with negligible error

*Proof.* Mux-PLONK for  $R_{MExe,n}[R_{plonk}]$  (Figure 8) inherits the completeness error from the underlying CosetLkup look up protocol, and for any adversary A against knowledge soundness, we provide an extractor X using  $X_{zt}$ , an

| Mux-PLONK.P        | $ \begin{pmatrix} ([tsel_j]_{j \in [\ell_s]}, [t\sigma_j]_{j \in [\ell_z]}, G), \\ ([x_0]], [x_n]), \\ ([inst_i, mem_i, w_i]_{i=0}^n, x_0, x_n) \end{pmatrix} $   | $ \end{pmatrix} \leftrightarrow Mux-PLONK.V \left( \begin{array}{c} () \\ () \end{array} \right) $ | $\llbracket tsel_j \rrbracket]_{j \in [\ell_s]}, \llbracket t\sigma_j \rrbracket]_{j \in [\ell_z]}, C$ $[x_0], \llbracket x_n \rrbracket)$ | $\tilde{x}$ ), )   |
|--------------------|---|--|--|--|
| (1) P computes, se | ends polynomials $[sel_j]_{j \in \ell_s}$ and $[d]_{j \in \ell_s}$  | $[\sigma'_j]_{j \in \ell_z}$ , and proves their w  | ellformedness with respect t   | o instruction tables:                                    |
| (a) Construct p    | olynomials $[sel_j]_{j \in \ell_s}$ and $[\sigma'_j]_{j \in \ell_s}$  | $_{z}$ using evaluations over $\mathbb{G}$   | setting the evaluation of each   | h coset $\mathbb V$ in $\mathbb G$ to the corresponding  |
| table entry        | for instruction $inst_i$ :  |  |  |  |
|                    | $\Big[ \big[ sel_j(\mu^i \mathbb{V}) = tsel_j(\omega^i \mathbb{V}) \big] = tsel_j(\omega^i \mathbb{V}) \Big]$ | $[inst_i \mathbb{V})]_{i \in [n]}\Big]_{j \in [\ell_s]}$   | $\left[ \left[ \sigma_j'(\mu^i \mathbb{V}) = t \sigma_j(\omega^{\text{inst}_i} \mathbb{V}) \right] \right]$                                | $\left[ \right]_{i\in[n]} \left]_{j\in[\ell_z]} \right]$ |
| (b) P and V en     | gage in $(\ell_s + \ell_z)$ -CosetLkup to pr  | ove  |  |  |
|                    |   |  |  |  |

$$\left\{ [sel_j(\mu^i \mathbb{V})]_{j \in \ell_s}, [\sigma'_j(\mu^i \mathbb{V})]_{j \in \ell_z} \right\}_{i \in [n]} \subseteq \left\{ [tsel_j(\omega^i \mathbb{V})]_{j \in \ell_s}, [t\sigma_j(\omega^i \mathbb{V})]_{j \in \ell_z} \right\}_{i \in [\ell]}$$

(2) P computes and sends offset  $[\sigma_j]_{j \in \ell_z}$  to recover global permutation, and proves wellformedness:

(a) P computes and sends offset polynomial s, and proves its well-formedness.

-P computes and sends s over G setting the evaluation of coset of 
$$[\mu^i \mathbb{V}]_{i \in [n]}$$
 in G to be the shift  $id\ell_z$ :  $[s(\mu^i \mathbb{V}) = id\ell_z]_{i \in [n]}$ .

- $-\mathsf{P} \text{ and }\mathsf{V} \text{ engage in }\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}\cup\mathbb{G}) \text{ to prove } s(X)=0 \text{ over }\mathbb{V}.$
- $-\mathsf{P} \text{ and }\mathsf{V} \text{ engage in }\mathsf{ZeroTest}(\mathbb{G},\mathbb{H}\cup\mathbb{G}) \text{ to prove } (d\ell_z+s(X)-s(\mu X))(Z_{\mu^{n-1}\mathbb{V}}(X))=0 \text{ over }\mathbb{G}.$
- (b) P computes and sends polynomials:  $\left[ \left[ \sigma_j(\mu^i \mathbb{V}) = id\ell_z + \sigma'_j(\mu^i \mathbb{V}) \right]_{i \in [n]} \right]_{j \in [\ell_z]}$ .
- (c) P and V engage in ZeroTest( $\mathbb{G}, \mathbb{H} \cup \mathbb{G}$ ) to prove  $\left[\sigma_j(X) = \sigma'_j(X) + s(X)\right]_{i \in [\ell_r]}$  over  $\mathbb{G}$ .

(3) P computes and sends statement polynomial x, and proves its well-formedness.

(a) P computes and sends x by setting evaluations of each coset of  $\mathbb{V}$  in  $\mathbb{G}$  equal to the statement for each step of execution.

 $\left[ x(\mu^{i} \mathbb{V}) = \left[ \text{inst}_{i}, \text{mem}_{i}, \text{inst}_{i+1}, \text{mem}_{i+1}, \mathbf{w}_{i} \right] \right]_{i=0}^{n-1} \text{ s.t. } \left[ x(\mu^{i} \mathbb{V}_{in}) = \left[ \text{inst}_{i}, \text{mem}_{i} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ \text{inst}_{i+1}, \text{mem}_{i+1} \right] \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ x(\mu^{i} \mathbb{V}_{out}) \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ x(\mu^{i} \mathbb{V}_{out}) \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) = \left[ x(\mu^{i} \mathbb{V}_{out}) \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) \right]_{i=0}^{n-1} \left[ x(\mu^{i} \mathbb{V}_{out}) \right]_{i=0}^{n-1} \left[ x$ 

- (b) P and V prove initial input with  $\operatorname{ZeroTest}(\mathbb{V}_{in}, \mathbb{H} \cup \mathbb{G})$  to prove  $x(X) = x_0(X)$  over  $\mathbb{V}_{in}$ .
- (c) P and V prove final output with  $\operatorname{ZeroTest}(\mathbb{V}_{out}, \mathbb{H} \cup \mathbb{G})$  to prove  $x(\mu^{n-1}X) = x_n(X)$  over  $\mathbb{V}_{out}$ .
- (d) P proves input carryover:  $x(\mu^i \mathbb{V}_{in}) = x(\mu^{i-1} \mathbb{V}_{out})$  for  $i \in [1, n)$ .

-Define  $\mathbb{G}_{in} = \bigcup_{i=0}^{n-1} \mu^i \mathbb{V}_{in}$  and  $\mathbb{G}_n = \langle \mu^d \rangle$  as as the subgroup of  $\mathbb{G}$  of size n. P computes and sends  $f = Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}$  to V.

 $-\mathsf{V}$  sends random challenge  $r \in \mathbb{F} \setminus (\mathbb{H} \cup \mathbb{G})$ .

–P computes and sends polynomials g,h defined using the following evaluations over  $\mathbb{G}_n$ .

$$g(1) = 1 \qquad [g(\mu^{di}) = \prod_{k=1}^{i} Z_{\mu^{k} \mathbb{V}_{in}}(r)]_{i=1}^{n-1}, \qquad h(1) = 1 \qquad [h(\mu^{di}) = \mu^{d_{x}i/2}]_{i=1}^{n-1}.$$

 $-\mathsf{P} \text{ and }\mathsf{V} \text{ engage in }\mathsf{ZeroTest}(\mathbb{G}_n,\mathbb{H}\cup\mathbb{G}) \text{ to prove } L_{1,\mathbb{G}_n}(X)(h(X)-1)=0 \text{ over }\mathbb{G}_n.$ 

 $-\mathsf{P} \text{ and } \mathsf{V} \text{ engage in } \mathsf{ZeroTest}(\mathbb{G}_n,\mathbb{H}\cup\mathbb{G}) \text{ to prove } (\mu^{d_X/2}\cdot h(X)-h(\mu^dX))(X-\mu^{d(n-1)})=0 \text{ over } \mathbb{G}_n.$ 

-P and V engage in ZeroTest( $\mathbb{G}_n, \mathbb{H} \cup \mathbb{G}$ ) to prove  $L_{1,\mathbb{G}_n}(X)(g(X)-1) = 0$  over  $\mathbb{G}_n$ 

-P and V engage in ZeroTest( $\mathbb{G}_n, \mathbb{H} \cup \mathbb{G}$ ) to prove  $(X-1)(g(X) - g(X/\mu^d) \cdot (r^{d_X/2} - h(X)) = 0$  over  $\mathbb{G}_n$ .

 $-\mathsf{P} \text{ and } \mathsf{V} \text{ engage in } \mathsf{ZeroTest}(\mathbb{G}_n,\mathbb{H}\cup\mathbb{G}) \text{ to prove } L_{\mu^{d(n-1)},\mathbb{G}_n}(X)(g(X)-f(r))=0 \text{ over } \mathbb{G}_n \text{ where } \mathsf{V} \text{ queries } f \text{ on } r.$ 

-P and V engage in ZeroTest( $\mathbb{G}_{in} \setminus \mathbb{V}_{in}, \mathbb{H} \cup \mathbb{G}$ ) to prove  $x(X) = x(\mu^{nd/d_x - 1}X)$  over  $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$  using  $f = Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}$ .

(4) P and V engage in PLONK proving protocol with vector-lookup constructed index polynomials:

$$\mathsf{PLONK}.\mathsf{P}\left(\begin{array}{c} \left([sel_{j}]_{j\in[\ell_{s}]},[\sigma_{j}]_{j\in[\ell_{z}]},G\right),([\![x]\!],\mathbb{G}),\\ (\bot,[inst_{i},mem_{i},inst_{i+1},mem_{i+1},w_{i}]_{i=0}^{n}) \end{array}\right) \leftrightarrow \mathsf{PLONK}.\mathsf{V}\left(\left([[\![tsel_{j}]\!]]_{j\in[\ell_{s}]},[[\![t\sigma_{j}]\!]]_{j\in[\ell_{z}]},G\right),([\![x]\!],\mathbb{G})\right)$$

Figure 8: Mux-PLONK: PLONK polyIOP with vector lookup for machine execution.

extractor for ZeroTest, using  $X_{plonk}$ , an extractor for PLONK, and  $X_{lk}$ , an extractor for CosetLkup, such that

$$\begin{split} &\mathsf{Adv}^{\text{sound}}_{\mathsf{Mux-PLONK},\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{plonk}}],\mathsf{X},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{V}_{in},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{V}_{out},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 5\mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{G}_{n},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}_{n}|+1),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{G}_{n}\setminus\mathbb{V}_{in},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \frac{|\mathbb{G}_{in}|}{|\mathbb{F}|-|\mathbb{G}|-|\mathbb{H}|} \\ &+ \mathsf{Adv}^{\text{sound}}_{(\ell_{s}+\ell_{z})-\mathsf{CosetLkup}(\mathbb{G},\mathbb{H}),\mathsf{R}_{\mathsf{vlkup}},\mathsf{X}_{\mathsf{lk}},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{G},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \ell_{s}\cdot\mathsf{Adv}^{\text{sound}}_{\mathsf{ZeroTest}(\mathbb{G},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\text{sound}}_{\mathsf{PLONK}(\mathbb{G},\ell_{s},\ell_{z}),\mathsf{R}_{\mathsf{plonk}},\mathsf{X}_{\mathsf{plonk}},\mathcal{A}}(\lambda). \end{split}$$

<u>Completeness.</u> The success of steps (1-3) follow directly from the completeness of the underlying polyIOP subprotocols and the polynomial constructions of a valid prover. All that remains to show is the success of execution of the Plonk polyIOP in step (4). The Plonk index polynomials  $[sel_j]_{j \in [\ell_s]}, [\sigma_j]_{j \in [\ell_s]}$  are created such that

$$\left[\left[sel_{j}(\mu^{i}\mathbb{V}) = tsel_{j}(\omega^{inst_{i}}\mathbb{V})\right]_{i\in[n]}\right]_{j\in[\ell_{s}]} \quad \left[\left[\sigma_{j}(\mu^{i}\mathbb{V}) = t\sigma_{j}(\omega^{inst_{i}}\mathbb{V}) + id\ell_{z}\right]_{i\in[n]}\right]_{j\in[\ell_{z}]}$$

Since the outputs of selector polynomials are invariant and directly match the values encoded in the table, we have  $sel_j(\mu^i \mathbb{V}) = sel_{inst_i,j}, \forall i \in [n], j \in [\ell_s]$ . Since the outputs of permutation polynomials encoded in the table map to same subgroup  $\mathbb{V}$ , after shifting the *i*-th instruction *j*-th permutation polynomial maps to coset  $\mu^{i \cdot j} \mathbb{V}$ . Hence,  $\sigma_j(\mu^i \mathbb{V}) = \sigma_{inst_i,j}, \forall i \in [n], j \in [\ell_z]$ . Finally, because  $[x(\mu^i \mathbb{V}) = [inst_i, mem_i, inst_{i+1}, mem_{i+1}, w_i]]_{i=0}^{n-1}$ ,  $[[sel_{inst_i,j}]_{i \in [n]}]_{j \in [\ell_z]}$  and  $[[\sigma_{inst_i,j}]_{i \in [n]}]_{j \in [\ell_z]}$  are the correct index polynomials for Plonk and the completeness directly follows from that of Plonk.

Knowledge Soundness. We bound the advantage of adversary A by bounding the advantage of each of a series of game hops [BR06]. We define  $G_0 =$ 

SOUND<sup>A</sup><sub>Mux-PLONK,R<sub>MExe,n</sub>[R<sub>plonk</sub>], $\chi(\lambda)$ . The inequality above follows from the following claims that we will justify: (1)</sub>

$$\begin{split} &|\Pr[\mathbf{G}_{0}=1]-\Pr[\mathbf{G}_{1}=1]| \leq \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\mathbb{V}_{in},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\mathbb{V}_{out},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 5\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\mathbb{G}_{n},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}_{n}|+1),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\mathbb{G}_{n}\setminus\mathbb{V}_{in},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \frac{|\mathbb{G}_{in}|}{|\mathbb{F}|-|\mathbb{G}|-|\mathbb{H}|} \end{split}$$

(2)

$$\begin{split} |\Pr[G_1 = 1] - \Pr[G_2 = 1]| &\leq \mathsf{Adv}^{sound}_{(\ell_s + \ell_z) - \mathsf{CosetLkup}(\mathbb{G}, \mathbb{H}), \mathsf{R}_{\mathsf{vlkup}}, \mathsf{X}_{\mathsf{lk}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{V}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) + \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{G}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \ell_s \cdot \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{G}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

 $(3) \quad |\Pr[G_2 = 1] - \Pr[G_3 = 1]| \le \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{PLONK}(\mathbb{G}, \ell_s, \ell_z), \mathsf{R}_{\mathsf{plonk}}, \mathsf{X}_{\mathsf{plonk}}, \mathcal{A}}(\lambda)$   $(4) \quad \Pr[G_3 = 1] = 0$ 

Claim 1 argues for the well-formedness of the statement polynomial x. Claim 2 argues for the well-formedness of the index polynomials  $[sel_j]_{j \in [\ell_s]}$ ,  $[\sigma_j]_{j \in [\ell_z]}$  to invoke Plonk. Claim 3 argues that the Plonk relation for the unrolled execution is satisfied. Lastly Claim 4 argues that the constructed extractor always succeeds for an accepting verifier.

Claim 1: In this first step, we argue for the well-formedness of the statement polynomial x in that it (1) properly encodes  $x_0$  and  $x_n$  from the statement, and (2) repeats the instruction and memory components for the part of the computation representing each of the execution steps. Consider the following tests run in step (3bc):

- $x(X) = x_0(X)$  over  $\mathbb{V}_{in}$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}}(\mathbb{V}_{in}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$ : Checks correct first statement.
- $x(\mu^{n-1}X) = x_n(X)$  over  $\mathbb{V}_{out}$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}}(\mathbb{V}_{out}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$ : Checks correct last statement.

In  $G_1$ , we invoke the zero test extractor  $X_{zt}$  to extract the witnesses and check if the above hold, returning zero otherwise. We can begin to bound the difference in probability of returning 1 in  $G_0$  and  $G_1$  through an identical-until-bad argument with a series of hybrid games by setting a "bad" flag [BR06] in this failure case (i.e., when the extractor fails). In each hybrid, we bound the probability of the bad flag being set exactly by the advantage against the soundness of the zero test protocol. The advantage of the zero test for the test is included in the bullet point.

Now consider the tests run in step (3d):

- $L_{1,\mathbb{G}_n}(X)(h(X)-1) = 0$  over  $\mathbb{G}_n$  with advantage  $\operatorname{Adv}^{\operatorname{sound}}_{\operatorname{ZeroTest}}(\mathbb{G}_n, \mathbb{H} \cup \mathbb{G}, 2|\mathbb{G}_n|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks base case h(1) = 1.
- $(\mu^{\frac{d_X}{2}} \cdot h(X) h(\mu^d X))(X \mu^{d(n-1)}) = 0$  over  $\mathbb{G}_n$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}}(\mathbb{G}_n, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}_n|+1), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$ : Checks inductive step:

$$\left[h(\mu^{dj}) = \mu^{d_x j/2}\right]_{j=1}^{n-1}$$

- $L_{1,\mathbb{G}_n}(X)(g(X)-1) = 0$  over  $\mathbb{G}_n$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{G}_n, \mathbb{H} \cup \mathbb{G}, 2|\mathbb{G}_n|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks base case g(1) = 1.
- $(X-1)(g(X) g(X/\mu^d) \cdot (r^{\frac{d_X}{2}} h(X)) = 0$  over  $\mathbb{G}_n$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{G}_n, \mathbb{H} \cup \mathbb{G}, 2|\mathbb{G}_n|+1), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks inductive step:

$$\left[g(\mu^{dj}) = g(\mu^{d(j-1)}) \cdot (r^{\frac{d_x}{2}} - \mu^{d_x j/2}) = \prod_{k=1}^j Z_{\mu^k \mathbb{V}_{in}}(r)\right]_{j=1}^{n-1}$$

•  $L_{\mu^{d(n-1)},\mathbb{G}_n}(X)(g(X) - f(r)) = 0$  over  $\mathbb{G}_n$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{G}_n, \mathbb{H} \cup \mathbb{G}, 2|\mathbb{G}_n|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks that  $f(r) = Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}(r).$ 

In this last bullet point, by checking that  $f = Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}$  on a random verifier challenge r, we can set a bad flag and return 0 if  $f \neq Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}$  and bound the probability of this case using the Schwartz-Zippel lemma to at most  $\frac{|\mathbb{G}_{in}|}{|\mathbb{F}|-|\mathbb{G}|-|\mathbb{H}|}$ .

Finally the well-formedness of x is completed with the last test:

•  $x(X) = x(\mu^{\frac{nd}{d_x}-1}X)$  over  $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$  using  $f = Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{G}_n \setminus \mathbb{V}_{in}, \mathbb{H} \cup \mathbb{G}, 2|\mathbb{G}|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks that the statement is copied over between instructions,  $x(\mu^j \mathbb{V}_{in}) = x(\mu^{j-1} \mathbb{V}_{out}).$ 

In addition to setting the bad flag with respect to the well-formedness of f,  $G_1$  employs  $X_{zt}$  for all the tests described above and aborts if the extractor fails. Given the claimed advantage bounds for each of these hybrids, this completes the argument for Claim 1 and results in the claimed probability bound.

*Claim 2:* In this second step, we argue for the well-formedness of the selector polynomials  $[sel_j]_{j \in [\ell_s]}$  and the shifted permutation polynomials  $[\sigma_j]_{j \in [\ell_z]}$ , and that they correctly match the desired polynomials output by the index algorithm for the unrolled computation.

First consider the vector lookup in step (1). G<sub>2</sub> employs the extractor X<sub>lk</sub> to check the valid lookup relation between  $[sel_j]_{j \in [\ell_s]}, [\sigma'_j]_{j \in [\ell_z]}$  and the table polynomials  $[tsel_j]_{j \in [\ell_s]}, [t\sigma_j]_{j \in [\ell_z]}$ , aborting if the extractor fails. The probability of the bad flag being set is bounded by the soundness advantage of the vector lookup protocol,

•  $(\ell_s + \ell_z)$ -CosetLkup for  $[sel_j]_{j \in [\ell_s]}$ ,  $[\sigma'_j]_{j \in [\ell_z]}$  with table polynomials  $[tsel_j]_{j \in [\ell_s]}$ ,  $[t\sigma_j]_{j \in [\ell_z]}$  with advantage  $\operatorname{Adv}^{\text{sound}}_{(\ell_s + \ell_z) - \operatorname{CosetLkup}(\mathbb{G}, \mathbb{H}), \mathbb{R}_{\text{vlkup}}, \mathbb{X}_{\text{lk}, \mathcal{A}}}(\lambda)$ .

Next consider the steps to show the well-formedness of the shift polynomial s in step (2a):

• s(X) = 0 over  $\mathbb{V}$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}}(\mathbb{V}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$ : Checks base case  $s(\mathbb{V}) = 0$ .

•  $(d\ell_z + s(X) - s(\mu X))(Z_{\mu^{n-1}\mathbb{V}}(X)) = 0$  over  $\mathbb{G}$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{G}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}| + |\mathbb{V}|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks inductive step:  $[s(\mu^i \mathbb{V}) = id\ell_z]_{i=1}^{n-1}$ .

And then the lookup polynomials  $[\sigma'_j]_{j \in [\ell_z]}$  are shifted to construct  $[\sigma_j]_{j \in [\ell_z]}$  that represent valid permutation polynomials.

•  $\left[\sigma_j(X) = \sigma'_j(X) + s(X)\right]_{j \in [\ell_z]}$  over  $\mathbb{G}$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}}(\mathbb{G}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$ : Checks

$$\left[\left[\sigma_j(\mu^i \mathbb{V}) = t\sigma_j(\omega^{inst_i} \mathbb{V}) + id\ell_z\right]_{i \in [n]}\right]_{j \in [\ell_z]}$$

Again, since the Plonk selector polynomials match exactly the row for  $inst_i$  in the table,  $sel_j(\mu^i \mathbb{V}) = sel_{inst_i,j}, \forall i \in [n], j \in [\ell_s]$ . Since the *i*-th instruction *j*-th Plonk permutation polynomials are shifted from row  $inst_i$  in the permutation table *j* by  $id\ell_z, \sigma_j(\mu^i \mathbb{V}) = \sigma_{inst_i,j}, \forall i \in [n], j \in [\ell_z]$ .

As before  $G_2$  employs the zero test extractor  $X_{zt}$  to check the above tests and aborts if the extractor fails. The probability bound comes from a series of hybrids bounding each hybrid by the soundness advantage for the zero test.

*Claim 3:* Now we argue that the statement polynomial satisfies the unrolled execution Plonk. From Claim 2, we have that  $[sel_j]_{j \in [\ell_s]}$ ,  $[\sigma_j]_{j \in [\ell_z]}$  represent valid selector polynomials and permutation polynomials, respectively. G<sub>3</sub> employs the Plonk extractor X<sub>plonk</sub> to check the Plonk relation in step (5), and aborts if the extractor fails.

• PLONK for  $[sel_j]_{j \in [\ell_s]}, [\sigma_j]_{j \in [\ell_z]}$  and statement x with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{PLONK}(\mathbb{G}, \ell_s, \ell_z), \mathsf{R}_{\mathsf{plonk}}, \mathsf{X}_{\mathsf{plonk}}, \mathcal{A}(\lambda)$ .

*Claim 4:* We conclude by constructing an extractor X that always succeeds on a verifying prover. As argued in the completeness proof, the Plonk relation for  $[sel_j]_{j \in [\ell_s]}, [\sigma_j]_{j \in [\ell_z]}$  is satisfied, and  $x(\mu^j \mathbb{V}) = [inst_j, mem_j, inst_{j+1}, mem_{j+1}, w_j]$ . In particular,  $x_0(\mathbb{V}_{in}) = [inst_0, mem_0]$  and  $x_n(\mathbb{V}_{out}) = [inst_n, mem_n]$ . Thus, our extractor X simply employs the Plonk extractor X plonk to extract x and outputs,

$$\left(\left[\operatorname{inst}_{j}, \operatorname{mem}_{j}, \operatorname{w}_{j}\right]_{i=0}^{n}, x_{0} = \left[\operatorname{inst}_{0}, \operatorname{mem}_{0}\right], x_{n} = \left[\operatorname{inst}_{n}, \operatorname{mem}_{n}\right]\right)$$

By Claim 3, in  $G_3$ , if the verifier succeeds, the Plonk extractor always succeeds and so similarly, so will our constructed extractor.

**Theorem 9.** The compiled polyIOP using the compiler in Figure 19 of Mux-PLONK for  $R_{MExe,n}[R_{plonk}]$  (Figure 8) is honest-verifier zero-knowledge.

*Proof.* It is evident from Figure 8 that all subprotocols are ZT-Based polyIOPs and the verifier never queries outside these polyIOPs except in step 3(d). In step 3(d), V queries f on r to prove  $f = Z_{\mathbb{G}_{in} \setminus \mathbb{V}_{in}}$ . The purposes of steps 4 to 8 are to prove f is a vanishing polynomial over a public set,  $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$ . Since it is a public set, the polynomials used in steps 3d, and 4-8 do not encode values dependent on the witness so we can exclude the zero tests and queries from the zero-knowledge compiler. The remainder of the protocol satisfies the template of a ZT-Based polyIOP, so we apply the zero knowledge compiler to all steps except 3(d) items 4-8, and HVZK follows directly from compiler guarantees.

#### 5.2 Extensions to Other PolyIOPs

**Mux-HyperPLONK: Supporting a linear-time prover via a multivariate polyIOP.** HyperPLONK [CBBZ23] is a multivariate polyIOP for the PLONK arithmetization (Definition 7), mirroring the approach of the PLONK polyIOP but adapting the techniques for multivariate polynomials with vector encodings over the boolean hypercube. By using multivariate polynomials, for which we have linear-time polynomial commitment algorithms, this approach leads to a linear-time prover, avoiding the quasilinear cost FFTs of univariate polynomial commitment algorithms.

Using our multivariate vector lookup argument (Appendix C), we build Mux-HyperPLONK using the same strategy as Mux-PLONK but adapted for the multivariate setting. As in the adaption of the vector lookup argument, we use the

Binius hypercube shift operator [DP23] to mimic coset shifts from the univariate setting. We defer the full construction and security proofs for the protocol to Appendix D.

**Mux-Marlin: Supporting alternate R1CS instruction arithmetization.** While Mux-PLONK and Mux-HyperPLONK encode instructions using the Plonkish arithmetization, Marlin uses a rank-1 constraint system (R1CS) arithmetization which offers encoding tradeoffs [STW23a]. We show that the Marlin polyIOP can be adapted for use with our machine computation approach by building Mux-Marlin. In order to do this, we propose a variant of Marlin that we call ZK-Marlin that is *index-private*, a stronger notion of zero-knowledge introduced by Boneh et al [BNO21]. The full details are given in Appendix E.

# **6** Evaluation

In this section, we evaluate our proposed protocols for succinct vector lookups and machine computation. We report benchmarks on an implementation of our univariate vector lookup protocol, CosetLkup, as well as a cost accounting of Mux-PLONK based on these benchmarks.

#### 6.1 Succinct Vector Lookup Evaluation

**Implementation.** We implement CosetLkup (Figure 5) in Rust using the arkworks libraries for zkSNARK development [ark]. The implementation is generic to choice of pairing-friendly FFT-friendly curve and to choice of polynomial commitment scheme; we report on an instantiation over BLS12-381 with the constant-size MarlinKZG polynomial commitment scheme to target target 128 bits of security. We also implement a number of standard optimizations for polyIOP compilation including batching zero tests and batching polynomial evaluation openings (see Marlin compilation for more details [CHM<sup>+</sup>20]). We note our implementation supports zero-knowledge via the query-count blinding strategy of Marlin [CHM<sup>+</sup>20] instead of the zero-knowledge zero test strategy of our zero-knowledge compiler; the zero-knowledge zero test strategy is more general but less efficient, incurring up to a factor of two overhead in proof size and prover time. Our implementation is available open source <sup>4</sup>.

As a baseline comparison point, we also implement a vector commitment derived from the naive linear combination transformation to a succinct univariate element lookup protocol. More precisely, the succinct univariate element lookup protocol we use is a variant of Plookup [GW20] optimized using the lookup technique from Haböck [Hab22]; we refer to this construction as LC-Haböck. Recall this approach is not succinct with respect to vector size. The element lookup implementation component of LC-Haböck may be of independent interest as it is more efficient than previous succinct (constant-size) element lookup protocols. The benchmarks are run on a MacBook Pro (Apple M3 Pro Chip, 36 GB RAM, 12 cores).



Figure 9: Effect of vector size on prover time (left), verifier time (middle), proof size (right) for vector lookup protocols. Table size and lookup size are fixed to  $2^6$  and  $2^{10}$ , respectively.

<sup>&</sup>lt;sup>4</sup>https://github.com/lucasxia01/mux-proofs-impl



Figure 10: Effect of lookup size on prover time (left) and verifier time (middle left), and effect of table size on prover time (middle right) and verifier time (right). Vector size, table size and lookup size are fixed to  $2^{10}$ ,  $2^6$  and  $2^{10}$ , respectively.

| Protocol Proof size |  | size                                | Prover computation  |
|---------------------|--|-------------------------------------|---|
| Mux-PLONK           | $\boxed{63+4\ell_z+2\ell_s\mathbb{G}_1}$ | $62 + 5\ell_z + 4\ell_s \mathbb{F}$ | $\begin{array}{c} 14 v\text{-}MSM(2C\ell) + (37 + 4\ell_z + 2\ell_s) v\text{-}MSM(nC) \\ + 2 v\text{-}MSM(n) + 3 v\text{-}MSM(2nC\ell_z) \end{array}$ |
| PLONK for UC        | $8+2\ell_z\mathbb{G}_1$                  | $5+\ell_z\mathbb{F}$                | $(5+2\ell_z)$ v-MSM $(nC\ell)$ +3v-MSM $(2nC\ell\ell_z)$  |

Figure 11: Accounting of dominant costs Mux-PLONK where  $\ell$  is the number of instructions in the instruction set, C is the constraint size for a single instruction, and n is the number of instructions executed. Parameters  $\ell_s$  and  $\ell_z$  are the number of selectors and inputs, respectively, of the Plonkish encoding of C constraints for an instruction.  $\mathbb{G}_1$  refers to one group in a pairing-friendly group and  $\mathbb{F}$  refers to the scalar field modulo group order. Prover computation is dominated by variable-base multiscalar multiplications (v-MSM) for committing to and opening polynomial commitments. The costs for PLONK when applied to an unrolled universal circuit (UC) with  $nC\ell$  constraints is given as a comparison point.

**Benchmark results.** We aim to evaluate the prover time, verifier time, proof size, and commitment size of CosetLkup with respect to (1) vector size, (2) lookup size, and (3) table size. An important aspect of CosetLkup is succinctness with respect to vector size. In comparison to LC-Haböck, whose verifier costs and commitment size scale linearly with vector size, we find that CosetLkup begins to compare favorably to LC-Haböck at a vector size of around 8. For larger vector size, take for example size  $2^{10}$ , CosetLkup verifies in 6ms and has a constant commitment size of 192B,  $60 \times$  faster and  $1000 \times$  smaller than LC-Haböck. See Figure 9 for more details. The proof size of both protocols is constant: CosetLkup is approximately 3.2KB and LC-Haböck is 1.6KB.

With respect to lookup size and table size, both CosetLkup and LC-Haböck prover time scale quasilinearly with respect to the sum of lookup and table size, while verifier time is constant, shown in Figure 10. The CosetLkup prover incurs around a  $500 \times$  prover overhead on top of LC-Haböck to achieve succinctness for vector sizes of  $2^{10}$ . In contrast, recent proposals for succinct vector lookups based on cached quotients [CGG<sup>+</sup>23, CFF<sup>+</sup>23] scale quasilinearly with only the lookup size; online proof characteristics do not depend on table size. However, constant factor costs for these protocols are higher than that of CosetLkup.

# 6.2 Machine Computation Evaluation

We provide accounting of Mux-PLONK prover computation and proof size in Figure 11. For most practical instance sizes, the dominating cost for the prover is the multiscalar group multiplications (MSMs) for producing and evaluating Marlin-KZG polynomial commitments [CHM<sup>+</sup>20]; To put the prover costs and proof size into perspective, we compare against applying PLONK (with MarlinKZG) to an unrolled universal circuit, i.e., a proof system that incurs the cost of the full instruction set per executed instruction. Recall, prior to our work, this was the only approach to achieve zero-knowledge for machine computation without using IVC. For concrete comparisons, let us estimate with the parameters of a minimal CPU instruction set like RISC-V. Consider an instruction set of size  $\ell = 2^5$  instructions with  $C = 2^6$  constraints per instruction. We can consider programs on the order of  $n = 2^{11}$  executed instructions. We will also consider the basic circuit Plonkish arithmetization with  $\ell_s = 1$  and  $\ell_z = 3$ .

In terms of proof size, Mux-PLONK produces a constant-sized proof that is around  $10 \times$  the size of basic PLONK.

Even so, when instantiated with BLS12-381, the proof is a manageable size of approximately 11KB.

Before comparing the prover computation of Mux-PLONK with PLONK for unrolled universal circuits ( $nC\ell$ ), first consider the cost of PLONK for unrolled instruction circuits (nC) which Mux-PLONK runs as a subroutine. We estimate that the PLONK subroutine makes up approximately 20% of the prover computation of Mux-PLONK. In other words, Mux-PLONK incurs about a 2.5× overhead of simply proving the unrolled instructions with PLONK (which would not provide zero-knowledge). Now consider PLONK applied to the unrolled universal instruction circuits, which does provide zero-knowledge but incurs  $\ell = 2^5 \times$  more constraints. Here we estimate that the Mux-PLONK prover performs approximately 9× less work than the PLONK for universal circuits prover. In practice, a universal circuits approach would not incur the full  $\ell = 2^5 \times$  increase in constraint size as optimizations can be made to reuse logic across instructions. However, even so, this evaluation indicates Mux-PLONK is competitive with unrolled universal approaches and will compare even more favorably for expressive instruction sets with larger  $\ell$  and C.

## Acknowledgments

Zijing Di and Lucas Xia were funded by Stanford IOG Research Hub. Wilson Nguyen was partially funded by NSF, DARPA, the Simons Foundation, and NTT Research. Nirvan Tyagi was supported by NSF grant CNS-2120651 and by the Stanford Future of Digital Currency Initiative (FDCI). Any opinions, findings, and conclusions or recommendations expressed in the material are those of the authors and do not necessarily express reflect the views of DARPA and funding parties listed.

# References

| [ACF21]               | Thomas Attema, Ronald Cramer, and Serge Fehr. Compressing proofs of k-out-of-n partial knowledge. In <i>CRYPTO (4)</i> , volume 12828 of <i>Lecture Notes in Computer Science</i> , pages 65–91. Springer, 2021.   |
|-----------------------|--|
| [AFK22]               | Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In <i>TCC (1)</i> , volume 13747 of <i>Lecture Notes in Computer Science</i> , pages 113–142. Springer, 2022.  |
| [AHG22]               | Diego F. Aranha, Youssef El Housni, and Aurore Guillevic. A survey of elliptic curves for proof systems. <i>IACR Cryptol. ePrint Arch.</i> page 586, 2022.   |
| [AOS02]               | Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In <i>ASIACRYPT</i> , volume 2501 of <i>Lecture Notes in Computer Science</i> , pages 415–432. Springer, 2002.   |
| [ark]                 | arkworks zksnark ecosystem. https://arkworks.rs.   |
| [AS92]                | Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In <i>FOCS</i> , pages 2–13. IEEE Computer Society, 1992.  |
| [AST23]               | Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. <i>IACR Cryptol. ePrint Arch.</i> , page 1217, 2023.  |
| [Bab85]               | László Babai. Trading group theory for randomness. In STOC, pages 421-429. ACM, 1985.  |
| [BC23]                | Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. <i>IACR Cryptol. ePrint</i> Arch., page 620, 2023.  |
| [BCCT12]              | Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In <i>ITCS</i> , pages 326–349. ACM, 2012.  |
| [BCCT13]              | Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-<br>carrying data. In <i>STOC</i> , pages 111–120. ACM, 2013.  |
| [BCG <sup>+</sup> 13] | Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In <i>CRYPTO (2)</i> , volume 8043 of <i>Lecture Notes in Computer Science</i> , pages 90–108. Springer, 2013      |
| [BCG <sup>+</sup> 18] | Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In <i>ASIACRYPT (1)</i> , volume 11272 of <i>Lecture Notes in Computer Science</i> , pages 595–626. Springer, 2018 |
| [BCG <sup>+</sup> 19] | Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner. Linear-size constant-query iops for delegating computation. In <i>TCC (2)</i> , volume 11892 of <i>Lecture Notes in Computer Science</i> , pages 494–521. Springer, 2019.      |
| [BCG+20]              | Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: enabling decentralized private computation. In <i>IEEE Symposium on Security and Privacy</i> , pages 947–964. IEEE, 2020.  |

- [BCL<sup>+</sup>21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In CRYPTO (1), volume 12825 of Lecture Notes in Computer Science, pages 681–710. Springer, 2021.
- [BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In TCC (2), volume 12551 of Lecture Notes in Computer Science, pages 1–18. Springer, 2020.
- [BCR<sup>+</sup>19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In EUROCRYPT (1), volume 11476 of Lecture Notes in Computer Science, pages 103–128. Springer, 2019.
- [BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In TCC (B2), volume 9986 of Lecture Notes in Computer Science, pages 31–60, 2016.
- [BCTV14a] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In CRYPTO (2), volume 8617 of Lecture Notes in Computer Science, pages 276–294. Springer, 2014.
- [BCTV14b] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In USENIX Security Symposium, pages 781–796. USENIX Association, 2014.
- [BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In CRYPTO (1), volume 12825 of Lecture Notes in Computer Science, pages 649–680. Springer, 2021.
- [BEG<sup>+</sup>91] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. In FOCS, pages 90–99. IEEE Computer Society, 1991.
- [BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31. ACM, 1991.
- [BFR<sup>+</sup>13] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In SOSP, pages 341–357. ACM, 2013.
- [BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In *EUROCRYPT (1)*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020.
- [BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. *IACR Cryptol. ePrint Arch.*, page 1021, 2019.
- [BMRS21] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In *CRYPTO (4)*, volume 12828 of *Lecture Notes in Computer Science*, pages 92–122. Springer, 2021.
- [BNO21] Dan Boneh, Wilson Nguyen, and Alex Ozdemir. Efficient functional commitments: How to commit to private functions. *IACR Cryptol. ePrint Arch.*, page 1342, 2021.
- [BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In EUROCRYPT, volume 4004 of Lecture Notes in Computer Science, pages 409–426. Springer, 2006.
- [CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT (2)*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2023.
- [CCG<sup>+</sup>23] Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. Proof-carrying data from arithmetized random oracles. In EUROCRYPT (2), volume 14005 of Lecture Notes in Computer Science, pages 379–404. Springer, 2023.
- [CCS22] Megan Chen, Alessandro Chiesa, and Nicholas Spooner. On succinct non-interactive arguments in relativized worlds. In *EUROCRYPT* (2), volume 13276 of *Lecture Notes in Computer Science*, pages 336–366. Springer, 2022.
- [CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.
- [CFF<sup>+</sup>23] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees. *IACR Cryptol. ePrint Arch.*, page 1518, 2023.
- [CGG<sup>+</sup>23] Arka Rai Choudhuri, Sanjam Garg, Aarushi Goel, Sruthi Sekar, and Rohit Sinha. Sublonk: Sublinear prover plonk. *IACR Cryptol. ePrint Arch.*, page 902, 2023.
- [CHM<sup>+</sup>20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In *EUROCRYPT (1)*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, 2020.
- [DG23] Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In EUROCRYPT (2), volume 14005 of Lecture Notes in Computer Science, pages 531–562. Springer, 2023.
- [DMWG23] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak fiat-shamir attacks on modern proof systems. *IACR Cryptol. ePrint Arch.*, page 691, 2023.
- [DP23] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. *IACR Cryptol. ePrint Arch.*, page 1784, 2023.
- [EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. IACR Cryptol. ePrint Arch., page 1763, 2022.

- [FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *INDOCRYPT*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79. Springer, 2012.
- [GGHK22] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose \$\varsigma \$\\parsigma\$ protocols for disjunctions. In EUROCRYPT (2), volume 13276 of Lecture Notes in Computer Science, pages 458–487. Springer, 2022.
- [GHK23] Aarushi Goel, Mathias Hall-Andersen, and Gabriel Kaptchuk. Dora: Processor expressiveness is (nearly) free in zero-knowledge for RAM programs. *IACR Cryptol. ePrint Arch.*, page 1749, 2023.
- [GHKS22] Aarushi Goel, Mathias Hall-Andersen, Gabriel Kaptchuk, and Nicholas Spooner. Speed-stacking: Fast sublinear zero-knowledge proofs for disjunctions. IACR Cryptol. ePrint Arch., page 1419, 2022.
- [GK15] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 253–280. Springer, 2015.
- [GK22] Ariel Gabizon and Dmitry Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. *IACR Cryptol. ePrint Arch.*, page 1447, 2022.
- [GKK<sup>+</sup>22] Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michal Zajac. What makes fiat-shamir zksnarks (updatable SRS) simulation extractable? In *SCN*, volume 13409 of *Lecture Notes in Computer Science*, pages 735–760. Springer, 2022.
- [GLS<sup>+</sup>23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In CRYPTO (2), volume 14082 of Lecture Notes in Computer Science, pages 193–226. Springer, 2023.
- [GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In STOC, pages 291–304. ACM, 1985.
- [GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108. ACM, 2011.
- [GW20] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. *IACR Cryptol. ePrint Arch.*, page 315, 2020.
- [GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for occumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, page 953, 2019.
- [Hab22] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. IACR Cryptol. ePrint Arch., page 1530, 2022.
- [HK20] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In EUROCRYPT (3), volume 12107 of Lecture Notes in Computer Science, pages 569–598. Springer, 2020.
- [Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In STOC, pages 723–732. ACM, 1992.
- [KPPS20] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. MIRAGE: succinct arguments for randomized algorithms with applications to universal zk-snarks. In USENIX Security Symposium, pages 2129–2146. USENIX Association, 2020.
- [KS22] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *IACR Cryptol. ePrint Arch.*, page 1758, 2022.
- [KS23] Abhiram Kothapalli and Srinath T. V. Setty. Cyclefold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. IACR Cryptol. ePrint Arch., page 1192, 2023.
- [KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In CRYPTO (4), volume 13510 of Lecture Notes in Computer Science, pages 359–388. Springer, 2022.
- [KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010.
- [LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. J. ACM, 39(4):859–868, 1992.
- [LNS20] Jonathan Lee, Kirill Nikitin, and Srinath T. V. Setty. Replicated state machines without replicated execution. In *IEEE Symposium on Security and Privacy*, pages 119–134. IEEE, 2020.
- [Mic94] Silvio Micali. CS proofs (extended abstracts). In FOCS, pages 436–453. IEEE Computer Society, 1994.
- [NBS23] Wilson D. Nguyen, Dan Boneh, and Srinath T. V. Setty. Revisiting the nova proof system on a cycle of curves. In AFT, volume 282 of LIPIcs, pages 18:1–18:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [NDC<sup>+</sup>24] Wilson D. Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based snarks. *IACR Cryptol. ePrint Arch.*, page 416, 2024.
- [PFM+22] Luke Pearson, Joshua Brian Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. Plonkup: Reconciling plonk with plookup. *IACR Cryptol. ePrint Arch.*, page 86, 2022.
- [PK22] Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. IACR Cryptol. ePrint Arch., page 957, 2022.

- [Pol] Polygon zkevm. https://wiki.polygon.technology/docs/zkEVM/introduction.
- [RIS] Risc zero. https://www.risczero.com/docs/explainers.
- [RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In STOC, pages 49–62. ACM, 2016.
- [SAGL18] Srinath T. V. Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In OSDI, pages 339–356. USENIX Association, 2018.
- [Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *CRYPTO (3)*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020.
- [STW23a] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Customizable constraint systems for succinct arguments. *IACR Cryptol. ePrint Arch.*, page 552, 2023.
- [STW23b] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with lasso. *IACR Cryptol. ePrint Arch.*, page 1216, 2023.
- [Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. Found. Trends Priv. Secur., 4(2-4):117–660, 2022.
- [Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In TCC, volume 4948 of Lecture Notes in Computer Science, pages 1–18. Springer, 2008.
- [WSR<sup>+</sup>15] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In NDSS. The Internet Society, 2015.
- [XCZ<sup>+</sup>22] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. VERI-ZEXE: decentralized private computation with universal setup. *IACR Cryptol. ePrint Arch.*, page 802, 2022.
- [XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *CRYPTO (4)*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328. Springer, 2022.
- [YH23] Yibin Yang and David Heath. Two shuffles make a RAM: improved constant overhead zero knowledge RAM. *IACR Cryptol. ePrint Arch.*, page 1115, 2023.
- [YHH<sup>+</sup>23] Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. Batchman and robin: Batched and non-batched branching for interactive ZK. In *CCS*, pages 1452–1466. ACM, 2023.
- [YHH<sup>+</sup>24] Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. Tight zk cpu: Batched zk branching with cost proportional to evaluated instruction. *IACR Cryptol. ePrint Arch.*, page 456, 2024.
- [ZBK<sup>+</sup>22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In CCS, pages 3121–3134. ACM, 2022.
- [ZGK<sup>+</sup>18] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vram: Faster verifiable RAM with program-independent preprocessing. In *IEEE Symposium on Security and Privacy*, pages 908–925. IEEE Computer Society, 2018.
- [ZGK+22] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. IACR Cryptol. ePrint Arch., page 1565, 2022.
- [zkS] zksync. https://v2-docs.zksync.io/dev/.
- [zkW] zkwasm. https://github.com/DelphinusLab/zkWasm.

$$\mathsf{R}_{\mathsf{zero}} = \left\{ \begin{array}{c} \left( \bot, (\mathbb{K}, \mathbb{G}, \mathbb{F}, [\llbracket f_i \rrbracket]_i^k, G \in \mathbb{F}[X_1, \dots, X_j], [v_i]_i^j \in \mathbb{F}[X], \phi \in [j] \to [k]), ([f_i]_i^k) \right) : \\ F(X) \leftarrow G(X, f_{\phi(1)}(v_1(X)), \dots, f_{\phi(j)}(v_j(X))) \in \mathbb{F}[X]^{\leq B} \\ \forall x \in \mathbb{K}, \ F(x) = 0 \end{array} \right\}$$

 $\mathsf{ZeroTest.P}(\bot, (\mathbb{K}, \mathbb{G}, [\llbracket f_i \rrbracket]_i^k, G, [v_i]_i^j), ([f_i]_i^k)) \leftrightarrow \mathsf{ZeroTest.V}(\bot, (\mathbb{K}, \mathbb{G}, [\llbracket f_i \rrbracket]_i^k, G, [v_i]_i^j))$ 

(1) P computes and sends the oracle of quotient polynomial  $q(X) = \frac{F(X)}{Z_{K}(X)}$ .

(2) Verifier samples a random point  $\beta \leftarrow \mathbb{F} \setminus \mathbb{G}$  and queries oracles to check:  $q(\beta)Z_{\mathbb{K}}(\beta) \stackrel{?}{=} F(\beta)$ .

Figure 12: PolyIOP for testing whether a polynomial F(X) evaluates to zero over the domain  $\mathbb{K}$ .

 $\begin{aligned} & \frac{\mathsf{R}_{\mathsf{sum}} = \left\{ \bot, (\mathbb{K} = \langle \omega \rangle, \mathbb{G}, \llbracket f \rrbracket, H), f : \sum_{x \in \mathbb{K}} f(x) = H \right\} \\ & \underline{\mathsf{SumCheck.P}(\bot, (\mathbb{K}, \mathbb{G}, \llbracket f \rrbracket, H), f) \leftrightarrow \mathsf{SumCheck.V}(\bot, (\mathbb{K}, \mathbb{G}, \llbracket f \rrbracket, H))} \\ & (1) \ \mathsf{P} \ \text{interpolates and sends polynomial } T \ \text{over } \mathbb{K} \ \text{with evaluations set as partial sums:} \\ & T(1) = 0 \qquad \left[ T(\omega^i) = \sum_{j=0}^{i-1} \left( f(\omega^j) - \frac{H}{|\mathbb{K}|} \right) \right]_{i=1}^{|\mathbb{K}| - 1} \\ & (2) \ \mathsf{P} \ \text{and } \mathsf{V} \ \text{engage in } \mathsf{ZeroTest}(\mathbb{K}, \mathbb{G}) \ \text{to prove } L_{1,\mathbb{K}}(X)T(X) = 0 \ \text{over } \mathbb{K}. \\ & (3) \ \mathsf{P} \ \text{and } \mathsf{V} \ \text{engage in } \mathsf{ZeroTest}(\mathbb{K}, \mathbb{G}) \ \text{to prove } T(\omega X) - \left( T(X) + f(X) - \frac{H}{|\mathbb{K}|} \right) = 0 \ \text{over } \mathbb{K}. \end{aligned}$ 



# A Useful PolyIOPs

We present a number of useful polyIOPs that are used as building blocks in our main constructions. In this appendix, we present univariate versions of these polyIOPs as we are not aware of existing work that provides a unified presentation. We also will make use of multivariate versions of these polyIOPs for which we refer the reader to the unified presentation of [CBBZ23, Section 3].

<u>Advantage notation</u>. We use the following shorthand to specify parameters of the following subprotocols when describing advantages. We write  $\operatorname{Adv}_{\operatorname{ZeroTest}(\mathbb{K},\mathbb{G},d),\operatorname{R}_{\operatorname{Zero}},\mathsf{X}_{\operatorname{zt}},n,\mathcal{A}}(\lambda)$  for  $\operatorname{ZeroTest}(\mathbb{K},\mathbb{G})$  with degree bound d on the oracle F(X) being tested as in Figure 12. We write  $\operatorname{Adv}_{\operatorname{SundCheck}(\mathbb{K},\mathbb{G},d),\operatorname{R}_{\operatorname{sum}},\mathsf{X}_{\operatorname{sum}},n,\mathcal{A}}(\lambda)$  for  $\operatorname{SumCheck}(\mathbb{K},\mathbb{G})$  with degree bound d on the oracle f as in Figure 13. Note for the multivariate ZeroTest and SumCheck, only the admissible domain  $\mathbb{G}$  is related to soundness and so we drop  $\mathbb{K}$ . We write  $\operatorname{Adv}_{\operatorname{ProductCheck}(\mathbb{K},\mathbb{G},d),\operatorname{R}_{\operatorname{prod}},\mathsf{X}_{\operatorname{prod}},n,\mathcal{A}}(\lambda)$  for  $\operatorname{ProductCheck}(\mathbb{K},\mathbb{G})$  with degree bound d on the oracle f as in Figure 14.

**Zero test.** We provide a polyIOP for the relation  $R_{zero}$  that we refer to as ZeroTest in Figure 12 following [CHM<sup>+</sup>20] and [GWC19]. The protocol allows for testing whether a polynomial F(X) evaluates to zero over a domain K. For generality, F(X) is defined with respect to polynomial oracles  $[\llbracket f_i \rrbracket]_i^k$  that a verifier may have access to. To ensure domain admissibility, we also specify a restricted domain G from which the verifier will not sample evaluation challenges from. ZeroTest satisfies completeness and has soundness advantage at most  $\frac{B}{|\mathbb{F} \setminus \mathbb{G}|}$  where B is the degree bound of F(X) [CHM<sup>+</sup>20]. For simplicity, we will use the following shorthand for applying the zero test when the rest of the parameters are clear from context: ZeroTest(K,G), specifying the test will be evaluated over domain K, with the randomness sampled outside of the set G.

Sum check. We provide a polyIOP to prove a polynomial f(X) sums to H over an evaluation domain  $\mathbb{K}$ . We refer to the protocol as SumCheck defined in Figure 13. The protocol satisfies completeness and has soundness advantage at most  $\frac{2B}{|\mathbb{F} \setminus \mathbb{G}|}$  where B is the degree bound of f [GWC19]. We remark that [CHM<sup>+</sup>20] provides an alternative univariate sum-check protocol. The version that we use incurs an additional query but reduces to the univariate zero-test and is thus domain admissable making it convenient for use with polyIOP compilers. Again, when clear, we will use SumCheck( $\mathbb{K}, \mathbb{G}$ ) to denote the test will be evaluated over  $\mathbb{K}$  and the randomness will be sampled outside  $\mathbb{G}$ .

 $\mathsf{R}_{\mathsf{prod}} = \left\{ \bot, (\mathbb{K} = \langle \omega \rangle, \mathbb{G}, \llbracket f \rrbracket), f : \prod_{x \in \mathbb{K}} f(x) = 1 \right\}$ 

 $\mathsf{ProductCheck}.\mathsf{P}(\bot,(\mathbb{K},\mathbb{G},\llbracket f \rrbracket),f) \leftrightarrow \mathsf{ProductCheck}.\mathsf{V}(\bot,(\mathbb{K},\mathbb{G},\llbracket f \rrbracket))$ 

(1) P interpolates and sends polynomial T over  $\mathbb K$  with evaluations set as partial products:

$$T(1) = 1 \qquad \left[ T(\omega^i) = \prod_{j=0}^{i-1} \left( f(\omega^j) \right) \right]_{i=1}^{|\mathbb{K}|-1}$$

(2) P and V engage in ZeroTest( $\mathbb{K}, \mathbb{G}$ ) to prove  $L_{1,\mathbb{K}}(X)(T(X)-1) = 0$  over  $\mathbb{K}$ .

(3) P and V engage in ZeroTest( $\mathbb{K}$ ,  $\mathbb{G}$ ) to prove  $T(\omega X) - (T(X)f(X)) = 0$  over  $\mathbb{K}$ .

Figure 14: PolyIOP for checking that the product of evaluations of f(X) over domain  $\mathbb{K}$  is equal to 1.

$$\begin{split} & \mathsf{P}_{\mathsf{multiset}} = \left\{ \begin{array}{l} \bot, (\mathbb{K} = \langle \omega \rangle, \mathbb{G}, [[\llbracket f_{i,j} \rrbracket, \llbracket g_{i,j} \rrbracket]_{j \in [m]}]_{i \in [n]}), [[f_{i,j}, g_{i,j}]_{j \in [m]}]_{i \in [n]} \\ \vdots \bigcup_{i \in [n]} \{ [f_{i,j}(x)]_{j \in [m]} \} _{x \in \mathbb{K}} = \bigcup_{i \in [n]} \{ [g_{i,j}(x)]_{j \in [m]} \} _{x \in \mathbb{K}} \right\} \\ & \\ \hline \mathsf{MultisetCheck.P}(\bot, (\mathbb{K}, \mathbb{G}, [[\llbracket f_{i,j} \rrbracket, \llbracket g_{i,j} \rrbracket]_{j \in [m]}]_{i \in [n]}), [[f_{i,j}, g_{i,j}]_{j \in [m]}]_{i \in [n]}) \\ & \leftrightarrow \mathsf{MultisetCheck.V}(\bot, (\mathbb{K}, \mathbb{G}, [[\llbracket f_{i,j} \rrbracket, \llbracket g_{i,j} \rrbracket]_{j \in [m]}]_{i \in [n]}), [[f_{i,j}, g_{i,j}]_{j \in [m]}]_{i \in [n]}) \\ & \leftrightarrow \mathsf{MultisetCheck.V}(\bot, (\mathbb{K}, \mathbb{G}, [[\llbracket f_{i,j} \rrbracket, \llbracket g_{i,j} \rrbracket]_{j \in [m]}]_{i \in [n]}), [[f_{i,j}, g_{i,j}]_{j \in [m]}]_{i \in [n]}) \\ & \leftrightarrow \mathsf{MultisetCheck.V}(\bot, (\mathbb{K}, \mathbb{G}, [[\llbracket f_{i,j} \rrbracket, \llbracket g_{i,j} \rrbracket]_{j \in [m]}]_{i \in [n]})) \\ \hline (1) \mathsf{ V sends random challenges } \alpha, \beta \leftrightarrow (\mathbb{F} \setminus \mathbb{G})^2 \\ (2) \mathsf{ P and V both derive} \\ & [f_i(X) = \sum_{j \in [m]} \beta^j f_{i,j}(X)]_{i \in [n]} \quad [g_i(X) = \sum_{j \in [m]} \beta^j g_{i,j}(X)]_{i \in [n]} \\ \hline (3) \mathsf{ P sends oracle of polynomial $T$ over $\mathbb{K}$ with evaluations:} \\ & \left[ T(x) = \frac{\prod_{i \in [n]} (\alpha + f_i(x))}{\prod_{i \in [n]} (\alpha + g_i(x))} \right]_{x \in \mathbb{K}} . \\ \hline (4) \mathsf{ P and V engage in ProductCheck(\mathbb{K}, \mathbb{G}) to prove: \\ & \prod_{x \in \mathbb{K}} T(x) = 1 . \\ \hline (5) \mathsf{ P and V engage in ZeroTest(\mathbb{K}, \mathbb{G}) to prove: \\ & T(X) \prod_{i \in [n]} (\alpha + g_i(X)) = \prod_{i \in [n]} (\alpha + f_i(X)) \text{ over $\mathbb{K}$}. \\ \hline \end{cases} \end{split}$$



**Product check.** We also provide a polyIOP for proving the product of evaluations of a polynomial f(X) over domain  $\mathbb{K}$  equals one. We refer to the protocol as ProductCheck defined in Figure 14. The protocol satisfies completeness and has soundness advantage at most  $\frac{3|\mathbb{K}|+B}{|\mathbb{F}\backslash\mathbb{G}|}$  where B is the degree bound of f [GWC19].

**Multiset equality check.** Next, we provide a polyIOP for proving the multisets of evaluations of polynomials over domain  $\mathbb{K}$  are equal. We refer to the protocol as MultisetCheck defined in Figure 15. The protocol satisfies completeness and has soundness advantage at most  $\frac{m+5|\mathbb{K}|+nB}{|\mathbb{F}|-|\mathbb{G}|}$  where B is the max degree bound of  $f_{i,j}$ 's and  $g_{i,j}$ 's [GWC19].

**Cross-group product check.** Lastly, our Plookup-based vector lookup argument will require performing a product check across different domains. We provide a polyIOP for checking that the product of evaluations of a polynomial  $f_0(X)$  over a domain  $\mathbb{H}_0$  is equal to the product of evaluations of a polynomial  $f_1(X)$  over a different domain  $\mathbb{H}_1$ . We refer to the protocol as XGProductCheck defined in Figure 16.

**Theorem 10.** XGProductCheck for  $R_{prod}$  satisfies perfect completeness and for any adversary A against knowledge soundness, we provide an extractor X such that

$$\mathsf{Adv}_{\mathsf{XGProductCheck},\mathsf{R}_{\mathsf{xgprod}},\mathsf{X},2,\mathcal{A}}(\lambda) \leq \frac{4|\mathbb{H}_0| + 3|\mathbb{H}_1| + B_0 + B_1 + \max(|\mathbb{H}_0|,|\mathbb{H}_1|) + 2}{|\mathbb{F} \setminus (\mathbb{H}_0 \cup \mathbb{H}_1)|}$$

$$\begin{aligned} & \left[ \begin{array}{c} \mathsf{R}_{\mathsf{xgprod}} = \left\{ \bot, (\mathbb{H}_{0} = \langle \psi \rangle, \mathbb{H}_{1} = \langle \mu \rangle, \llbracket f_{0} \rrbracket, \llbracket f_{1} \rrbracket), (f_{0}, f_{1}) : \prod_{x \in \mathbb{H}_{0}} f_{0}(x) = \prod_{x \in \mathbb{H}_{1}} f_{1}(x) \right\} \\ & \left[ \begin{array}{c} \mathsf{XGProductCheck.P} \left( \begin{array}{c} \bot, \\ (\mathbb{H}_{0}, \mathbb{H}_{1}, \llbracket f_{0} \rrbracket, \llbracket f_{1} \rrbracket), \\ (f_{0}, f_{1}) \end{array} \right) \leftrightarrow \mathsf{XGProductCheck.V} \left( \begin{array}{c} \bot, \\ (\mathbb{H}_{0}, \mathbb{H}_{1}, \llbracket f_{0} \rrbracket, \llbracket f_{1} \rrbracket) \end{array} \right) \\ \hline (1) \text{ P computes and sends product polynomials } T_{0}, T_{1} \text{ and proves that they are well-formed.} \\ (a) \text{ P interpolates and sends polynomials } T_{0} \text{ over } \mathbb{H}_{0} \text{ and } T_{1} \text{ over } \mathbb{H}_{1} \text{ such that:} \\ & \left[ T_{0}(\psi^{i}) = \prod_{k=i}^{|\mathbb{H}_{0}|^{-1}} f_{0}(\psi^{k}) \right]_{i=0}^{|\mathbb{H}_{0}|^{-1}} \left[ T_{1}(\mu^{i}) = \prod_{k=i}^{|\mathbb{H}_{1}|^{-1}} f_{1}(\mu^{k}) \right]_{i=0}^{|\mathbb{H}_{1}|^{-1}} \\ \hline (b) \text{ P and V engage in ZeroTest}(\mathbb{H}_{0}, \mathbb{H}_{0} \cup \mathbb{H}_{1}) \text{ to prove } (X - \psi^{|\mathbb{H}_{0}|^{-1}})(T_{0}(\psi X) f_{0}(X) - T_{0}(X)) = 0 \text{ over } \mathbb{H}_{0}. \\ \hline (c) \text{ P and V engage in ZeroTest}(\mathbb{H}_{1}, \mathbb{H}_{0} \cup \mathbb{H}_{1}) \text{ to prove } (X - \mu^{|\mathbb{H}_{1}|^{-1}}, \mathbb{K}_{1}(X)(T_{0}(X) - T_{1}(X)) = 0 \text{ over } \mathbb{H}_{1}. \\ \hline (d) \text{ P and V engage in ZeroTest}(\mathbb{H}_{0}, \mathbb{H}_{0} \cup \mathbb{H}_{1}) \text{ to prove } L_{\mu^{|\mathbb{H}_{1}|^{-1}, \mathbb{H}_{0}}(X)(T_{0}(X) - f_{0}(X)) = 0 \text{ over } \mathbb{H}_{0}. \\ \hline (e) \text{ P and V engage in ZeroTest}(\mathbb{H}_{1}, \mathbb{H}_{0} \cup \mathbb{H}_{1}) \text{ to prove } L_{\mu^{|\mathbb{H}_{1}|^{-1}, \mathbb{H}_{1}}(X)(T_{1}(X) - f_{1}(X)) = 0 \text{ over } \mathbb{H}_{1}. \\ \hline (2) \text{ P and V engage in ZeroTest}(\mathbb{H}_{0}, \mathbb{H}_{0} \cup \mathbb{H}_{1}) \text{ to prove } L_{\mu^{|\mathbb{H}_{1}|^{-1}, \mathbb{H}_{1}}(X)(T_{0}(X) - T_{1}(X)) = 0 \text{ over } \mathbb{H}_{1}. \\ \hline (2) \text{ P and V engage in ZeroTest}(\mathbb{H}_{0}, \mathbb{H}_{0} \cup \mathbb{H}_{1}) \text{ to prove } L_{1,\mathbb{H}_{0}}(X)(T_{0}(X) - T_{1}(X)) = 0 \text{ over } \mathbb{H}_{1}. \\ \hline \end{array} \right\}$$

Figure 16: XGProductCheck: Protocol for checking that the product of  $f_0(X)$  over group  $\mathbb{H}_0$  is equal to the product of  $f_1(X)$  over group  $\mathbb{H}_1$ 

where  $B_0$  and  $B_1$  are the degree bounds of  $f_0$  and  $f_1$ , respectively.

Proof. We provide arguments for completeness and soundness separately.

<u>Completeness.</u> By inspection, from the construction of  $T_0$  and  $T_1$  and from the completeness of the zero-test, the zero-tests in steps (1b), (1c), (1d), and (1e) will succeed. If the two products are indeed equal, then again, by construction of  $T_0$  and  $T_1$  and the completeness of the zero-test, the zero-test in step (2) will succeed.

<u>Knowledge soundness.</u> By the soundness of the zero-test in steps (1b) and (1d), we have that  $T_0(\psi^{|\mathbb{H}_0|-1}) = f_0(\psi^{|\mathbb{H}_0|-1})$ and that  $T_0(x) = T_0(\psi x) f_0(x)$  for all  $x \in \mathbb{H}_0 \setminus \{\psi^{|\mathbb{H}_0|-1}\}$ . By induction, this means that for all  $i \in [|\mathbb{H}_0|]$ ,  $T_0(\psi^i) = \prod_{\substack{k=0 \ k=0}}^{|\mathbb{H}_0|-1} f_0(\psi^k)$ , and in particular,  $T_0(1) = \prod_{\substack{k=0 \ k=0}}^{|\mathbb{H}_0|-1} f_0(\psi^k)$ . Respectively, by (1c) and (1e), we have that  $T_1(1) = \prod_{\substack{k=0 \ k=0}}^{|\mathbb{H}_0|-1} f_1(\mu^k)$ . Finally, by the soundness of the zero-test in step (2), we have that  $T_0(1) = T_1(1)$ . We define X to simply employ X<sub>zt</sub> to retrieve  $f_0, f_1$ . By the soundness bounds of each of the zero-tests, we complete the argument.

# **B** Generic Zero-Knowledge Compiler for Univariate PolyIOPs

Here, we introduce a generic zero-knowledge compiler for a class of polyIOPs that satisfy domain-admissibility and are of a form where no polynomial evaluation queries are made outside of zero tests. This is a common structure for polyIOP design, and in this case, we show how to construct a resulting polyIOP that provides perfect honest verifier zero knowledge. We give a definition for polyIOPs that satisfy this zero test requirement, referring to them as ZT-Based polyIOPs in Figure 17. At a high level, the template for ZT-Based polyIOPs follows two stages. In the first stage, the prover and verifier engage in  $n_1$  rounds where in each round the prover sends a polynomial oracle and the verifier responds with a random challenge. In the second stage, the prover and verifier engage in  $n_2$  zero tests over the polynomial oracles provided to the verifier.

Given a ZeroTest-based, domain-admissible polyIOP, our compiler in Figure 18 compiles into and outputs a zeroknowledge ZeroTest-based polyIOP. In the first stage, the new prover and verifier are unchanged, sending oracles and responding with challenges as in the original polyIOP. In the second stage, the new prover and verifier engage in a zero-knowledge zero test, ZK-ZeroTest [BNO21] given in Figure 18, instead of the standard zero test ZeroTest.

**Theorem 11.** Given a ZT-Based polyIOP (Figure 17) with completeness error  $\epsilon_c$  and knowledge soundness error as a function of  $\operatorname{Adv}_{\operatorname{ZeroTest}(\mathbb{H},\mathbb{G},d),\operatorname{R}_{\operatorname{Zero}},\operatorname{X}_{\operatorname{zt}},\mathcal{A}}(\lambda)$  for relation R, our zero-knowledge compiler (Figure 19) out-

 $\mathsf{ZT}\text{-}\mathsf{Based}.\mathsf{P}(([f_i]_{i \in [k_1]}), (\mathbb{F}, \llbracket f_i \rrbracket]_{i \in [k_1, k_1 + k_2]}), ([f_i]_{i \in [k_1, k_1 + k_2]})) \leftrightarrow \mathsf{ZT}\text{-}\mathsf{Based}.\mathsf{V}(([\llbracket f_i \rrbracket]_{i \in [k_1]}), (\mathbb{F}, \llbracket f_i \rrbracket]_{i \in [k_1, k_1 + k_2]})) \mapsto \mathsf{ZT}\text{-}\mathsf{Based}.\mathsf{V}((\llbracket f_i \rrbracket]_{i \in [k_1]}), (\mathbb{F}, \llbracket f_i \rrbracket]_{i \in [k_1, k_1 + k_2]})) \mapsto \mathsf{ZT}\text{-}\mathsf{Based}.\mathsf{V}((\llbracket f_i \rrbracket]_{i \in [k_1]}), (\mathbb{F}, \llbracket f_i \rrbracket]_{i \in [k_1, k_1 + k_2]})) \mapsto \mathsf{ZT}\text{-}\mathsf{ZT}\text{$ (1) P and V engage in  $n_1$  rounds where, in each round, P sends a polynomial oracle and V replies with a random verifier challenge. For each round  $i \in [n_1]$ :

- (a) P sends the oracle of a polynomial  $p_i$ . The oracle may also be a virtual oracle consisting of a linear combination of previous oracles  $[f_j]_{j \in [k_1+k_2]}$  and  $[p_j]_{j \in [i-1]}$ . (b) V sends random challenge  $r_i \leftarrow \mathbb{F}$ .
- (2) P and V engage in  $n_2$  rounds of zero tests. For each round  $i \in [n_2]$ :
  - (a) P and V specify parameters for a zero test: domains  $\mathbb{K}_i$ ,  $\mathbb{G}_i$ , polynomial  $G_i \in \mathbb{F}[X_{\lfloor k_1 + k_2 + n_1 + 1 \rfloor}]$ , rotations  $[v_{i,j}]_{j \in [\ell]}$ , and map  $\phi_i \in [\ell] \to [k_1 + k_2 + n_1].$
  - (b) P and V engage in ZeroTest( $\mathbb{K}_i, \mathbb{G}_i, \mathbb{F}, [\llbracket f_i \rrbracket]_{i \in [k_1+k_2]} \| [\llbracket p_i \rrbracket]_{i \in [n_1]}, G_i, [v_{i,j}]_{j \in [\ell]}, \phi_i$ ).

Figure 17: Template for polyIOPs that are ZeroTest-based. All polynomial evaluation queries made by the verifier are restricted to zero tests.



Figure 18: A ZK-PolyIOP for testing whether a polynomial F(X) evaluates to zero over the domain K.

puts a ZKZT-Based polyIOP with completeness error  $\epsilon_c$  and knowledge soundness error as a same function of  $\operatorname{\mathsf{Adv}}_{\mathsf{ZK-ZeroTest}(\mathbb{H},\mathbb{G},d),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda)$  for the same relation R.

Proof. The first stage remains unchanged. The second stage replaces ZeroTest with ZK-ZeroTest, where the completeness and knowledge soundness of ZK-ZeroTest are proven in [BNO21]. Thus, since both ZeroTest and ZK-ZeroTest achieve perfect completeness, the completeness is unchanged. The knowledge soundness advantage swaps advantage against ZeroTest with advantage against ZK-ZeroTest in the exact same function.  $\square$ 

Theorem 12. Given a domain-admissible ZT-Based polyIOP (Figure 17) for relation R, our zero-knowledge compiler (Figure 19) outputs a ZKZT-Based polyIOP that is honest-verifier zero-knowledge for the same relation R.

*Proof.* We provide the simulator in Figure 20. In the first stage, where the honest verifier receives oracle polynomials and sends uniformly distributed randomness, the simulator generates randomness honestly. In the second stage, our

| Z  | $(K-Compiler(ZT-Based.P \leftrightarrow ZT-Based.V) \rightarrow (ZKZT-Based.P \leftrightarrow ZKZT-Based.V)$  |
|----|---|
| C  | Construct and output ZKZT-Based.P and ZKZT-Based.V as follows:  |
| (1 | ZKZT-Based mimics ZT-Based in the first $n_1$ rounds of sending polynomial oracles and producing verifier challenges.<br>For each round $i \in [n_1]$ of ZT-Based:  |
|    | <ul> <li>(a) P sends oracle of polynomial p<sub>i</sub> from ZT-Based.</li> <li>(b) V sends random challenge r<sub>i</sub> ← ℝ.</li> </ul>  |
| (2 | 2) In the final $n_2$ rounds of zero tests, ZKZT-Based replaces each zero test of ZT-Based with a zero-knowledge zero test.<br>For each round $i \in [n_2]$ of ZT-Based:  |
|    | (a) P and V specify parameters for a zero test: domains $\mathbb{K}_i$ , $\mathbb{G}_i$ , polynomial $G_i \in \mathbb{F}[X_{[k_1+k_2+n_1+1]}]$ , rotations $[v_{i,j}]_{j \in [\ell]}$ , and map $\phi_i \in [\ell] \to [k_1+k_2+n_1]$ . |
|    | (b) P and V engage in ZK-ZeroTest( $\mathbb{K}_i, \mathbb{G}_i, \mathbb{F}, [\llbracket f_i \rrbracket]_{i \in [k_1+k_2]} \parallel [\llbracket p_i \rrbracket]_{i \in [n_1]}, G_i, [v_{i,j}]_{j \in [\ell]}, \phi_i$ ).                |
|    |   |

Figure 19: A ZK compiler for ZeroTest-based polyIOPs.

| $\underline{S(\mathbb{F},i,x)}$   |
|---|
| (1) For $i \in [n_1]$ , uniformly sample honest verifier randomness $r_i \leftarrow \mathbb{F}$ .     |
| (2) For $i \in [n_2]$ , run $vw_i \leftarrow S_{ZK-ZeroTest}$ from [BNO21] to simulate the zero test. |
| (3) Output $r_1 \  \dots \  r_{n_1} \  vw_1 \  \dots \  vw_{n_2}$ .                                   |

#### Figure 20: Simulator for ZKZT-Based polyIOPs

simulator runs the ZK-ZeroTest simulator defined in [BNO21]. Since the zero tests are domain-admissible, that is, the zero test query falls outside the domain over which the polynomial encodes witness elements, we can invoke the honest-verifier zero-knowledge of ZK-ZeroTest proven in [BNO21]. Thus, the verifier view in the second phase is indistinguishiable from simulator output.  $\Box$ 

#### C Multivariate Succinct Vector Lookup

Recall the multivariate polynomial encoding for vectors from Section 2.2. Given a table of vectors  $[[t_{i,j}]_{j \in [m]}]_{i \in [d_1]}$ and a list of claimed looked up vectors  $[[f_{i,j}]_{j \in [m]}]_{i \in [d_0]}$ , we consider polynomial encodings  $f \in \mathbb{F}[X_{[\log m + \log d_0]}]$ and  $t \in \mathbb{F}[X_{[\log m + \log d_1]}]$  as follows. The vectors are encoded as the evaluations of  $\log m$ -dimension subcubes of the boolean hypercube evaluation domain:

$$\begin{split} & \left[ \left[ f(j,i) = f_{\mathsf{int}_{\log d_0}(i),\mathsf{int}_{\log m}(j)} \right]_{j \in \{0,1\}^{\log m}} \right]_{i \in \{0,1\}^{\log d_0}} \\ & \left[ \left[ t(j,i) = t_{\mathsf{int}_{\log d_1}(i),\mathsf{int}_{\log m}(j)} \right]_{j \in \{0,1\}^{\log m}} \right]_{i \in \{0,1\}^{\log d_1}}. \end{split}$$

Then the vector lookup relation is as follows:

$$\mathsf{R}_{\mathsf{vlkup}} = \begin{array}{l} \left\{ \bot, (\llbracket f \rrbracket, \llbracket t \rrbracket), (f, t) : \left\{ [f(j, i)]_{j \in \{0, 1\}^{\log m}} \right\}_{i \in \{0, 1\}^{\log d_{1}}} \\ & \subseteq \left\{ [t(j, i)]_{j \in \{0, 1\}^{\log m}} \right\}_{i \in \{0, 1\}^{\log d_{1}}} \end{array} \right\}$$

For a polynomial encoding  $f \in \mathbb{F}[X_{\lfloor \log m + \log d_0 \rfloor}]$  and for any  $d_1 \ge d_0$ , we say f is padded with dummy variables if for any input  $(a_1, \ldots, a_{\log m + \log d_1}) \in \mathbb{F}^{\log m + \log d_0}$ , we ignore the extra  $\log d_1 - \log d_0$  elements and only compute  $f(a_1, \ldots, a_{\log m + \log d_0})$ .

The full construction is provided in Figure 21.

**Security.** We prove the completeness, knowledge soundness, and zero knowledge of SubcubeLkup in the following two theorems. The zero-knowledge of SubcubeLkup is achieved through the zero-knowledge compiler of Chen et al. [CBBZ23] by observing that SubcubeLkup reduces down to multivariate sum check polynomial identities.

**Theorem 13.** SubcubeLkup for vlkup (Figure 21) has completeness error  $\frac{d_0m+d_1m}{|\mathbb{F}|-md_0-md_1}$  and for any adversary  $\mathcal{A}$  against knowledge soundness, we provide an extractor X using X<sub>zt</sub>, an extractor for ZeroTest, and using X<sub>sum</sub>, an

$$\mathsf{R}_{\mathsf{vlkup}} = \left\{ \left( \bot, (\llbracket f \rrbracket, \llbracket t \rrbracket), (f, t) \right) : \left\{ [f(j, i)]_{j \in \{0, 1\}^{\log m}} \right\}_{i \in \{0, 1\}^{\log d_0}} \subseteq \left\{ [t(j, i)]_{j \in \{0, 1\}^{\log m}} \right\}_{i \in \{0, 1\}^{\log d_1}} \right\}$$

 $\mathsf{SubcubeLkup}.\mathsf{P}(\bot,(\llbracket f \rrbracket,\llbracket t \rrbracket),(f,t)) \leftrightarrow \mathsf{SubcubeLkup}.\mathsf{V}(\bot,(\llbracket f \rrbracket,\llbracket t \rrbracket))$ 

(1) P computes, sends, and proves the wellformedness of the count polynomial  $\tilde{c}$  that encodes the counts  $[c_i]_{i \in [d_1]}$  where  $c_i$  is the number of times the  $i^{th}$  subcube appears in f.

(a) P sends  $\widetilde{c} \in \mathbb{F}[X_{\lfloor \log(md_1) \rfloor}]$  as the MLE with constant evaluation in each subcube:  $\left[ [\widetilde{c}(j,i) = c_i]_{i \in \{0,1\}^{\log d_1}} \right]_{j \in \{0,1\}^{\log m}}$ .

- (b) P and V engage in ZeroTest( $\{0,1\}^{\log(md_1)}$ ) to prove every subcube is constant:  $\widetilde{shft}_{\log m}(\tilde{c})(X) = \tilde{c}(X)$  over  $\{0,1\}^{\log(md_1)}$ .
- (2) V sends random challenges  $(\alpha, \beta) \in (\mathbb{F} \setminus \{0, 1\}^{\log(m \cdot \max(d_0, d_1))})^2$ .
- (3) P computes and sends the position-indexing powers-of- $\beta$  polynomial  $\widetilde{I}_b(X)$  for  $b \in \{0,1\}$  and proves its well-formedness:
  - (a) P sends  $\widetilde{I}_b \in \mathbb{F}[X_{\lfloor \log m + \log d_b \rfloor}]$  as the MLE with evaluations over subcubes  $j \in \{0,1\}^{\log m}$  set to the corresponding power-of- $\beta$ ,  $\beta^{\operatorname{int}_{\log m}(j)}$ :

$$\left[\left[\widetilde{I_b}(j,i)=\beta^{\mathsf{int}_{\log m}(j)}\right]_{i\in\{0,1\}^{\log d_b}}\right]_{j\in\{0,1\}^{\log d_b}}$$

- (b) P and V engage in ZeroTest( $\{0,1\}^{\log(md_b)}$ ) to prove  $(\widetilde{I}_b(X_{\lceil \log(md_b) \rceil}) 1)(\widetilde{eq}_{\log m}(X_{\lceil \log m \rceil}, 0_{\lceil \log m \rceil})) = 0$  over  $\{0,1\}^{\log(md_b)}$ .
- (c) P and V engage in  $\text{ZeroTest}(\{0,1\}^{\log(md_b)})$  to prove:

$$\widetilde{shft}_{\log m}(\widetilde{I_b})(X_{\lfloor \log(md_b) \rfloor}) - \beta \cdot \widetilde{I_b}(X_{\lfloor \log(md_b) \rfloor}) \Big) \cdot (1 - \widetilde{eq}_{\log m}(X_{\lfloor \log m \rfloor}, 1_{\lfloor \log m \rfloor})) = 0 \text{ over } \{0, 1\}^{\log(md_b)}.$$

- (4) P computes and sends the summation polynomial  $\widetilde{S}_b(X)$  for  $b \in \{0,1\}$  and proves its well-formedness:
  - (a) P sends  $\widetilde{S_b} \in \mathbb{F}[X_{\lfloor \log(md_b) \rfloor}]$  as the MLE setting the evaluation to be constant in each subcube to summation of the values in the subcube  $p_b$  multiplied by powers of  $\beta$ . Let  $p_0 = f$  and  $p_1 = t$ :

$$\left[\left[\widetilde{S_b}(j,i) = \sum_{k \in \{0,1\}^{\log m}} \beta^{\mathsf{int}_{\log m}(k)} \cdot p_b(k,i)\right]_{i \in \{0,1\}^{\log d_b}}\right]_{j \in \{0,1\}^{\log m}}$$

(b) P and V engage in ZeroTest( $\{0,1\}^{\log(md_b)}$ ) to prove every subcube is constant:  $\widetilde{shft}_{\log m}(\widetilde{S_b})(X) = \widetilde{S_b}(X)$  over  $\{0,1\}^{\log(md_b)}$ .

(5) P computes and sends the induction polynomial  $\widetilde{B}_b(X)$  for  $b \in \{0,1\}$  and prove its well-formedness. Let  $p_0 = f$  and  $p_1 = t$ :

(a) P sends  $\widetilde{B_b} \in \mathbb{F}[X_{\lceil \log(md_b) \rceil}]$  as the MLE with evaluations over the hypercube that accumulate the normalized summation:

$$\left\lfloor \left[ \widetilde{B_b}(j,i) = \sum_{k \in [\mathsf{int}_{\log m}(j)]} \left( \beta^k \cdot p_b(\mathsf{bin}_{\log m}(k),i) - \frac{\widetilde{S_b}(j,i)}{m} \right) \right]_{i \in \{0,1\}^{\log d_b}} \right\rfloor_{j \in \{0,1\}^{\log m}}$$

(b) P and V engage in ZeroTest( $\{0,1\}^{\log(md_b)}$ ) to prove induction:

$$\widetilde{shft}_{\log m}(\widetilde{B_b})(X) = \widetilde{B_b}(X) + \widetilde{shft}_{\log m}(\widetilde{I_b})(X) \cdot \widetilde{shft}_{\log m}(p_b)(X) - \frac{\widetilde{S_b}(X)}{m} \text{ over } \{0,1\}^{\log(md_b)}.$$

- (6) P computes and sends the inverse polynomial  $\widetilde{U_b}(X)$  for  $b \in \{0,1\}$  and proves its well-formedness:
  - (a) P sends  $\widetilde{U_b}$  as the MLE defined with evaluations set to the inverse of the summation and random challenge  $\alpha$  as appears in the denominator of the Haböck lemma. Let  $p_0 = f$  and  $p_1 = t$ :

$$\left[ \left[ \widetilde{U_b}(j,i) = \frac{1}{\alpha + \sum_{k \in \{0,1\}^{\log m}} \beta^{\mathsf{int}_{\log m}(k)} \cdot p_b(k,i)} \right]_{i \in \{0,1\}^{\log d_b}} \right]_{j \in \{0,1\}^{\log m}}$$

(b) P and V engage in ZeroTest( $\{0,1\}^{\log(md_b)}$ ) to prove inversion:  $\widetilde{U_b}(X) \cdot (\alpha + \widetilde{S_b}(X)) = 1$  over  $\{0,1\}^{\log(md_b)}$ .

(7) P proves the summations of U<sub>0</sub> and c̃ · U<sub>1</sub> over {0,1}<sup>log(md<sub>0</sub>)</sup> and {0,1}<sup>log(md<sub>1</sub>)</sup>, respectively, are equal. Assume multivariate inputs are padded appropriately to log(m · max(d<sub>0</sub>,d<sub>1</sub>)) with dummy variables. P and V engage in SumCheck({0,1}<sup>log(m·max(d<sub>0</sub>,d<sub>1</sub>))</sup>) to prove:

$$\widetilde{U_0}(X) - \widetilde{c}(X) \cdot \widetilde{U_1}(X) \cdot \frac{d_1}{d_0} \quad \text{sums to 0 over } \{0,1\}^{\log(m \cdot \max(d_0,d_1))}$$

Figure 21: Multivariate vector lookup argument in which vectors are encoded as evaluations of subcubes over the boolean hypercube.

extractor for SumCheck, such that

$$\begin{split} &\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{SubcubeLkup,vlkup},\mathsf{X},\mathcal{A}}(\lambda) \leq 7\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_1)},\log(md_1)+\log(m)),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ 5\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_0)},2\log(md_0)),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\mathrm{sound}}_{\mathsf{SumCheck}(\{0,1\}^{\log(m\max(d_0,d_1))},2\log(md_1)+\log(md_0)),\mathsf{R}_{\mathsf{sum}},\mathsf{X}_{\mathsf{sum}},\mathcal{A}}(\lambda) \\ &+ \frac{md_0 + md_1}{|\mathbb{F}| - md_0 - md_1}. \end{split}$$

*Proof.* <u>Completeness.</u> The correctness of checks about polynomials  $c, I_b, S_b, B_b, U_b$  follow directly from Theorem 3 except that we change to multivariate encodings and use <u>shfth</u> to traverse subcubes instead of cosets.

For the accumulated sum of  $U_0$  and  $c \cdot U_1$ , by paddings when we sum over  $\{0,1\}^{\log(m \cdot \max(d_0,d_1))}$ , the last  $\log d_0 - \log d_1$  bits are ignored. For inputs  $(a_1, \ldots, a_{\log(m \cdot \max(d_0,d_1))})$  with the same suffix  $(a_{\log(m \cdot d_1)+1}, \ldots, a_{\log(m \cdot \max(d_0,d_1))})$ , the same evaluations on  $c \cdot U_1$  sum for  $\frac{d_0}{d_1}$  times. Given that

$$\sum_{j \in \{0,1\}^{\log(m)}, i \in \{0,1\}^{\log(d_0)}} U_0(k,i) = \sum_{j \in \{0,1\}^{\log(m)}, i \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), i \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), i \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), i \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), i \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), i \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i), j \in \{0,1\}^{\log(d_1)} = \sum_{j \in \{0,1\}^{\log(d_1)} c(k,i), j \in \{0,1\}^{\log(d_1)} (k,i), j$$

if we sum the RHS over  $\{0,1\}^{\log(m \cdot \max(d_0,d_1))}$  and multiply by  $\frac{d_1}{d_0}$  to cancel out the extra summands, the equation should stay equal. The rest follows by the completeness of sum check. The completeness error also comes from the undefined fraction evaluations and can be made perfect complete using technique in Remark 4.

<u>Knowledge Soundness</u>. We bound the advantage of adversary  $\mathcal{A}$  by bounding the advantage of each of a series of game hops [BR06]. We define  $G_0 = SOUND_{SubcubeLkup,R_{vlkup},X}^{\mathcal{A}}(\lambda)$ . The inequality above follows from the following claims that we will justify:

(1)

$$\begin{split} \Pr[G_0 = 1] - \Pr[G_1 = 1] &| \leq 2\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_1)}, \log(md_1) + \log(m)), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ 2\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_0)}, \log(md_0) + \log(m)), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(2)

$$\begin{split} |\Pr[G_1 = 1] - \Pr[G_2 = 1]| &\leq \mathsf{Adv}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_0)}, \log(md_0)), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_1)}, \log(md_1)), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) + \mathsf{Adv}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_0)}, 2\log(md_0)), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_1)}, 2\log(md_1)), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(3)

$$\begin{split} |\Pr[\mathbf{G}_2 = 1] - \Pr[\mathbf{G}_3 = 1]| &\leq \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_0)}, 2\log(md_0)), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ 2\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(md_1)}, 2\log(md_1)), \mathsf{R}_{\mathsf{Zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(4)

$$|\Pr[G_3 = 1] - \Pr[G_4 = 1]| \leq \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{SumCheck}(\{0,1\}^{\log(m\max(d_0,d_1))}, 2\log(md_1) + \log(md_0)), \mathsf{R}_{\mathsf{sum}}, \mathsf{X}_{\mathsf{sum}}, \mathcal{A}}(\lambda)$$

(5)  $|\Pr[G_4 = 1] - \Pr[G_5 = 1]| \le \frac{md_0 + md_1}{|\mathbb{F}| - md_0 - md_1}$ (6)  $\Pr[G_5 = 1] = 0$  Our proof follows closely the univariate case in Theorem 3. The plan for the soundness proof is as follows: Claim 1 argues that polynomial  $I_b$ 's are constructed properly. Claim 2 argues that  $S_0$  and  $S_1$  encode hashes of each coset of f and t, respectively. Claim 3 argues that  $u_b$  encode the summand of the logarithmic derivatives evaluated at  $(\alpha, \beta)$ . Claim 4 argues that the logarithmic derivative evaluations are equal, claim 5 argues that if so then  $\left\{ [f(j,i)]_{j\in\{0,1\}^{\log m}} \right\}_{i\in\{0,1\}^{\log d_0}} \subseteq \left\{ [t(j,i)]_{j\in\{0,1\}^{\log d_0}} \right\}_{i\in\{0,1\}^{\log d_1}}$ . Lastly, Claim 6 argues that the constructed extractor always succeeds for an accepting verifier.

Claim 1: This step follows the same idea as Claim 1 in Theorem 3.  $\tilde{eq}_{\log m}$  serves as the Lagrange polynomial  $L_{1,\mathbb{V}}$  in the univariate case. In particular,  $\tilde{eq}_{\log m}(X_{\lceil \log m \rceil}, 0_{\lceil \log m \rceil})$  is 1 only when it is evaluated at the first element in the subcube, while  $\tilde{eq}_{\log m}(X_{\lceil \log m \rceil}, 1_{\lceil \log m \rceil})$  is one only when it is evaluated at the last element.  $\tilde{shft}_{\log m}$  traverse elements in the subcube of the first  $\log m$  bits to make sure the next element is  $\beta$  times the previous one within the same subcube generated by the trailing  $\log d_b$  bits.

 $I_f(X_{[\log(mn)]})$  is a  $(\log(mn))$ -variate polynomial encoding the *m* powers of  $\beta$  in the boolean subcubes of its leading  $\log m$  bits:

$$\left[ \left[ I_f(j,i) = \beta^{\mathsf{int}_{\log m}(j)} \right]_{j \in \{0,1\}^{\log m}} \right]_{i \in \{0,1\}^{\log m}}.$$

The following polynomial identities checked via multivariate zero tests [Set20, CBBZ23] over  $\{0,1\}^{\log(mn)}$  verify the wellformedness of  $I_f$ :

- $(I_b(X_{\lfloor \log(mn) \rfloor}) 1)(\widetilde{eq}_{\log m}(X_{\lfloor \log m \rfloor}, 0_{\lfloor \log m \rfloor})) = 0$  over  $\{0, 1\}^{\log(md_b)}$ : The first term checks that  $I_b$  is anchored to 1. The second term  $\widetilde{eq}_{\log m}(X_{\lfloor \log m \rfloor}, 0_{\lfloor \log m \rfloor})$  enforces this check only for the first position  $(0_{\lfloor \log m \rfloor})$  of each subcube. The polynomial  $\widetilde{eq}_{\log m}$  takes in two boolean vectors of length  $\log m$  and outputs 1 if they are equal and 0 otherwise.
- $\left(\widetilde{shft}_{\log m}(\widetilde{I}_b)(X_{\lfloor \log(md_b) \rfloor}) \beta \cdot \widetilde{I}_b(X_{\lfloor \log(md_b) \rfloor})\right) \cdot (1 \widetilde{eq}_{\log m}(X_{\lfloor \log m \rfloor}, 1_{\lfloor \log m \rfloor})) = 0$  over  $\{0, 1\}^{\log(md_b)}$ : The first term enforces the next element in the subcube (generated by  $\widetilde{shft}$ ) is equal to  $\beta$  times the previous element. The last term excludes the last element of the subcube  $(1_{\lfloor \log m \rfloor})$  preventing carry-over:  $\beta^{m-1} \cdot \beta \neq 1$ . As before, since the first element was anchored to 1 and the last element is excluded, this sets the evaluations of every subcube to be equal to the powers of  $\beta$ .

*Claim 2-3:* Claim 2-3 are the same as those in Theorem 3.  $shft_{\log m}$  enables traversing in subcube as the coset traversing in univariate case.

Claim 4: In this step, we deal with different hypercubes by paddings. Suppose WLOG,  $d_0 \ge d_1$ . Since  $c \cdot U_1 \in \mathbb{F}[X_{\lfloor \log m + \log d_1}]]$ , when we sum over  $\{0,1\}^{\log(m \cdot \max(d_0,d_1))}$ , the last  $\log d_0 - \log d_1$  bits are ignored and each one of the ignored bits has  $2^{\log d_0 - \log d_1} = \frac{d_0}{d_1}$  number of possible prefixes. Hence, for inputs that have suffix chopped off, the same output evaluated only on the prefix repeats for  $\frac{d_0}{d_1}$  times. In order to compute the sum of  $c \cdot U_1$  over  $\{0,1\}^{\log(m \cdot d_1)}$ , we hence multiply by  $\frac{d_1}{d_0}$ . The last check is therefore verifying

$$\sum_{i \in \{0,1\}^{\log(m)}, i \in \{0,1\}^{\log(d_0)}} U_0(k,i) = \sum_{j \in \{0,1\}^{\log(m)}, i \in \{0,1\}^{\log(d_1)}} c(k,i) \cdot U_1(k,i)$$

Expand the definition of  $U_b$  and c, it is equivalent to

$$\begin{split} &\sum_{j \in \{0,1\}^{\log(m)}, i \in \{0,1\}^{\log(d_0)}} \frac{1}{\alpha + \sum_{k \in \{0,1\}^{\log m}} \beta^{\mathsf{int}_{\log m}(k)} \cdot p_0(k,i)} \\ &= \sum_{k \in \{0,1\}^{\log(m)}, i \in \{0,1\}^{\log(d_1)}} \frac{c_i}{\alpha + \sum_{k \in \{0,1\}^{\log m}} \beta^{\mathsf{int}_{\log m}(k)} \cdot p_1(k,i)} \end{split}$$

Since j does not affect the sum, sum over j cancels out in both sides and becomes:

$$\sum_{\substack{i \in \{0,1\}^{\log(d_0)}}} \frac{1}{\alpha + \sum_{k \in \{0,1\}^{\log m}} \beta^{\operatorname{int}_{\log m}(k)} \cdot p_0(k,i)} = \sum_{\substack{i \in \{0,1\}^{\log(d_1)}}} \frac{c_i}{\alpha + \sum_{k \in \{0,1\}^{\log m}} \beta^{\operatorname{int}_{\log m}(k)} \cdot p_1(k,i)}$$

 $G_4$  employs the zero test extractor  $X_{sum}$  to check the above tests and aborts if the extractor fails.

*Claim 5-6:* Steps are same as those in Theorem 3, where we replace univariate evaluations of f and t to the multivariate ones.

**Theorem 14.** The compiled polyIOP using the compiler in [CBBZ23] of SubcubeLkup for vlkup (Figure 21) is honest-verifier zero-knowledge.

*Proof.* [CBBZ23] provides a multivariate ZeroTest protocol from SumCheck, and also provdes a generic ZK-compiler that transforms sumcheck-based polyIOPs into a zero knowledge one. Since verifier in SubcubeLkup never makes queries outside ZeroTest, the compiler can be applied to produce an HVZK protocol.

**Corollary 15.** Using random linear combinations, k-SubcubeLkup for  $R_{k-vlkup}$  is complete and knowledge sound with negligible errors.

# **D** Mux-HyperPLONK Construction

We begin by providing a description of the HyperPLONK indexer in Figure 22 as it is relevant to our use within Mux-HyperPLONK. It follows closely to that of PLONK encoding the selector and permutation index inputs as multivariate polynomials. The HyperPLONK [CBBZ23] multivariate polyIOP targets a slightly modified version of the PLONK arithmetization relation adapted for the multivariate polyIOP setting: We refer to the full HyperPLONK paper for further details on the proving construction [CBBZ23].

Define the following hypercube evaluation domains for the machine execution setting:

- Say instruction encodings are of size d and can therefore be encoded in a boolean subcube of length  $\log d$ .
- Say the public statement for an instruction is of size d<sub>x</sub> and is further divided into two components, input and output, each of size d<sub>x</sub>/2. For the hypercube {0,1}<sup>d<sub>x</sub></sup>, say the input portion is encoded on {X<sub>[log d<sub>x</sub>]</sub> ∈ {0,1}<sup>log d<sub>x</sub></sup> : X<sub>(log d<sub>x</sub>)-1</sub> = 0} and the output portion is encoded on {X<sub>[log d<sub>x</sub>]</sub> ∈ {0,1}<sup>log d<sub>x</sub></sup> : X<sub>(log d<sub>x</sub>)-1</sub> = 1}.
- Say the number of instructions in the instruction set is  $\ell$ .
- Say the number of unrolled execution steps is n.

With this notation, Figure 22 gives the setup algorithm for computing the table polynomial encodings of instructions within subcubes. Figure 23 provides the proving protocol details. As in the adaption of the vector lookup argument, we use the Binius hypercube shift operator  $\widetilde{shft}_b$  [DP23]. Whereas  $\widetilde{shft}_b$  shifts the *b* low-order bits of the boolean representation, we will further define the convenience operator  $\widetilde{shft}_b(f(X_\mu))$  to shift the *b* high-order bits.

**Security.** We prove the completeness, knowledge soundness, and zero knowledge of Mux-HyperPLONK in the following two theorems. The zero-knowledge of Mux-HyperPLONK is achieved through the multivariate zero-knowledge compiler of Chen et al. [CBBZ23] observing that Mux-HyperPLONK satisfies the domain admissibility requirements.

**Theorem 16.** Mux-HyperPLONK for  $R_{MExe,n}[R_{plonk}]$  (Figure 23) inherits completeness error from its underlying SubcubeLkup look up protocol, and for any adversary A against knowledge soundness, we provide an extractor X using

$$\mathsf{R}_{\mathsf{plonk}} = \left\{ \begin{pmatrix} & x(\{0,1\}^{\log d_x}) = x \\ & x(\{0,1\}^{\log d_x}) = x \\ & x = x \parallel w \in \mathbb{F}^{d \times \ell_z} \\ & z = x \parallel w \in \mathbb{F}^{d \times \ell_z} \\ & \vdots \bigwedge_{i \in [d]} G([sel_{i,j}]_{j \in [\ell_s]}, [z_{i,j}]_{j \in [\ell_z]}) = 0 \\ & \bigwedge_{i \in [d\ell_z]} z_{\lfloor i/d \rfloor, i \mod d} = z_{\lfloor \sigma(i)/d \rfloor, \sigma(i) \mod d} \\ & \end{pmatrix} \right\}$$

 $\mathsf{HyperPLONK}.\mathsf{Setup}(\lambda) : \mathsf{Return} \perp$ 

 $\mathsf{HyperPLONK}.\mathsf{Index}(\bot,(\mathbb{F},sel,\sigma,G,\ell_s,\ell_z,d,d_{\mathsf{x}}))$ 

(1) Encode selector polynomial  $\widetilde{sel}$  as the MLE setting the following evaluations over  $\{0,1\}^{\log d\ell_s}$ :

$$\left[\left[\widetilde{sel}(i,j) = sel_{\operatorname{int}_{\log d}(i),\operatorname{int}_{\log \ell_s(j)}}\right]_{i \in \{0,1\}^{\log d}}\right]_{j \in \{0,1\}^{\log \ell_s}}$$

(2) Encode permutation polynomial  $\tilde{\sigma}$  as the MLE setting the following evaluation over  $\{0,1\}^{\log d\ell_z}$ :

$$\left[\left[\widetilde{\sigma}(i,j) = \sigma(\operatorname{int}_{\log \ell_z}(j) \cdot \ell_z + \operatorname{int}_{\log d}(i))\right]_{i \in \{0,1\}^{\log d}}\right]_{j \in \{0,1\}^{\log \ell_z}}.$$

 $(3) \ \operatorname{Return} \Big( pp \leftarrow \Big(\widetilde{sel}, \widetilde{\sigma}, G\Big), vp \leftarrow \Big( [\![\widetilde{sel}]\!], [\![\widetilde{\sigma}]\!], G \Big) \Big).$ 

$$\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{plonk}}] = \left\{ \begin{pmatrix} & x_0(\{0,1\}^{\log d_x/2}) = [inst_0, mem_0] \\ & ([\mathbb{F}, d, d_x, G, \ell_s, \ell_z, [sel_i, \sigma_i]_{i \in [\ell]}] \end{pmatrix}, \\ & ([\mathbb{I}x_0]], [[x_n]]), \\ & ([inst_j, mem_j, w_j]_{j \in [n]}, x_0, x_n) \end{pmatrix} \right\} \xrightarrow{x_0(\{0,1\}^{\log d_x/2}) = [inst_n, mem_n]}_{j \in [n]} \left\{ \begin{pmatrix} (\mathbb{F}, sel_{inst_j}, \sigma_{inst_j}, G, \ell_s, \ell_z, d, d_x), \\ & [inst_j, mem_j, inst_{j+1}, mem_{j+1}], \\ & w_j \end{pmatrix} \in \mathsf{R}_{\mathsf{plonk}} \right\}$$

 $\mathsf{Mux-HyperPLONK}.\mathsf{Setup}(\lambda) : \mathsf{Return} \perp$ 

 $\underbrace{\mathsf{Mux-HyperPLONK.Index}\Big(\bot, \Big(\mathbb{F}, d, d_x, G, \ell_s, \ell_z, [sel_i, \sigma_i]_{i \in [\ell]}\Big)\Big)}_{=}$ 

(1) Compute the prover parameter polynomials for each instruction index using the HyperPLONK indexer.

$$\left[ \left( \begin{array}{c} pp_i \leftarrow \left(\widetilde{sel}_i, \widetilde{\sigma}_i, G\right), \\ vp_i \leftarrow \left( \widetilde{[sel}_i], [\widetilde{\sigma}_i], G \right) \end{array} \right) \leftarrow \mathsf{HyperPLONK}.\mathsf{Index}(\bot, (\mathbb{F}, sel_i, \sigma_i, G, \ell_s, \ell_z, d, d_x) \right]_{i \in [\ell]} \right]$$

(2) Construct table selector polynomial  $\widetilde{tsel} \in \mathbb{F}[X_{\lfloor \log d\ell_s \ell \rfloor}]$  and table permutation polynomial  $\widetilde{t\sigma} \in \mathbb{F}[X_{\lfloor \log d\ell_z \ell \rfloor}]$  as the MLE setting the evaluation of subcubes to correspond to each instruction:

$$\left[\left[\left[\widetilde{tsel}(k,j,i)=\widetilde{sel_i}(k,j)\right]_{k\in\{0,1\}^{\log d}}\right]_{j\in\{0,1\}^{\log \ell_s}}\right]_{i\in\{0,1\}^{\log \ell}} \left[\left[\left[\widetilde{t\sigma}(k,j,i)=\widetilde{\sigma_i}(k,j)\right]_{k\in\{0,1\}^{\log d}}\right]_{j\in\{0,1\}^{\log \ell_s}}\right]_{i\in\{0,1\}^{\log \ell}}$$

$$(3) \text{ Return } \left(pp\leftarrow\left(\widetilde{tsel},\widetilde{t\sigma},G\right), vp\leftarrow\left([\widetilde{tsel}],[\widetilde{t\sigma}],G\right)\right).$$

Figure 22: Setup algorithm for HyperPLONK encoding selectors and permutation as multivariate polynomials (top). Setup algorithm for Mux-HyperPLONK encoding HyperPLONK computation commitment values for each instruction into subcubes of an evaluation hypercube domain for the machine commitment (bottom).

Figure 23: Mux-HyperPLONK: HyperPLONK polyIOP adapted with multivariate vector lookup for unrolled machine execution.

 $X_{zt}$ , an extractor for ZeroTest, using  $X_{plonk}$ , an extractor for HyperPLONK, and using  $X_{lk}$ , an extractor for SubcubeLkup, such that

$$\begin{split} &\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{Mux-HyperPLONK},\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{plonk}}],\mathsf{X},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(d_{x}n/2)},\log(d\ell_{s}n)+\log(n)),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ 2\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(d_{x}/2)},\log(d\ell_{s}n)),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 3\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}(\{0,1\}^{\log(d\ell_{z}n)},\log(d\ell_{z}n)+\log(n)),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\mathrm{sound}}_{2\operatorname{-SubcubeLkup}(\{0,1\}^{\log(d\max(\ell_{s},\ell_{z})n)},\{0,1\}^{\log(d\max(\ell_{s},\ell_{z})\ell)}),\mathsf{R}_{\mathsf{vlkup}},\mathsf{X}_{\mathsf{lk}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{\mathrm{sound}}_{\mathsf{PLONK}(\{0,1\}^{\log(d\ell_{z}n)},\ell_{s},\ell_{z}),\mathsf{R}_{\mathsf{plonk}},\mathsf{X}_{\mathsf{plonk}},\mathcal{A}}(\lambda). \end{split}$$

*Proof.* <u>Completeness.</u> The completeness directly follows from that of Theorem 8, except that we change to multivariate encodings and we use  $\widetilde{shfth}$  to traverse subcubes.

<u>Knowledge Soundness</u>. We bound the advantage of adversary A by bounding the advantage of each of a series of game hops [BR06]. We define  $G_0 =$ 

SOUND<sup>A</sup><sub>Mux-HyperPLONK,R<sub>MExe,n</sub>[R<sub>plonk</sub>], $\mathbf{x}(\lambda)$ . The inequality above follows from the following claims that we will justify: (1)</sub>

$$\begin{split} |\Pr[\mathbf{G}_0 = 1] - \Pr[\mathbf{G}_1 = 1]| &\leq \mathsf{Adv}_{\mathsf{ZeroTest}(\{0,1\}^{\log(d_{\mathsf{X}}n/2)}, \log(d\ell_s n) + \log(n)), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ 2\mathsf{Adv}_{\mathsf{ZeroTest}(\{0,1\}^{\log(d_{\mathsf{X}}/2)}, \log(d\ell_s n)), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(2)

$$\begin{split} |\Pr[G_1 = 1] - \Pr[G_2 = 1]| &\leq \mathsf{Adv}_{2\mathsf{-SubcubeLkup}(\{0,1\}^{\log(d\max(\ell_s,\ell_z)n)},\{0,1\}^{\log(d\max(\ell_s,\ell_z)\ell)}),\mathsf{R}_{\mathsf{vlkup}},\mathsf{X}_{\mathsf{lk}},\mathcal{A}}(\lambda) \\ &+ 3\mathsf{Adv}_{\mathsf{ZeroTest}(\{0,1\}^{\log(d\ell_zn)},\log(d\ell_zn) + \log(n)),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \end{split}$$

(3) 
$$|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \le \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{PLONK}(\{0,1\}^{\log(d\ell_z n)}, \ell_s, \ell_z), \mathsf{R}_{\mathsf{plonk}}, \mathsf{X}_{\mathsf{plonk}}, \mathcal{A}}(\lambda)$$

(4) 
$$\Pr[G_3 = 1] = 0$$

The proof idea follows closely to the proof of Theorem 8. Claim 1 argues for the well-formedness of the statement polynomial x. Claim 2 argues for the well-formedness of the index polynomials  $\widetilde{sel}$ ,  $\tilde{\sigma}$  to invoke HyperPLONK. Claim 3 argues that the Plonk relation for the unrolled execution is satisfied. Lastly Claim 4 argues that the constructed extractor always succeeds for an accepting verifier.

Claim 1: Instead of having  $\mathbb{V}_{in}$  and  $\mathbb{V}_{out}$  for the first and last statement respectively, in boolean hypercube we use 0 in the  $([\log d_x/2] + 1)$ -th input for the input hypercube, and 1 for the output hypercube. Then the first statement is recorded in the first out of n instructions in the input hypercube, and the last statement is recorded in the last out of n instructions in the output hypercube.

•

$$\begin{aligned} & (\widetilde{x}(X_{[\log d_x/2]}, 1, 0_{[\log d\ell_z/d_x]}, X_{[\log d_x/2, \log d_x n/2]}) \\ & - \widetilde{shfth}_{\log n}(\widetilde{x})(X_{[\log d_x/2]}, 0, 0_{[\log d\ell_z/d_x]}, X_{[\log d_x/2, \log d_x n/2]})) \\ & \cdot \left(1 - \widetilde{eq}_{\log n}(X_{[\log d_x/2, \log d_x n/2]}, 1_{[\log n]})\right) = 0 \end{aligned}$$

over  $\{0,1\}^{\log d_x n/2}$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}(\{0,1\}^{\log(d_x n/2)}, \log(d\ell_s n) + \log(n)), \mathsf{R}_{\operatorname{Zero},\mathsf{X}_{\operatorname{zt}},\mathcal{A}}(\lambda)$ : Checks the output statement is the same as the shifted input statement except the last output (which was checked by 3(b-c) instead). In other words, the next input statement  $[inst_{j+1}, mem_{j+1}]$  should be the same as the current output statement  $[inst_j, mem_j]$ .

 $G_1$  employs  $X_{zt}$  for all the tests described above and aborts if the extractor fails.

Claim 2: In this step, we encode  $\ell_s$  different selector polynomials into one using the  $(\log d + 1)$ -th to  $(\log d + \log \ell_s)$ -th inputs to select among them, and encode  $\ell_z$  permutation polynomials into one using the  $(\log d + 1)$ -th to  $(\log d + \log \ell_z)$ -th inputs. Then the subcube inclusion

$$\begin{split} &\left\{\left[\left(\widetilde{sel}(j,i),\widetilde{\sigma'}(j,i)\right)\right]_{j\in\{0,1\}^{\log d\max(\ell_s,\ell_z)}}\right\}_{i\in\{0,1\}^{\log n}} \\ &\subseteq \left\{\left[\left(\widetilde{tsel}(j,i),\widetilde{t\sigma}(j,i)\right)\right]_{j\in\{0,1\}^{\log d\max(\ell_s,\ell_z)}}\right\}_{i\in\{0,1\}^{\log \ell}} \end{split}$$

is by the soundness of 2-SubcubeLkup.

The wellformedness of the shift polynomial s and  $\tilde{\sigma}$  are proven in the same way as in the Claim 2 of Theorem 8. The difference besides encoding is that in hypercube we use  $\widetilde{shfth}_{\log n}$  for traversing.

*Claim 3-4:* The last steps also follow from Theorem 8 claim 3-4. We use HyperPLONK to prove the multivariate Plonk relation instead.

**Theorem 17.** The compiled polyIOP using the compiler in [CBBZ23] of Mux-HyperPLONK for  $R_{MExe,n}[R_{plonk}]$  (Figure 23) is honest-verifier zero-knowledge.

*Proof.* Since verifier in Mux-HyperPLONK never makes queries outside ZeroTest, it can be made HVZK by the same reasonings as in Theorem 14 and by the HVZK of underlying HyperPLONK after polynomial maskings.  $\Box$ 

# **E** Mux-Marlin Construction

We begin by reviewing some preliminaries of the Marlin proof system [CHM<sup>+</sup>20].

**Rank-1 constraint satisfiability (R1CS).** A common arithmetization used in proofs for relations in NP is rank-1 constraint satisfiability (R1CS).

**Definition 18.** An R1CS relation is indexed by the tuple  $(\mathbb{F}, A, B, C, d, d_x)$  where  $A, B, C \in \mathbb{F}^{d \times d}$ . The statement and witness  $\begin{bmatrix} w \\ x \end{bmatrix} \in \mathbb{F}^d$  together form a vector (with the length of the statement  $d_x \leq d$  specified) where the following algebraic relation is satisfied:

$$\mathsf{R}_{\mathsf{r1cs}} = \left\{ \begin{pmatrix} (\mathbb{F}, A, B, C, d, d_x), \\ x, \\ w \end{pmatrix} : A \begin{bmatrix} w \\ x \end{bmatrix} \circ B \begin{bmatrix} w \\ x \end{bmatrix} = C \begin{bmatrix} w \\ x \end{bmatrix} \right\}$$

The typical R1CS formulation also specifies an additional parameter for an upper bound on the number of non-zero entries in the matrices A, B, C. In this work, we will assume the number of non-zero entries in the matrices is equal to the dimension of the matrices d; this can be done by extending the matrices with trivial zero rows and columns. This adaptation will be important for our application of tuple lookups for machine execution.

Marlin polyIOP for R1CS. Figure 24 gives a description of the Marlin polyIOP with slight modification to account for our simplification of the R1CS where the dimensions of the matrices are equal to the number of non-zero elements. We change the formulation of R1CS slightly to fit that of a polyIOP. We define a polynomial z that encodes  $\begin{bmatrix} w \\ x \end{bmatrix}$  as evaluations over a multiplicative subgroup  $\mathbb{H} \leq \mathbb{F}$  where  $|\mathbb{H}| = d$  and, more precisely, further encodes x as evaluations over a subgroup  $\mathbb{H}_x \leq \mathbb{H}$  where  $|\mathbb{H}_x| = d_x$ . For zero knowledge purposes, we also define a group  $\mathbb{K}$  as an input to be part of the restricted domain.

Marlin makes use of the following bivariate polynomial  $u_{\mathbb{H}}$  which for a multiplicative subgroup  $\mathbb{H}$  can be expressed as follows:

$$u_{\mathbb{H}}(X,Y) = \frac{Z_{\mathbb{H}}(X) - Z_{\mathbb{H}}(Y)}{X - Y} = \frac{X^d - Y^d}{X - Y} = \sum_{i=0}^{d-1} X^i Y^{d-1-i},$$

$$\begin{aligned} \left| \begin{array}{l} R_{\text{Liss}} = \left\{ \left( \begin{array}{c} (\mathbb{F}, \mathbb{H}, \mathbb{H}_{x}, A, B, C), \\ ([x]], \mathbb{K}), \\ (w, x) \end{array} \right) : \begin{array}{c} A\left[ \begin{bmatrix} w \\ x \end{bmatrix} \in B\left[ \begin{bmatrix} w \\ x \end{bmatrix} \right] = C\left[ \begin{bmatrix} w \\ x \end{bmatrix} \right] \\ x(\mathbb{H}_{x}) = x \end{array} \right\} \end{aligned} \right. \\ \\ \begin{aligned} & \text{Marlin,Setup}(\lambda): \text{Return } \bot \\ & \text{Marlin,Index}(\bot, (\mathbb{F}, \mathbb{H}, \mathbb{H}_{x}, A, B, C)) \\ (1) \text{ For each matrix } M \in [A, B, C], \text{ compute the polynomials } row_{M}, col_{M}, val_{M} \text{ using evaluations defined over } \mathbb{H}. \text{ Assume a canonical mapping} \\ \phi_{M} \text{ from the indices } [0, d] \text{ to the non-zero elements } of M. \\ & - \text{ Set } [row_{M}(\omega')] = \omega^{2}]_{t=0}^{d-1} \text{ where } j \text{ is the row of the element } \phi_{M}(i) \text{ in } M. \\ & - \text{ Set } [cal_{M}(\omega')] = \omega^{2}]_{t=0}^{d-1} \text{ where } j \text{ is the column of the clement } \phi_{M}(i) \text{ in } M. \\ & - \text{ Set } [val_{M}(\omega')] = \frac{\phi_{M}(i)}{u_{\text{ell}}(row_{M}(\omega'), row_{M}(\omega')), u_{\text{ell}}(cal_{M}(\omega'), cal_{M}(\omega'))]} \right]_{t=0}^{d-1}. \\ \end{array} \right] \\ (2) \text{ Return } \left( p\phi \leftarrow [(row_{M}, cal_{M}, val_{M})]_{M \in [A, B, C]}, \psi \leftarrow [([row_{M}], [[col_{M}], [val_{M}]])]_{M \in [A, B, C]}, ([x], \mathbb{K})) \\ (1) \text{ P computes and sends polynomial  $z$  that evaluates to the vector  $\mathbb{H}_{x}. \\ (2) \text{ For each matrix } M \in [A, B, C], \text{ P computes and sends the polynomial  $z_{M}$  that evaluates to the vector  $M \cdot \begin{bmatrix} w \\ w \end{bmatrix} \text{ over } \mathbb{H}. \\ (3) \text{ V sends random challenge } r_{1} \in \mathbb{F} \setminus (\mathbb{H} \cup \mathbb{K}). \\ (4) \text{ For each matrix } M \in [A, B, C], \text{ P computes and sends the polynomial  $z_{M}: \\ w = 0 \text{ computes and sends polynomial  $y_{M} \text{ as follows:} \\ \\ q_{M}(X) = \left( z(X) \cdot \sum_{\omega \in \mathbb{H}} u_{W}(\omega)) \cdot u_{W}(X, col_{M}(\omega)) \cdot val_{M}(\omega) \right) - z_{M}(X) \cdot |\mathbb{H}|^{-1} \\ (b) \text{ P and V engage in SumCheck}(\mathbb{H} \cup \mathbb{K}) \text{ protocol to prove  $y_{M} \text{ suns to 0 over } \mathbb{H}. \\ (c) \text{ V sends random challenge  $r_{M} \in \mathbb{F} \setminus (\mathbb{H} \cup \mathbb{K}). \\ (d) \text{ P proves well-formedness of  $q_{M}: \\ - \text{ P ond V engage in SumCheck}(\mathbb{H} \cup \mathbb{U} \text{ proveoto to prove  $q_{M} \text{ suns to 0 over } \mathbb{H}. \\ (b) \text{ P ond V engage in SumCheck}(\mathbb{H} \cup \mathbb{H} \cup \mathbb{K}) \text{ protocol to prove  $row_{M}(x_{M}) | r_{M}(z_{M}) | r_{M}(z_$$$$$$$$$$$

#### Figure 24: Marlin: Marlin polyIOP [CHM<sup>+</sup>20].

which has a few properties we will make use of. Namely, for  $x, y \in \mathbb{H}$  when  $\mathbb{H}$  is a multiplicative subgroup,  $u_{\mathbb{H}}(x, y) = 0$  when  $x \neq y$  and  $u_{\mathbb{H}}(x, y) = dx^{d-1}$  when x = y.

**Construction.** We define multiplicative subgroups of the following sizes to be used to encode components of the unrolled machine execution as polynomials:

- Define V = ⟨γ⟩ as the multiplicative subgroup of size d where the R1CS instruction index has matrices of dimension d.
- Define G = ⟨μ⟩ as the multiplicative subgroup of size dn where n is the number of unrolled execution steps.
   Denote the n cosets of V in G as [μ<sup>i</sup>V]<sup>n-1</sup><sub>i=0</sub>.
- Define ℍ = ⟨ω⟩ as the multiplicative subgroup of size dℓ where ℓ is the number of instructions. Denote the ℓ cosets of V in ℍ as [ω<sup>i</sup>V]<sup>ℓ-1</sup><sub>i=0</sub>.
- Define  $\mathbb{V}_x \leq \mathbb{V}$  as the multiplicative subgroup of size  $d_x$  of the R1CS instruction index where  $d/d_x = a$ .
- Define  $\mathbb{V}_{in} \leq \mathbb{V}_x$  as the multiplicative subgroup of size  $d_x/2$  generated by  $\mu^{\frac{2nd}{d_x}}$  and  $\mathbb{V}_{out} = \mu^{\frac{2nd}{d_x}} \mathbb{V}_{in}$  as the other

$$\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{rlcs}}] = \left\{ \begin{pmatrix} \left(\mathbb{F}, \mathbb{G}, \mathbb{H}, \mathbb{V}, \mathbb{V}_{\mathsf{x}}, \mathbb{V}_{in}, [A_i, B_i, C_i]_{i=0}^{\ell-1}\right), \\ \left(\mathbb{I}_{x_0}], [x_n]\right), \\ \left([inst_j, mem_j, w_j]_{j=0}^n, x_0, x_n\right) \end{pmatrix} \right) \xrightarrow{n}_{j=0} \begin{pmatrix} x_0(\mathbb{V}_{in}) = [inst_0, mem_0] \\ x_n(\mathbb{V}_{out}) = [inst_n, mem_n] \\ \vdots \\ n-1 \\ \left(\mathbb{F}, A_{inst_j}, B_{inst_j}, C_{inst_j}, |\mathbb{V}_{\mathsf{x}}|\right), \\ [inst_j, mem_j, inst_{j+1}, mem_{j+1}], \\ w_j \end{pmatrix} \in \mathsf{R}_{\mathsf{rlcs}} \right\}$$

 $\mathsf{Mux-Marlin}.\mathsf{Setup}(\lambda):\mathsf{Return}\perp$ 

Mux-Marlin.Index  $\left( \perp, \left( \mathbb{F}, \mathbb{G}, \mathbb{H}, \mathbb{V}, \mathbb{V}_x, \mathbb{V}_{in}, [A_i, B_i, C_i]_{i=0}^{\ell-1} \right) \right)$ 

(1) Compute the prover parameter polynomials for each instruction index using Marlin.

$$\left(\begin{array}{c} pp_i \leftarrow \left[\left(row_{M,i}, col_{M,i}, val_{M,i}\right)\right]_{M \in [A,B,C]}, \\ vp_i \leftarrow \left[\left(\left[row_{M,i}\right], \left[col_{M,i}\right], \left[val_{M,i}\right]\right)\right]_{M \in [A,B,C]}\end{array}\right) \leftarrow \mathsf{Marlin.Index}(\bot, (\mathbb{F}, \mathbb{V}, \mathbb{V}_x, A_i, B_i, C_i)\right]_{i \in [\ell]}$$

(2) For  $M \in [A, B, C]$ , construct polynomials  $trow_M, tcol_M$  over  $\mathbb{H}$  by interpolating over the following defined evaluations of the cosets  $[\omega^i \mathbb{V}]_i^{\ell-1}$  of  $\mathbb{H}$ :

$$\left[trow_{M}(\omega^{i}\mathbb{V}) = row_{M,i}(\mathbb{V})\right]_{i=0}^{\ell-1} \qquad \left[tcol_{M}(\omega^{i}\mathbb{V}) = col_{M,i}(\mathbb{V})\right]_{i=0}^{\ell-1}$$

(3) For  $M \in [A, B, C]$ , create polynomial  $tval_M$  over  $\mathbb{H}$  by interpolating over the following defined evaluations of the cosets  $[\omega^i \mathbb{V}]_i^{\ell-1}$  of  $\mathbb{H}$ . Instead of directly using the normalized value polynomials from Marlin, recompute with unnormalized values. Assume canonical mappings  $[\phi_i]_i^{\ell-1}$  between the non-zero elements of  $M_i$  and the elements of  $\mathbb{V}$ .

$$\left[\left[tval_{M}(\omega^{i}\gamma^{k})=\phi_{i}(\gamma^{k})\right]_{k=0}^{d}\right]_{i=0}^{d-1}$$

 $(4) \operatorname{Return}\left(pp \leftarrow \left[\left(trow_{M}, tcol_{M}, tval_{M}\right)\right]_{M \in [A, B, C]}, vp \leftarrow \left[\left(\left[trow_{M}\right], \left[tcol_{M}\right], \left[tval_{M}\right]\right)\right]_{M \in [A, B, C]}\right)$ 

Figure 25: Setup for Mux-Marlin: Adapted Marlin polyIOP for unrolled machine execution.

coset of  $\mathbb{V}_{in}$  in  $\mathbb{V}_x$  encoding the two parts of the machine execution statement.

Figure 25 and Figure 26 provide the details for our adaptation of the Marlin proof system for unrolled machine execution. As in Mux-PLONK, computation commitments for each instruction are encoded into table polynomials representing a machine commitment.

Each instruction in the instruction set of size  $\ell$  can be represented as an R1CS computation with coefficient matrices  $[(A_i, B_i, C_i)]_{i=0}^{\ell}$ , where in turn, each matrix M is encoded by row, column, and value polynomials  $[(row_{M,i}, col_{M,i}, val_{M,i})]_{M \in [A,B,C]}$ . Take for example the row polynomial encoding. We encode together the row positions for each instruction in the instruction set into a single "table" polynomial  $trow_M$  defined over the larger group  $\mathbb{H}$  as:

$$[trow_M(\omega^i \mathbb{V}) = row_{M,i}(\mathbb{V})]_{i=0}^{\ell-1}.$$

To produce a row encoding for a machine computation that executes n instructions  $[inst_j \in [\ell]]_{j=0}^n$ , we define a new polynomial  $row'_M$  over  $\mathbb{G}$  that "looks up" the row encodings for each executed instruction from  $trow_M$ . The wellformedness of such a polynomial can be proved using exactly our new vector lookup argument:

$$\begin{bmatrix} row'_M(\mu^j \mathbb{V}) = trow_M(\omega^{inst_j} \mathbb{V}) \end{bmatrix}_{j=0}^{n-1} \qquad M' = \begin{bmatrix} M_{inst_0} & \dots & M_{inst_{n-1}} \\ 0 & 0 & 0 \\ \vdots & 0 & \ddots \end{bmatrix} \begin{bmatrix} \mathbb{V} \\ \vdots \\ \mu^{n-1} \mathbb{V} \end{bmatrix}$$

Unfortunately,  $row'_M$  is not the correct row encoding for the sparse matrix describing the executed machine computation. To see this, observe that  $row'_M$  encodes a matrix (and corresponding computation) of size  $|\mathbb{G}| = mn$ . However, the encoded row positions that are looked up from  $trow_M$  are fixed to the range of  $trow_M$  which is  $\mathbb{V}$ . This results in a row encoding for a matrix M' in which all instructions are encoded in the rows corresponding to  $\mathbb{V}$ . Instead, to capture the correct global structure of the desired executed machine computation, the matrix M should encode the

$$\begin{aligned} & \left| \begin{array}{l} \mathsf{Mux}.\mathsf{Marlin}.\mathsf{P} \left( \begin{array}{l} [(trow_{M}, tcol_{M}, tval_{M})]_{M \in [A, B, C]}, \\ ([tota)_{1}, [[tota)_{1}, [[tota)_{M}]_{1, [(tota)_{M}]_{1, [(t$$

#### Figure 26: Mux-Marlin: Adapted Marlin polyIOP for unrolled machine execution.

instructions along the diagonal:

$$\left[ row_M(\mu^j \mathbb{V}) = \mu^j \cdot row'_M(\mu^j \mathbb{V}) \right]_{j=0}^{n-1} \qquad M = \begin{bmatrix} M_{inst_0} & 0 & \dots \\ 0 & \ddots & 0 \\ \vdots & 0 & M_{inst_{n-1}} \end{bmatrix} \begin{bmatrix} \mathbb{V} \\ \vdots \\ \mu^{n-1} \mathbb{V} \end{bmatrix}$$

With this observation, the encoded row mappings for the individual instructions to  $\mathbb{V}$  can be corrected by offsetting them by their position in the execution computation, i.e., the mappings for  $j^{th}$  executed instruction need to be offset by  $\mu^j$  to instead map to coset  $\mu^j \mathbb{V}$ . We provide a protocol for proving the correct offset shift of  $row_M$  with respect to  $row'_M$ . The prover interpolates a shift polynomial *s* defined over  $\mathbb{G}$  such that each coset evaluates to the desired shift: for all  $j \in [0, n)$ , the evaluation  $s(\mu^j \mathbb{V}) = \mu^j$ . As before using standard zero test protocols (see preliminaries in Section 3.3), the prover will prove the following polynomial identities to convince the verifier of  $row_M$  wellformedness:

• s(1) = 1: The first element of s is anchored to equal to 1.

- $s(\gamma X) = s(X)$  over  $\mathbb{V}$ : Every element of the first coset  $\mathbb{V}$  is the same, i.e., set to 1.
- $(s(\mu X) \mu \cdot s(X)) \cdot Z_{\mu^{n-1}\mathbb{V}}(X) = 0$  over  $\mathbb{G}$ : The next element in  $\mathbb{G}$  is equal to  $\mu$  times the previous element in  $\mathbb{G}$  (excluding the last coset). Since the first coset is set to 1, this ensures that each coset is set to the next power of  $\mu$ .
- $row_M(X) = row'_M(X) \cdot s(X)$  over  $\mathbb{G}$ : This directly checks the offset shift between  $row_M$  and  $row'_M$ .

The column vector  $col_M$  follows the same strategy as  $row_M$ , while  $val_M$  takes a different approach. The polynomial  $val_M$  is defined to be normalized over group  $\mathbb{G}$ :

$$val_M(\omega^i) = \frac{\phi(\omega^i)}{u_{\mathbb{G}}(row_M(\omega^i), row_M(\omega^i)) \cdot u_{\mathbb{G}}(col_M(\omega^i), col_M(\omega^i))}$$

However, in our new Mux-Marlin construction, we do not know group  $\mathbb{G}$  (whose size depends on number of unrolling steps) beforehand and hence cannot create the table value polynomials  $tval_M$  in normalized form. We could either modify Marlin to use unnormalized  $val_M$  or normalize  $val'_M$  before calling the Marlin subroutine. We choose the later to perform minimal changes to Marlin and by observing the following fact:

$$\forall \mu \in \mathbb{G}, \quad val_{M}(\mu) = \frac{val'_{M}(\mu)}{u_{\mathbb{G}}(row_{M}(\mu), row_{M}(\mu)) \cdot u_{\mathbb{G}}(col_{M}(\mu), col_{M}(\mu))}$$

$$= \frac{val'_{M}(\mu)}{|\mathbb{G}|^{2} \cdot row_{M}(\mu)^{|\mathbb{G}|-1} \cdot col_{M}(\mu)^{|\mathbb{G}|-1}}$$

$$= \frac{val'_{M}(\mu) \cdot row_{M}(\mu) \cdot col_{M}(\mu)}{|\mathbb{G}|^{2}}$$

$$(2)$$

As a result, the prover can send  $val_M$  to the verifier and they together perform a ZeroTest to check that  $|\mathbb{G}|^2 \cdot val_M(X) = val'_M(X) \cdot row_M(X) \cdot col_M(X)$  over  $\mathbb{G}$ .

**Zero knowledge.** Following the strategy of Mux-PLONK and Mux-HyperPLONK, we would like to be able to directly call Marlin as a subroutine within Mux-Marlin. Unfortunately, this strategy does not work: Marlin requires querying index polynomial commitments *row*, *col*, *val* outside of zero test polynomial identities which means we cannot apply our zero knowledge compiler. We would like to show that Marlin can be simulated without knowledge of the underlying index. It turns out that this stronger notion of zero-knowledge was studied recently by Boneh et al. [BNO21] termed *index-hiding*. Boneh et al. incorrectly claim that Marlin is index-hiding. Here, we present a variant of Marlin that provides index-hiding that we term ZK-Marlin. The full details are given in Figure 28 and the formal definition for index-hiding is given in Figure 27.

**Index-Hiding.** An argument system is *index-hiding zero-knowledge* if the interactive protocol does not leak any information to the verifier besides membership in the relation, not even information about the oracles sent to the verifiers. We define security via the pseudocode game ZK in Figure 27 in which an adversary is tasked with distinguishing between an honest-verifier interaction with a prover with knowledge of a valid witness, and a simulated interaction without a witness and importantly, without oracles. In the pseudocode, View denotes the view of the verifier consisting of the transcript of prover messages.

**Definition 19.** An *n*-round protocol  $\Pi$  is index-hiding zero-knowledge if there exists a simulator S such that for all  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ , the following advantage probability is negligible in  $\lambda$ :

$$\mathsf{Adv}_{\mathsf{\Pi},\mathsf{R},\mathsf{S},n,\mathcal{A}}^{\mathrm{ID}\text{-}\mathrm{ZK}}(\lambda) = \left| \Pr \Big[ \mathrm{ID}\text{-}\mathrm{ZK}_{\mathsf{\Pi},\mathsf{R},\mathsf{S},n}^{\mathcal{A},1}(\lambda) = 1 \Big] - \Pr \Big[ \mathrm{ID}\text{-}\mathrm{ZK}_{\mathsf{\Pi},\mathsf{R},\mathsf{S},n}^{\mathcal{A},0}(\lambda) = 1 \Big] \right|$$

We provide the following three theorems, first showing that ZK-Marlin is knowledge sound and complete, second showing the ZK-Marlin is index-private, and third showing that Mux-Marlin using ZK-Marlin as a subroutine provides zero-knowledge.

**Index-Hiding Marlin.** Because the Marlin proof system does not reduce down to zero tests (it makes verifier queries outside of a zero test), we cannot use it directly with our ZK compiler. We define an index-private Marlin protocol modified from Marlin protocol in Figure 28.

$$\begin{array}{l} & \underbrace{ \operatorname{Game}\,\operatorname{ID-ZK}_{\Pi,\mathsf{R},\mathsf{S},n}^{\mathcal{A},b}(\lambda) } \\ & \overline{gp_1 \leftarrow^{\$} \Pi.\operatorname{Setup}(\lambda) \; ; \; (gp_0,st_\mathsf{S}) \leftarrow^{\$} \mathsf{S}.\operatorname{Setup}(\lambda) } \\ & (i,x,w,st_\mathcal{A}) \leftarrow^{\$} \mathcal{A}_1(gp_b) \\ & (vp,pp) \leftarrow \Pi.\operatorname{Index}(gp_b,i) \\ & vw_1 \leftarrow \operatorname{View}\langle \Pi.\mathsf{P}(pp,x,w,\bot) \leftrightarrow \Pi.\mathsf{V}(vp,x,\bot) \rangle_n \\ & vw_0 \leftarrow \operatorname{S.Sim}\operatorname{View}(x,st_\mathsf{S}) \\ & \operatorname{Return}\; \bigwedge \left( \begin{array}{c} \mathcal{A}_2(vw_b,st_\mathcal{A}) \\ & (i,x,w) \in \mathsf{R} \end{array} \right) \end{array}$$

Figure 27: index private security games for interactive argument systems.

We can encode the vector  $\begin{bmatrix} w \\ x \end{bmatrix}$  by labeling the entires with elements of  $\mathbb{H}$ . Similarly, we let  $z_A, z_B, z_C$  encode the vectors Az, Bz, Cz, respectively. Then we can think of Marlin protocol as three separate checks:

- 1.  $Az \circ Bz \stackrel{?}{=} Cz : z_A(\mathbb{H}) \cdot z_B(\mathbb{H}) \stackrel{?}{=} z_C(\mathbb{H}).$
- 2. Whether  $z_A, z_B, z_C$  correctly encode the matrix-vector product: For all  $M \in \{A, B, C\}$ , for all  $\omega \in \mathbb{H}$ ,  $z_M(\omega) \stackrel{?}{=} \sum_{\omega' \in \mathbb{H}} M_{\omega,\omega'} \cdot z(\omega')$
- 3. Whether z correctly encodes x in  $\mathbb{H}_x$ :  $z(\mathbb{H}_x) \stackrel{?}{=} x(\mathbb{H}_x)$

The first and last checks can be made zero-knowledge by zero-knowledge zero test easily. Let  $\tilde{M}$  be the unique bivariate polynomial that extends M. Then check 2 is equivalent to check if the following equality holds over  $\mathbb{H}$ :  $z_M(X) - \sum_{\omega \in \mathbb{H}} \tilde{M}(X, \omega) z(\omega) \equiv 0 \mod Z_{\mathbb{H}}(X)$ . Since it is unclear how to do the summation in zero test, we can instead think of  $\sum_{\omega \in \mathbb{H}} \tilde{M}(X, \omega) z(\omega)$  as an univariate polynomial. Then the zero test would check whether there exists a quotient polynomial  $q_{1,M}(X)$  such that  $q_{1,M}(r_1)Z_{\mathbb{H}}(r_1) \stackrel{?}{=} z_M(r_1) - \sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) z(\omega)$  at a random point  $r_1$  up to some soundness error. Instead of asking the verifier to query  $\sum_{\omega \in \mathbb{H}} \tilde{M}(X, \omega) z(\omega)$  at  $r_1$ , the prover sends  $\sigma_1$  that claims to be equal to  $\sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) z(\omega)$ .

We are left to check that  $\sigma_1$  encodes the correct sum over  $\mathbb{H}$  at point  $r_1$ . Define a polynomial  $S_M$  that claims to satisfy  $S_M(\mathbb{H}) = \tilde{M}(r_1, \mathbb{H})z(\mathbb{H})$ . The prover first uses a sum check to prove  $S_M$  sums to  $\sigma_1$  over  $\mathbb{H}$ . Then it proves that  $S_M$  indeed has the same evaluation as  $\tilde{M}(r_1, X)z(X)$  over  $\mathbb{H}$ . Ideally this can be done with a zero test, but it would be too costly for the verifier to evaluate the bivariate polynomial  $\tilde{M}$ . Instead, we employ the Marlin trick and "opens up" the zero test again. Inside the zero test, the prover and verifier engage in checking whether there exists a quotient polynomial  $q_{2,M}$  such that  $q_{2,M}(r_2)Z_{\mathbb{H}}(r_2) \stackrel{?}{=} S_M(r_1) - \tilde{M}(r_1,r_2)z(r_2)$ . The Marlin trick tells us that  $\tilde{M}(r_1,r_2) = \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1,row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2,col_M(\gamma)) \cdot val_M(\gamma)$ . Again, the prover sends  $\sigma_2$  that claims to be equal to  $\sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1,row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2,col_M(\gamma)) \cdot val_M(\gamma)$  instead of asking the verifier to query it.

To check that  $\sigma_2$  encodes the correct sum over  $\mathbb{K}$  at point  $(r_1, r_2)$ , we use the Marlin trick to send a polynomial  $f_M$  that should satisfy:

$$f_M(X) = \frac{Z_{\mathbb{H}}(r_1) \cdot Z_{\mathbb{H}}(r_2) \cdot val_M(X)}{(r_1 - row_M(X))(r_2 - col_M(X))}.$$

If this is true, then  $\sum_{\gamma \in \mathbb{K}} f(\gamma) = \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma)$ . The prover first proves that  $f_M$  indeed sums to  $\sigma_2$  over  $\mathbb{K}$ , and then proves  $f_M$  is constructed correctly using zero knowledge zero test.

Notice that the verifier makes queries for z and  $z_M$ , and sends the sums  $\sigma_1, \sigma_2$ , for which our zero-knowledge compiler does not apply. We can use bounded independence to blind polynomials z and  $z_M$  such that queries to them are independently and uniformly distributed in  $\mathbb{F}$ . However, the two sums inevitably leak information about the oracles since they contain information of row, col, val evaluted over group K. Our idea is to shift the sums by random values to make them uniformly distributed, i.e, we send  $\beta_1 \cdot \sigma_1, \beta_2 \cdot \sigma_2$  instead of  $\sigma_1, \sigma_2$ , respectively. The prover then needs to additionally provide shift polynomials  $B_1, B_2$  that encode some random values  $\beta_1, \beta_2$ , and the verifier only needs to

- P and V engage in  $\text{ZeroTest}(\mathbb{K}, \mathbb{H} \cup \mathbb{K})$  protocol to prove  $B_2(X) - B_2(\gamma X)$  evaluates to 0 over  $\mathbb{K}$ . – P and V engage in  $\text{ZeroTest}(\mathbb{H}, \mathbb{H} \cup \mathbb{K})$  protocol to prove  $B_2(X) - B_2(\omega X)$  evaluates to 0 over  $\mathbb{H}$ .

- P samples a random polynomial  $r_{sm}(X) \in \mathbb{F}^{<2}[X]$  and sends the shifted masked polynomial  $S'_M(X) = S_M(X) \cdot B_2(X) + r_{sm}(X)Z_{\mathbb{H}}(X)$ .

- P and V engage in  $\text{ZeroTest}(\mathbb{H}, \mathbb{H} \cup \mathbb{K})$  protocol to prove  $S'_M(X) - S_M(X) \cdot B_2(X)$  evaluates to 0 over  $\mathbb{H}$ .

– P computes and sends quotient polynomial  $q_{2,M}$  as follows:

$$q_{2,M}(X) = \frac{\left(S'_M(X) - z'(X) \cdot B'_1(X) \cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(X, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma)\right)}{Z_{\mathbb{H}}(X)}$$

– P computes and sends polynomial  $f_M$  defined by the following evaluations over  $\mathbb{K}$ :

$$\left[f_M(\gamma^i) = \frac{Z_{\mathbb{H}}(r_1) \cdot Z_{\mathbb{H}}(r_2) \cdot val_M(\gamma^i)}{(r_1 - row_M(\gamma^i))(r_2 - col_M(\gamma^i))} \cdot B_2(\gamma^i)\right]_{i=0}^{d-1}$$

. .

 $\begin{array}{l} - \operatorname{\sf V} \text{ queries } q_{2,M}, z', S'_M, B'_1 \text{ on } r_2 \text{ and computes } \sigma_2 = \frac{S'_M(r_2) - q_{2,M}(r_2) \cdot Z_{\mathbb{H}}(r_2)}{z'(r_2) \cdot B'_1(r_2)}.\\ - \operatorname{\sf P} \text{ and } \operatorname{\sf V} \text{ engage in SumCheck}(\mathbb{K}, \mathbb{H} \cup \mathbb{K}) \text{ to prove } f_M \text{ sums to } \sigma_2 \text{ over } \mathbb{K}. \end{array}$ 

- $-\mathsf{P} \text{ and } \mathsf{V} \text{ engage in } \mathsf{ZeroTest}(\mathbb{K}, \mathbb{H} \cup \mathbb{K}) \text{ protocol to prove } (r_1 row_M)(r_2 col_M)f_M Z_{\mathbb{H}}(r_1) \cdot Z_{\mathbb{H}}(r_2) \cdot val_M \cdot B_2 \text{ evaluates to } 0 \text{ over } \mathbb{K}.$

(4) P and V engage in ZeroTest(𝔄,𝔄∪𝔄) protocol to prove z'<sub>A</sub> · z'<sub>B</sub> - z'<sub>C</sub> evaluates to 0 over 𝔄.
(5) To prove correctness of z', P and V engage in ZeroTest(𝔄<sub>x</sub>,𝔄∪𝔄) protocol to prove z' - x evaluates to 0 over 𝔄<sub>x</sub>.

Figure 28: ZK-Marlin: Index-private Marlin polyIOP [CHM<sup>+</sup>20].

know that  $B_1$  is constant over  $\mathbb{H}$  and  $B_2$  is constant over  $\mathbb{H} \cup \mathbb{K}$ . The two main zero tests now change to:

$$\beta_1 \cdot z_M(X) - \sum_{\omega \in \mathbb{H}} \tilde{M}(X, \omega) \cdot z(\omega) \cdot \beta_1 \equiv 0 \mod Z_{\mathbb{H}}(X)$$
$$\beta_2 \cdot S_M(X) - z(X) \cdot \beta_1$$
$$\sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(X, col_M(\gamma)) \cdot val_M(\gamma) \cdot \beta_2 \equiv 0 \mod Z_{\mathbb{H}}(X)$$

**Theorem 20.** ZK-Marlin for  $R_{r1cs}$  (Figure 28) satisfies perfect completeness and for any adversary A against knowledge soundness, we provide an extractor X using  $X_{zt}$ , an extractor for ZeroTest, such that

$$\begin{split} &\mathsf{Adv}^{sound}_{\mathsf{ZK}\text{-}\mathsf{Marlin},\mathsf{R}_{\mathsf{r1cs}},\mathsf{X},\mathcal{A}}(\lambda) \leq 4\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H},\mathbb{H}\cup\mathbb{K},2|\mathbb{H}|+2),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{V},\mathbb{H}\cup\mathbb{K},|\mathbb{H}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{K},\mathbb{H}\cup\mathbb{K},2|\mathbb{H}|+|\mathbb{K}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{H}_{x},\mathbb{H}\cup\mathbb{K},|\mathbb{H}|+1),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \mathsf{Adv}^{sound}_{\mathsf{SumCheck}}(\mathbb{K},\mathbb{H}\cup\mathbb{K},|\mathbb{K}|),\mathsf{R}_{\mathsf{sum}},\mathsf{X}_{\mathsf{sum}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{SumCheck}}(\mathbb{H},\mathbb{H}\cup\mathbb{K},|\mathbb{H}|),\mathsf{R}_{\mathsf{sum}},\mathsf{X}_{\mathsf{sum}},\mathcal{A}}(\lambda) + \frac{6|\mathbb{H}|+6}{|\mathbb{F}|-|\mathbb{H}|-|\mathbb{K}|}. \end{split}$$

*Proof.* <u>Completeness.</u> Suppose that  $Az \circ Bz = Cz$  and z is the extension of vector  $\begin{bmatrix} w \\ x \end{bmatrix}$ . Because we modify polynomials  $z, z_M, B_1$  such that the evaluations do not change over the domain  $\mathbb{H}$  by the property of vanishing polynomials, we can ignore maskings in the completeness analysis. For all  $M \in \{A, B, C\}$ , the honest prover should encode  $z_M$  correctly as the extension of the matrix-vector product Mz, i.e., for all  $\omega \in \mathbb{H}$ ,  $z_M(\omega) = \sum_{\omega' \in \mathbb{H}} M_{\omega,\omega'} \cdot z(\omega')$ . Let  $\tilde{M}$  be the unique extension of M. It must be the case that  $z_M(X) - \sum_{\omega \in \mathbb{H}} M(X, \omega)z(\omega) \equiv 0 \mod Z_{\mathbb{H}}$ . Multiplying both sides by a random point  $\beta_1, z_M(X) \cdot \beta_1 - \sum_{\omega \in \mathbb{H}} M(X, \omega)z(\omega) \cdot \beta_1 \equiv 0 \mod Z_{\mathbb{H}}$  still holds. The honest prover constructs  $B_1(X)$  that evalues to  $\beta_1$  over all  $\mathbb{H}$ , and in particular is constant over  $\mathbb{H}$ , so it will pass check 3(a) by the completeness of zero test. Since  $B_1(\omega) = \beta_1$  for any  $\omega \in \mathbb{H}$  and  $B_1(X) \equiv \beta_1 \mod Z_{\mathbb{H}}(X)$ , we now have  $z_M(X) \cdot B_1(X) - \sum_{\omega \in \mathbb{H}} M(X, \omega)z(\omega) \cdot B_1(\omega) \equiv 0 \mod Z_{\mathbb{H}}$ . Hence, there exists a quotient polynomial such that  $q_{1,M}(X)Z_{\mathbb{H}}(X) = z_M(X)B_1(X) - \sum_{\omega \in \mathbb{H}} M(X, \omega)z(\omega)B_1(\omega)$ , which is true as polynomial equality, not to mention when evalued at a random point  $r_1$ . The verifier derives the sum  $\sigma_1 = z_M(r_1)B_1(r_1) - q_{1,M}(r_1)Z_{\mathbb{H}}(r_1) = \sum_{\omega \in \mathbb{H}} M(r_1, \omega)z(\omega)B_1(\omega)$ , and the prover sends  $S_M$  that encodes each summand at step 3(c). By construction,  $S_M$  sums to  $\sigma_1$  over  $\mathbb{H}$ , so the last check in 3(c) would pass.

Next, the prover wants to prove that  $S_M$  indeed encodes each summand in the summation. In other words,  $\forall \omega \in \mathbb{H}, S_M(\omega) = M(r_1, \omega) z(\omega) B_1(\omega)$ , i.e.,  $S_M(X) - M(r_1, X) z(X) B_1(X) \equiv 0 \mod Z_{\mathbb{H}}(X)$ . By Marlin, we can rewrite the summand as  $\sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(X, col_M(\gamma)) \cdot val_M(\gamma)$ . Multiplying both sides by  $\beta_2$ , we get  $\beta_2 \cdot S_M(X) - \beta_2 \cdot z(X) B_1(X) \cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(X, col_M(\gamma)) \cdot val_M(\gamma) \equiv 0 \mod Z_{\mathbb{H}}(X)$ . The honest prover constructs  $B_2(X)$  that evalutes to  $\beta_2$  over all  $\mathbb{H} \cup \mathbb{K}$ , and in particular is constant over  $\mathbb{H} \cup \mathbb{K}$ , so it will pass checks in 3(d) item 2 by the completeness of zero test. Since  $B_2(\gamma) = \beta_2$  for any  $\gamma \in \mathbb{K}$  and so  $B_2(X) \equiv \beta_2$ mod  $Z_{\mathbb{H}}(X)$ , we now have  $B_2(X)S_M(X) - z(X)B_1(X) \cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(X, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma) \equiv 0 \mod Z_{\mathbb{H}}(X)$ . By the completeness of zero test and by the construction of  $S'_M$ , the zero test in 3(d) item 3 would pass. Hence, there exists a quotient polynomial such that  $q_{2,M}(X)Z_{\mathbb{H}}(X) = S'_M(X) - z(X)B_1(X) \cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(X, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma)$ , which is true as polynomial equality, not to mention when evaluted at a random point  $r_2$ .

The verifier derives the sum  $\sigma_2 = \frac{S'_M(r_2) - q_{2,M}(r_2) \cdot Z_{\mathbb{H}}(r_2)}{z'(r_2) \cdot B_1(r_2)} = \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma)$ . By Marlin, each summand in this sum is equal to  $\frac{Z_{\mathbb{H}}(r_1) \cdot Z_{\mathbb{H}}(r_2) \cdot val_M(\gamma)}{(r_1 - row_M(\gamma))(r_2 - col_M(\gamma))} \cdot B_2(\gamma)$ . The prover sends  $f_M$  which encodes each summand over  $\mathbb{K}$ , so by construction it sums to  $\sigma_2$  over  $\mathbb{K}$ , and the sum check in 3(d) item 7 should pass. The last check in 3(d) will also pass by the definition of  $f_M(X)$ .

Finally, steps 4 and 5 are same as in Marlin, by the completeness of zero tests, these two checks should pass.

**Lemma 21** ([Tha22]). Let  $\mathbb{F}$  be a field and  $\mathbb{H} \subseteq \mathbb{F}$  be a subgroup. For  $d \ge |\mathbb{H}|$ , a degree-d univariate polynomial f over

 $\mathbb{H}$  vanishes on  $\mathbb{H}$  if and only if the vanishing polynomial  $Z_{\mathbb{H}}(X) \coloneqq \prod_{\omega \in \mathbb{H}} (X - \omega)$  divides f, i.e., if and only if there exists a polynomial q with  $\deg(q) \leq d - |\mathbb{H}|$  such that  $f = Z_{\mathbb{H}} \cdot q$ .

*Proof.* ( $\Rightarrow$ ) Since f vanishes over  $\mathbb{H}$ ,  $f(\omega) = 0, \forall \omega \in \mathbb{H}$ . Hence,  $X - \omega$  divides f, for all  $\omega \in \mathbb{H}$ , i.e.,  $Z_{\mathbb{H}}|f$ .

(⇐) If  $f = Z_{\mathbb{H}} \cdot q$ , then by vanishing polynomial,  $f(\omega) = Z_{\mathbb{H}}(\omega) \cdot q(\omega) = 0$ ,  $\forall \omega \in \mathbb{H}$ . Hence, f indeed vanishes over  $\mathbb{H}$ .

<u>Knowledge Soundness</u>. We bound the advantage of adversary  $\mathcal{A}$  by bounding the advantage of each of a series of game hops. We define  $G_0 = \text{SOUND}_{\mathsf{ZK-Marlin},\mathsf{R}_{\mathsf{rlcs}},\mathsf{X}}^{\mathcal{A}}(\lambda)$ . The inequality above follows from the following claims that we will justify:

(1)

$$\begin{split} |\Pr[G_0 = 1] - \Pr[G_1 = 1]| &\leq 2\mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{H}, \mathbb{H} \cup \mathbb{K}, |\mathbb{H}| + 2), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &\quad + \mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{V}, \mathbb{H} \cup \mathbb{K}, |\mathbb{H}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(2)  $|\Pr[G_1 = 1] - \Pr[G_2 = 1]| \le \frac{3|\mathbb{H}| + 3}{|\mathbb{F}| - |\mathbb{H}| - |\mathbb{K}|}$ (3)

$$|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \leq \frac{3|\mathbb{H}| + 3}{|\mathbb{F}| - |\mathbb{H}| - |\mathbb{K}|} + \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{H}, \mathbb{H} \cup \mathbb{K}, 2|\mathbb{H}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$$

(4)

$$|\Pr[G_3 = 1] - \Pr[G_4 = 1]| \leq \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{K}, \mathbb{H} \cup \mathbb{K}, 2|\mathbb{H}| + |\mathbb{K}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$$

(5)

$$\Pr[G_4 = 1] - \Pr[G_5 = 1]| \leq \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{SumCheck}(\mathbb{K}, \mathbb{H} \cup \mathbb{K}, |\mathbb{K}|), \mathsf{R}_{\mathsf{sum}}, \mathsf{X}_{\mathsf{sum}}, \mathcal{A}}(\lambda)$$

(6)

$$\Pr[G_5 = 1] - \Pr[G_6 = 1]| \le \mathsf{Adv}^{\mathsf{sound}}_{\mathsf{SumCheck}(\mathbb{H},\mathbb{H}\cup\mathbb{K},|\mathbb{H}|),\mathsf{R}_{\mathsf{sum}},\mathsf{X}_{\mathsf{sum}},\mathcal{A}}(\lambda)$$

(7)

$$\begin{split} |\Pr[G_6 = 1] - \Pr[G_7 = 1]| &\leq \mathsf{Adv}_{\mathsf{zero}(\mathbb{H}, \mathbb{H} \cup \mathbb{K}, 2|\mathbb{H}| + 2), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{zero}(\mathbb{H}_x, \mathbb{H} \cup \mathbb{K}, |\mathbb{H}| + 1), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(8)  $\Pr[G_7 = 1] = 0$ 

Claim 1 argues for the well-formedness of shift polynomials  $B'_1$  and  $B_2$ . Claim 2 reduces the sum check into a zero test for verifying the linear relation. Claim 3 unrolled yet another sum check for matrix polynomials into a zero test to argues for the well-formedness of the claimed sum. Claim 4 argues for the well-formedness of polynomial  $f_M$  that encodes the second sum. Claim 5 argues the well-formedness of the second claimed sum by checking that  $f_M$  indeed sums to it. Claim 6 argues the well-formedness of the first claimed sum. Finally, Claim 7 performs the R1CS check and instance check, and Claim 8 completes with an extractor that extracts the witness.

Recall that proving R1CS relation reduces to proving three relations described in Item 1-3. Item 1 and Item 3 are proven in steps (4) and (5) as in Marlin. The only difference is that the oracles are masked by random linear polynomials. Masking does not change the evaluations of  $z, z_M$  over  $\mathbb{H}$  since random polynomial times  $Z_{\mathbb{H}}$  evaluates to 0 over  $\mathbb{H}$ . By the soundness of zero tests, Item 1 and Item 3 are satisfied except with small soundness error. Notice that to make three relations consistent, they should be proven with respect to the same maked polynomials  $z', z'_M$ . We are left with proving the soundness for Item 2.

*Claim 1:* For the first step, we show that polynomial  $B_1$  encodes some random element  $\beta_1$  over  $\mathbb{H}$ , and similarly, polynomial  $B_2$  encodes  $\beta_2$  over  $\mathbb{H} \cup \mathbb{K}$ .

- $B'_1(X) B'_1(\omega X) = 0$  over  $\mathbb{H}$  with advantage  $\operatorname{Adv}^{\operatorname{sound}}_{\operatorname{ZeroTest}(\mathbb{H},\mathbb{H}\cup\mathbb{K},|\mathbb{H}|+2),\operatorname{R}_{\operatorname{Zero}},\operatorname{X}_{\operatorname{zt}},\mathcal{A}}(\lambda)$ : Since masking does not change the evaluation of  $B_1$  over  $\mathbb{H}$ , it checks that  $B_1$  is constant over  $\mathbb{H}$ , and encodes some unknown random element  $\beta_1$ .
- B<sub>2</sub>(X) − B<sub>2</sub>(ωX) = 0 over ℍ with advantage Adv<sup>sound</sup><sub>ZeroTest(ℍ,ℍ∪K,|ℍ|),R<sub>zero</sub>,x<sub>zt,A</sub>(λ): Similarly checks that B<sub>2</sub> encodes some β<sub>2</sub> over ℍ.
  </sub>
- B<sub>2</sub>(X) B<sub>2</sub>(γX) = 0 over K with advantage Adv<sup>sound</sup><sub>ZeroTest(V,H∪K,|H|),R<sub>zero</sub>,X<sub>zt</sub>,A(λ): Checks that B<sub>2</sub> encodes some β'<sub>2</sub> over K. Since H ∩ K ≠ Ø, it must be that β<sub>2</sub> = β'<sub>2</sub>.
  </sub>

 $G_1$  employs the zero test extractor  $X_{zt}$  to check the above tests and aborts if the extractor fails. The claimed probability bound follows from a series of hybrids bounding each hybrid by the soundness advantages for the zero tests as claimed above.

Claim 2: In the second step, we reduce Item 2 to verify the sum from the corresponding zero test. Recall that we want to show, for  $M \in \{A, B, C\}$ ,  $\forall \omega \in \mathbb{H}, z'_M(\omega) = \sum_{\omega' \in \mathbb{H}} M_{\omega,\omega'} \cdot z'(\omega')$ . Let  $\tilde{M}(X,Y)$  be the unique bivariate polynomial of degree at most  $|\mathbb{H}|$  in each variable that extends matrix M. We can instead show,  $\forall \omega \in \mathbb{H}, z'_M(\omega) = \sum_{\omega' \in \mathbb{H}} \tilde{M}(\omega, \omega') \cdot z'(\omega')$ , i.e.,  $z'_M(X) - \sum_{\omega \in \mathbb{H}} \tilde{M}(X, \omega) \cdot z'(\omega) \equiv 0 \mod Z_{\mathbb{H}}(X)$ .

If we multiply both sides by some random element  $\beta_1$ , we still have  $\beta_1 \cdot z'_M(X) - \beta_1 \cdot \sum_{\omega \in \mathbb{H}} \tilde{M}(X, \omega) \cdot z'(\omega) \equiv 0 \mod Z_{\mathbb{H}}(X)$ . Since  $B'_1(X) \equiv \beta_1 \mod Z_{\mathbb{H}}(X)$  and  $B'_1(\omega) = \beta_1, \forall \omega \in \mathbb{H}$ , it is equivalent to  $B'_1(X) \cdot z'_M(X) - \sum_{\omega \in \mathbb{H}} \tilde{M}(X, \omega) \cdot z'(\omega) \cdot B'_1(\omega) \equiv 0 \mod Z_{\mathbb{H}}(X)$ . By Lemma 21, it is equivalent to the existence of a polynomial  $q_{1,M}$  such that

$$z'_{M}(X) \cdot B'_{1}(X) - \sum_{\omega \in \mathbb{H}} \tilde{M}(X,\omega) \cdot z'(\omega) \cdot B'_{1}(\omega) = q_{1,M}(X) \cdot Z_{\mathbb{H}}(X)$$
(3)

The verifier probabilistically checks that Equation 3 holds by choosing a random  $r_1 \in \mathbb{F}$  and confirming that

$$z'_M(r_1) \cdot B'_1(r_1) - \sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) \cdot z'(\omega) \cdot B'_1(\omega) = q_{1,M}(r_1) \cdot Z_{\mathbb{H}}(r_1)$$

except with probability at most  $\frac{3|\mathbb{H}|+3}{|\mathbb{F}|-|\mathbb{H}|-|\mathbb{K}|}$ 

Claim 3: The verifier can query  $z'_M, B'_1, q_{1,M}$  at  $r_1$ , but it cannot compute  $\sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) \cdot z'(\omega) \cdot B'_1(\omega)$  by itself. Instead, the prover sends  $\sigma_1$  that claims to be the sum, and eventually tries to prove  $\sigma_1 \stackrel{?}{=} \sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) \cdot z'(\omega) \cdot B'_1(\omega)$ .

In this step, we first prove a polynomial  $S_M$  encodes each summand, i.e., has evalutions  $[\tilde{M}(r_1,\omega) \cdot z'(\omega) \cdot B'_1(\omega)]_{\omega \in \mathbb{H}}$  over  $\mathbb{H}$ , i.e.,  $S_M(X) - \tilde{M}(r_1, X) \cdot z'(X) \cdot B'_1(X) \equiv 0 \mod Z_{\mathbb{H}}(X)$ . If we multiply both sides by some random element  $\beta_2$ , we still have  $\beta_2 \cdot S_M(X) - \beta_2 \cdot \tilde{M}(r_1, X) \cdot z'(X) \cdot B'_1(X) \equiv 0 \mod Z_{\mathbb{H}}(X)$ . By Marlin,  $\tilde{M}(X, Y) = \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(X, row_M(\gamma)) \cdot u_{\mathbb{H}}(Y, col_M(\gamma)) \cdot val_M(\gamma)$ , so it is equivalent to  $\beta_2 \cdot S_M(X) - z'(X) \cdot B'_1(X) \cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(X, col_M(\gamma)) \cdot val_M(\gamma) \cdot \beta_2 \equiv 0 \mod Z_{\mathbb{H}}(X)$ .

•  $S'_{M}(X) - S_{M}(X) \cdot B_{2}(X) = 0$  over  $\mathbb{H}$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}, \mathbb{H} \cup \mathbb{K}, 2|\mathbb{H}|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks  $S'_{M}(X) - S_{M}(X) \cdot B_{2}(X) \equiv S'_{M}(X) - S_{M}(X) \cdot \beta_{2} \mod Z_{\mathbb{H}}(X)$  for some  $\beta_{2}$ .

Because of the above and because  $B_2(\gamma) = \beta_2, \forall \gamma \in \mathbb{K}$ , we can equivalently check that

$$S'_{M}(X) - z'(X) \cdot B'_{1}(X)$$
  
 
$$\cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_{1}, row_{M}(\gamma)) \cdot u_{\mathbb{H}}(X, col_{M}(\gamma)) \cdot val_{M}(\gamma) \cdot B_{2}(\gamma) \equiv 0 \mod Z_{\mathbb{H}}(X)$$

By Lemma 21, this is equivalent to the existence of a polynomial  $q_{2,M}$  such that

$$S'_{M}(X) - z'(X) \cdot B'_{1}(X) \cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_{1}, row_{M}(\gamma)) \cdot u_{\mathbb{H}}(X, col_{M}(\gamma)) \cdot val_{M}(\gamma) \cdot B_{2}(\gamma) = q_{2,M}(X) \cdot Z_{\mathbb{H}}(X)$$

$$(4)$$

The verifier probabilistically checks that Equation 3 holds by choosing a random  $r_2 \in \mathbb{F}$  and confirming that

$$S'_{M}(r_{2}) - z'(r_{2}) \cdot B'_{1}(r_{2})$$
  
 
$$\cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_{1}, row_{M}(\gamma)) \cdot u_{\mathbb{H}}(r_{2}, col_{M}(\gamma)) \cdot val_{M}(\gamma) \cdot B_{2}(\gamma) = q_{2,M}(r_{2}) \cdot Z_{\mathbb{H}}(r_{2})$$

except with probability at most  $\frac{3|\mathbb{H}|+3}{|\mathbb{F}|-|\mathbb{H}|-|\mathbb{K}|}$  by Schwartz-Zippel lemma.

 $G_3$  employs the zero test extractor  $X_{zt}$  to check the zero test for  $S'_M$  and aborts if the extractor fails. The claimed probability bound follows from a series of hybrids bounding each hybrid by the soundness advantages for the zero tests as claimed above.

Claim 4: The verifier can query  $S'_M, z', B'_1, q_{2,M}$  at  $r_2$ , but it cannot compute  $\sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma)$  by itself. Instead, the prover sends  $\sigma_2$  that claims to be the sum, and eventually tries to prove  $\sigma_2 \stackrel{?}{=} \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma).$ 

From this step, the proof is the same as that of Marlin. The prover sends a polynomial  $f_M$  that encodes the summand over  $\mathbb{K}$ .

•  $(r_1 - row_M)(r_2 - col_M)f_M - Z_{\mathbb{H}}(r_1) \cdot Z_{\mathbb{H}}(r_2) \cdot val_M \cdot B_2$  evaluates to 0 over  $\mathbb{K}$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}(\mathbb{K},\mathbb{H}\cup\mathbb{K},2|\mathbb{H}|+|\mathbb{K}|), \mathsf{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{Zero}}$ Checks that  $\forall \gamma \in \mathbb{K}, f_M(\gamma) = u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma) = \frac{Z_{\mathbb{H}}(r_1) \cdot Z_{\mathbb{H}}(r_2) \cdot val_M(\gamma)}{(r_1 - row_M(\gamma))(r_2 - col_M(\gamma))} \cdot B_2(\gamma)$ . The second equality is by Marlin.

 $G_4$  employs the zero test extractor  $X_{zt}$  to check the above tests and aborts if the extractor fails. The claimed probability bound follows from a series of hybrids bounding each hybrid by the soundness advantages for the zero tests as claimed above.

Claim 5: In this step, we reduce the check  $\sigma_2 \stackrel{?}{=} \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma)$  to a sumcheck over its summands.

•  $\sum_{\gamma \in \mathbb{K}} f_M(\gamma) = \sigma_2$  with advantage  $\operatorname{Adv}_{\operatorname{SumCheck}(\mathbb{K}, \mathbb{H} \cup \mathbb{K}, |\mathbb{K}|), \operatorname{R}_{\operatorname{sum}}, \operatorname{X}_{\operatorname{sum}}, \mathcal{A}}(\lambda)$ : Checks that  $\sigma_2 = \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma)$ .

 $G_5$  employs the sum check extractor  $X_{sum}$  to check the above tests and aborts if the extractor fails. The claimed probability bound follows from a series of hybrids bounding each hybrid by the soundness advantages for the sum check as claimed above.

*Claim 6:* We will go back to the check  $\sigma_1 \stackrel{?}{=} \sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) \cdot z'(\omega) \cdot B'_1(\omega)$  and reduce it to a sumcheck for a polynomial that encodes each summand.

•  $\sum_{\omega \in \mathbb{H}} S_M(\omega) = \sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) \cdot z'(\omega) \cdot B'_1(\omega) = \sigma_1$  with advantage  $\operatorname{Adv}_{\operatorname{SumCheck}(\mathbb{H},\mathbb{H}\cup\mathbb{K},|\mathbb{H}|),\operatorname{R}_{\operatorname{sum}},\mathsf{X}_{\operatorname{sum}},\mathcal{A}}(\lambda)$ : this checks  $\sigma_1$  encodes the correct value evaluated at  $r_1$  assuming  $S_M$  encodes the summand correctly.

 $G_6$  employs the sum check extractor  $X_{sum}$  to check the above tests and aborts if the extractor fails. The claimed probability bound follows from a series of hybrids bounding each hybrid by the soundness advantages for the sum check as claimed above.

*Claim 7:* In this claim,  $G_7$  employs the zero test extractor  $X_{zt}$  to check Item 1 and Item 3 in steps (4) and (5).

Claim 8: Finally, we construct our extractor X that always succeeds on a verifying prover. X employs  $X_{zt}$  to retrieve  $z'_M, z'$ . Given z', X retrives and outputs w and x. By Claim 2, in  $G_2$ , if the verifier succeeds,  $X_{zt}$  always succeeds and so our extractor will always succeed.

**Theorem 22.** The compiled polyIOP using the compiler in Figure 19 of ZK-Marlin for  $R_{r1cs}$  (Figure 28) is index-hiding honest-verifier zero-knowledge.

*Proof.* By zero-knowledge compiler, our simulator can run ZK-ZeroTest simulator and SumCheck simulator and hence HVZK are preserved within these tests. Consider the view of an honest execution of the protocol outside the tests, V queries  $z'_M, B'_1, q_{1,M}$  at  $r_1$ ; queries  $z', S'_M, B'_1, q_{2,M}$  at  $r_2$ . Expanding,

$$\begin{aligned} z'_{M}(r_{1}) &= z_{M}(r_{1}) + r_{zm}(r_{1})Z_{\mathbb{H}}(r_{1}) \\ B'_{1}(r_{1}) &= B'_{1}(r_{1}) + r_{b1}(r_{1})Z_{\mathbb{H}}(r_{1}) \\ z'(r_{2}) &= z(r_{2}) + r_{z}(r_{2})Z_{\mathbb{H}}(r_{2}) \\ S'_{M}(r_{2}) &= S_{M}(r_{2})B_{2}(r_{2}) + r_{sm}(r_{2})Z_{\mathbb{H}}(r_{2}) \\ B'_{1}(r_{2}) &= B'_{1}(r_{2}) + r_{b1}(r_{2})Z_{\mathbb{H}}(r_{2}) \end{aligned}$$

Since  $r_1, r_2 \in \mathbb{F} \setminus (\mathbb{H} \cup \mathbb{K}), Z_{\mathbb{H}}(r_1) \neq 0 \neq Z_{\mathbb{H}}(r_2)$ . Since  $r_{zm}, r_z, r_{sm}$  are random linear polynomial,  $r_{zm}(r_1)Z_{\mathbb{H}}(r_1), r_z(r_2)Z_{\mathbb{H}}(r_2), r_{sm}(r_2)Z_{\mathbb{H}}(r_2)$  are distributed independently and uniformly at random and so do  $z'_M(r_1), z'(r_2), S'_M(r_2)$ . Also since  $r_{b1}$  is degree-two random polynomial, for two queries at  $r_1, r_2$  that are distinct,  $r_{b1}(r_1)Z_{\mathbb{H}}(r_1), r_{b1}(r_2)Z_{\mathbb{H}}(r_2)$  are distributed independently and so do  $B'_1(r_1), B'_1(r_2)$ .

We are left to prove that  $q_{1,M}(r_1)$  and  $q_{2,M}(r_2)$  are distributed uniformly random. Notice that they satisfy the relations

$$q_{1,M}(r_1) = \frac{z'_M(r_1) \cdot B'_1(r_1) - z'_M(r_1)}{Z_{\mathbb{H}}(r_1)}$$
$$q_{2,M}(r_2) = \frac{S'_M(r_2) - \sigma_2 \cdot z'(r_2) \cdot B'_1(r_2)}{Z_{\mathbb{H}}(r_2)}$$

Recall from the soundness proof,

$$\begin{split} \sigma_1 &= \sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) \cdot z'(\omega) \cdot B'_1(\omega) \\ &= \beta_1 \cdot \sum_{\omega \in \mathbb{H}} \tilde{M}(r_1, \omega) \cdot z'(\omega), \\ \sigma_2 &= \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma) \cdot B_2(\gamma) \\ &= \beta_2 \cdot \sum_{\gamma \in \mathbb{K}} u_{\mathbb{H}}(r_1, row_M(\gamma)) \cdot u_{\mathbb{H}}(r_2, col_M(\gamma)) \cdot val_M(\gamma) \end{split}$$

Since  $\beta_1, \beta_2$  are random elements in  $\mathbb{F}$  and other elements in the relations are also random by the above argument, so do  $q_{1,M}(r_1)$  and  $q_{2,M}(r_2)$ .

**Theorem 23.** The compiled polyIOP using the compiler in Figure 19 and using ZK-Marlin of Mux-Marlin for  $R_{MExe,n}[R_{r1cs}]$  is honest-verifier zero-knowledge.

*Proof.* Our Mux-Marlin construction in Figure 26 partially reduces to ZeroTest with two places that make oracle queries outside of ZeroTest. The first place is in the Marlin construction in Figure 24. While we need the table polynomials  $row_M, col_M, val_M$  to be private, Marlin leaks oracle information at step 4(d) when V queries  $q_M, z_M, z$ . We present a fix to build a index-hiding Marlin in Figure 28. The second place is in the step 4 of Mux-Marlin construction in Figure 26 when V queries f. We argue that we do not worry zero knowledge here since we are trying to prove f is a public vanishing polynomial that is irrelevant to witness as in Theorem 9. The index-hiding HVZK then follows from zero knowledge compiler (Figure 19) and its underlying ZK-Marlin when we set  $\mathbb{H} = \mathbb{K}$ .

**Completeness and knowledge soundness.** Lastly, we prove the completeness and knowledge soundness of Mux-Marlin in the following theorem.

**Theorem 24.** Mux-Marlin for  $R_{MExe,n}[R_{r1cs}]$  (Figure 26) satisfies perfect completeness and for any adversary A against knowledge soundness, we provide an extractor X using  $X_{zt}$ , an extractor for ZeroTest, using  $X_{lk}$ , an extractor for 9-TuPlookup, and using  $X_{marlin}$ , an extractor for Marlin such that

$$\begin{split} &\mathsf{Adv}^{sound}_{\mathsf{Mux-Marlin},\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{r1cs}}],\mathsf{X},19,\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{V}_{in},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{V}_{out},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 5\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{G}_{n},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}_{n}|+1),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{G}_{n}\setminus\mathbb{V}_{in},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 5\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{G},\mathbb{H}\cup\mathbb{G},3|\mathbb{G}|+|\mathbb{V}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}\cup\mathbb{G},|\mathbb{G}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \mathsf{Adv}^{sound}_{9-\mathsf{TuPlookup},\mathsf{R}_{9-\mathsf{vlkup}},\mathsf{X}_{\mathsf{lk}},8,\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{Marlin}(\mathbb{G}),\mathsf{R}_{\mathsf{r1cs}},\mathsf{X}_{\mathsf{marlin}},\mathcal{A}}(\lambda) + \frac{|\mathbb{G}_{in}|}{|\mathbb{F}| - |\mathbb{G}| - |\mathbb{H}|} \end{split}$$

#### Proof of Theorem 24.

Proof. We consider each of completeness and soundness separately.

<u>Completeness</u>. The success of steps (1-4) follow directly from the completeness of the underlying polyIOP subprotocols and the polynomial constructions of a valid prover. All that remains to show is the success of execution of the Marlin polyIOP in step (5). The Marlin index polynomials  $row_M$ ,  $col_M$ ,  $val_M$  created by the prover are for  $M \in \{A, B, C\}$  such that:

$$\forall i, j \in [n] \quad M(\mu^{i} \mathbb{V}, \mu^{j} \mathbb{V}) = \begin{cases} M_{inst_{j}} & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases}$$
(5)

Since x evaluated on each coset  $x(\mu^{j}\mathbb{V}) = [inst_{j}, mem_{j}, inst_{j+1}, mem_{j+1}, w_{j}]$ , we have that for a valid prover

 $A_{\mathrm{inst}_{j}}\left[ x(\mu^{j}\mathbb{V}) \right] \circ B_{\mathrm{inst}_{j}}\left[ x(\mu^{j}\mathbb{V}) \right] = C_{\mathrm{inst}_{j}}\left[ x(\mu^{j}\mathbb{V}) \right].$ 

And thus by construction of A, B, C, we have

$$A[x(\mathbb{G})] \circ B[x(\mathbb{G})] = C[x(\mathbb{G})].$$

Finally, since this is the R1CS that is being checked by step (5), we have that the prover will succeed by the completeness of the Marlin polyIOP.

<u>Knowledge soundness.</u> We bound the advantage of adversary A by bounding the advantage of each of a series of game hops. We define  $G_0 =$ 

SOUND<sup>A</sup><sub>Mux-Marlin,R<sub>MExe,n</sub>[R<sub>r1cs</sub>], $X(\lambda)$ . The inequality above follows from the following claims that we will justify: (1)</sub>

$$\begin{split} |\Pr[\mathbf{G}_{0}=1] - \Pr[\mathbf{G}_{1}=1]| &\leq \mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{V}_{in}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{Zero}}, \mathsf{x}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{V}_{out}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{Zero}}, \mathsf{x}_{\mathsf{zt}}, \mathcal{A}}(\lambda) + 5\mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{G}_{n}, \mathbb{H} \cup \mathbb{G}, 2|\mathbb{G}_{n}|+1), \mathsf{R}_{\mathsf{Zero}}, \mathsf{x}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathsf{sound}}(\mathbb{G}_{n} \setminus \mathbb{V}_{in}, \mathbb{H} \cup \mathbb{G}, 2|\mathbb{G}|), \mathsf{R}_{\mathsf{Zero}}, \mathsf{x}_{\mathsf{zt}}, \mathcal{A}}(\lambda) + \frac{|\mathbb{G}_{in}|}{|\mathbb{F}| - |\mathbb{G}| - |\mathbb{H}|} \end{split}$$

(2)

$$\begin{split} |\Pr[G_1 = 1] - \Pr[G_2 = 1]| &\leq \mathsf{5Adv}_{\mathsf{ZeroTest}(\mathbb{G}, \mathbb{H} \cup \mathbb{G}, 3|\mathbb{G}| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}, \mathcal{A}}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}(\mathbb{V}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}, \mathcal{A}}}(\lambda) + \mathsf{Adv}_{9-\mathsf{TuPlookup}, \mathsf{R}_{9-\mathsf{ykup}}, \mathsf{X}_{\mathsf{lk}}, \mathsf{8}, \mathcal{A}}(\lambda) \end{split}$$

 $(3) \quad |\Pr[G_2 = 1] - \Pr[G_3 = 1]| \leq \mathsf{Adv}^{sound}_{\mathsf{Marlin}(\mathbb{G}),\mathsf{R}_{\mathsf{rlcs}},\mathsf{X}_{\mathsf{marlin}},\mathcal{A}}(\lambda)$ 

(4)  $\Pr[G_3 = 1] = 0$ 

Claim 1 argues for the well-formedness of the statement polynomial x. Claim 2 argues for the well-formedness of the index polynomials  $row_M$ ,  $col_M$ ,  $val_M$  to invoke Marlin. Claim 3 argues that the R1CS for the unrolled execution is satisfied. Lastly Claim 4 argues that the constructed extractor always succeeds for an accepting verifier.

Claim 1: Our first step is the same as Claim 1 of Theorem 8.

*Claim 2:* In this second step, we argue for the well-formedness of the shifted polynomials  $row_M$ ,  $col_M$  and normalized polynomials  $val_M$  and that they correctly match the desired polynomials output by the index algorithm for the unrolled computation.

First consider the tuple lookup in step (1).  $G_2$  employs the extractor  $X_{\text{TuPlookup}}$  to check the valid lookup relation between  $row'_M$ ,  $col'_M$ ,  $val'_M$  and the table polynomials  $trow_M$ ,  $tcol_M$ ,  $tval_M$ , aborting if the extractor fails. The probability of the bad flag being set is bounded by the soundness advantage of the tuple lookup protocol  $Adv_{9\text{-TuPlookup},R_{9\text{-vikup}},X_{\text{lk}},8,\mathcal{A}}(\lambda)$ 

Next consider the steps to show the well-formedness of the shift polynomial s in step (2a):

- $L_{1,\mathbb{G}}(X)(s(X)-1) = 0$  over  $\mathbb{G}$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\mathbb{G},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda)$ : Checks base case s(1) = 1.
- $s(X) = s(\gamma X)$  over  $\mathbb{V}$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}}(\mathbb{V}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda)$ : Checks  $s(\mathbb{V}) = 1$ .
- $(\mu \cdot s(X) s(\mu X))(Z_{\mu^{n-1}\mathbb{V}}(X)) = 0$  over  $\mathbb{G}$  with advantage  $\operatorname{Adv}^{\operatorname{sound}}_{\operatorname{ZeroTest}}(\mathbb{G}, \mathbb{H} \cup \mathbb{G}, |\mathbb{G}| + |\mathbb{V}|), \mathbb{R}_{\operatorname{Zero}}, X_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks inductive step:

$$\left[s(\mu^j \mathbb{V}) = \mu^j\right]_{j=1}^{n-1}.$$

And then the lookup polynomials  $row'_M$  and  $col'_M$  are shifted to construct  $row_M$  and  $col_M$  that represent valid index polynomials for the matrix M described above in Equation 5. The indices for the submatrices are some set of instruction indices  $[inst'_j]_{j=0}^{n-1}$  determined by the lookup protocol.

- $row_M(X) = row'_M(X)s(X)$  over  $\mathbb{G}$  with advantage  $\operatorname{Adv}^{\operatorname{sound}}_{\operatorname{ZeroTest}(\mathbb{G},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}|),\operatorname{R}_{\operatorname{Zero}},\operatorname{X}_{\operatorname{zt}},\mathcal{A}}(\lambda)$ .
- $col_M(X) = col'_M(X)s(X)$  over  $\mathbb{G}$  with advantage  $\operatorname{Adv}^{\operatorname{sound}}_{\operatorname{ZeroTest}(\mathbb{G},\mathbb{H}\cup\mathbb{G},2|\mathbb{G}|),\operatorname{R}_{\operatorname{Zero}},\operatorname{X}_{\operatorname{zt}},\mathcal{A}}(\lambda)$ .

Lastly, the lookup polynomial  $val'_M$  is normalized to construct  $val_M$  representing the valid index polynomial for M:

•  $|\mathbb{G}|^2 \cdot val_M(X) = val'_M(X) \cdot row_M(X) \cdot col_M(X)$  over  $\mathbb{G}$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}(\mathbb{G}, \mathbb{H} \cup \mathbb{G}, 3|\mathbb{G}|), \operatorname{R}_{\operatorname{Zero}}, \operatorname{X}_{\operatorname{zt}, \mathcal{A}}}(\lambda)$ .

As before  $G_2$  employs the zero test extractor  $X_{zt}$  to check the above tests and aborts if the extractor fails. The probability bound comes from a series of hybrids bounding each hybrid by the soundness advantage for the zero test.

*Claim 3:* Now we argue that the statement polynomial satisfies the unrolled execution R1CS. From Claim 2, we have that  $row_M$ ,  $col_M$ ,  $val_M$  represent valid index polynomials for matrices  $M \in A, B, C$  as described in Equation 5. G<sub>3</sub> employs the Marlin extractor X<sub>marlin</sub> to check the R1CS relation proved by the Marlin polyIOP in step (5), and aborts if the extractor fails.

• Marlin for  $row_M$ ,  $col_M$ ,  $val_M$  and statement x with advantage  $Adv_{Marlin(\mathbb{G}), R_{rlcs}, X_{marlin}, \mathcal{A}}^{sound}(\lambda)$ .

*Claim 4:* We conclude by constructing an extractor X that always succeeds on a verifying prover. As argued in the completeness proof, if the R1CS for A, B, C is satisfied, then for each coset along the diagonal that gives us:

$$A_{\text{inst}'_{i}}\left[x(\mu^{j}\mathbb{V})\right] \circ B_{\text{inst}'_{i}}\left[x(\mu^{j}\mathbb{V})\right] = C_{\text{inst}'_{i}}\left[x(\mu^{j}\mathbb{V})\right]$$

where x evaluated on each coset  $x(\mu^{j}\mathbb{V}) = [inst_{j}, mem_{j}, inst_{j+1}, mem_{j+1}, w_{j}]$ . By assumption, we have that the R1CS for each instruction stored in the table polynomials enforce that  $inst_{j}$  is correct, we have that  $inst_{j} = inst'_{j}$  for all j. Thus, our extractor X simply employs the Marlin extractor X<sub>marlin</sub> to extract x and outputs,

$$\left(\left[\operatorname{inst}_{j},\operatorname{mem}_{j},\operatorname{w}_{j}\right]_{j=0}^{n}, x_{0}=\left[\operatorname{inst}_{0},\operatorname{mem}_{0}\right], x_{n}=\left[\operatorname{inst}_{n},\operatorname{mem}_{n}\right]\right).$$

By Claim 3, in  $G_3$ , if the verifier succeeds, the Marlin extractor always succeeds and so similarly, so will our constructed extractor.

# F Plookup-based Succinct Vector Lookup

In this section, we present TuPlookup, a Plookup-based [GW20] univariate vector lookup protocol defined by the following relation:

$$\mathsf{R}_{\mathsf{vlkup}} = \left\{ \bot, (\llbracket f \rrbracket, \llbracket t \rrbracket), (f, t) : \left\{ (f(\psi^i \mathbb{V})) \right\}_{i \in [d_0]} \subseteq \left\{ (t(\mu^i \mathbb{V})) \right\}_{i \in [d_1]} \right\}.$$

The relation checks that every coset of  $\mathbb{V}$  in f over  $\mathbb{H}_0 = \langle \psi^i \mathbb{V} \rangle_{i \in [d_0]}$  exists as a coset of  $\mathbb{V}$  in t over  $\mathbb{H}_1 = \langle \mu^i \mathbb{V} \rangle_{i \in [d_1]}$ . In other words, the set of coset vectors in f is a subset of the set of coset vectors in t.

We build up to the argument through a series of steps. First, we construct a vector permutation argument TuPerm that shows the coset vectors of two polynomials are permutations of each other. We generalize this argument to work across k pairs of polynomials in k-TuPerm. Here we show that the each pair of polynomials are equivalent with respect to the same permutation. We then further extend this to k-XGTuPerm which allows us to perform a permutation argument across coset vectors for polynomials even when the coset vectors are evaluated over different groups on the different polynomials. This will be important to build our desired vector lookup argument where the lookup polynomial f and the table polynomial t may be of different sizes and are defined over different groups  $\mathbb{H}_0, \mathbb{H}_1$ .

#### F.1 Vector Permutation

We start with the task of performing a permutation check for vectors defined over coset evaluations of two polynomials f, g. It is given by the relation:

$$\mathsf{R}_{\mathsf{tp}} = \left\{ \bot, (\llbracket f \rrbracket, \llbracket g \rrbracket), (f, g) : \{\!\!\{(f(\omega^i \mathbb{V}))\}\!\!\}_{i \in [n]} = \{\!\!\{(g(\omega^i \mathbb{V}))\}\!\!\}_{i \in [n]} \right\}$$

Our protocol builds on the permutation argument checking permutations of fields elements encoded in polynomials presented by [GWC19]; it checks  $\{\{(f(\omega^i))\}\}_{\omega^i \in \mathbb{H}} = \{\{(g(\omega^i))\}\}_{\omega^i \in \mathbb{H}}\}$ . We extend their approach of constructing and checking equality of a multiset hash to the vector setting: constructing a multiset hash where each element in the set is a vector (or looking forward, a hash encoding of a vector).

To perform the multiset hash comparison, we construct two helper polynomials for each of f, g. Without loss of generality, polynomial  $B_f$  is constructed to accumulate a hash over each coset of f. Polynomial  $S_f$  is constructed to zero-out  $B_f$  with the claimed final hash summation so the accumulation induction of  $B_f$  holds over the coset (see description of protocol in Section 2.2). The full details are provided in Figure 29.

We provide the following theorem for the completeness and knowledge soundness of our vector permutation argument.

**Theorem 25.** TuPerm for  $R_{tp}$  (Figure 29) satisfies perfect completeness and for any adversary A against knowledge soundness, we provide an extractor X using  $X_{zt}$ , an extractor for ZeroTest, and using  $X_{prod}$ , an extractor for ProductCheck such that

$$\begin{split} & \mathsf{Adv}^{sound}_{\mathsf{TuPerm},\mathsf{R}_{\mathsf{tp}},\mathsf{X},4,\mathcal{A}}(\lambda) \leq 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{V},\mathbb{H},|\mathbb{H}|+|\mathbb{V}|), \mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ & + 5\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{H},\mathbb{H},2|\mathbb{H}|+|\mathbb{V}|), \mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \mathsf{Adv}^{sound}_{\mathsf{ProductCheck}}(\mathbb{H},\mathbb{H}), \mathsf{R}_{\mathsf{prod}},\mathsf{X}_{\mathsf{prod}},\mathcal{A}}(\lambda) \\ & + \frac{2|\mathbb{H}|}{|\mathbb{F}| - |\mathbb{H}|} + \frac{|\mathbb{V}|}{|\mathbb{F}| - |\mathbb{H}|}. \end{split}$$

*Proof.* We argue completeness and knowledge soundness separately.

<u>Completeness.</u> The honest prover first interpolates  $I, S_f, S_g, B_f, B_g, q$  as indicated in steps 1(b), 2(a), 3(a) and 4(b). Then in steps 1(c), 1(d), 1(e), 2(b), 3(b), 4(c) where the structures of these polynomials are tested, the verifier will succeed based on the completeness of zero-tests.

Assuming  $\{\!\{(f(\omega^i \mathbb{V}))\}\!\}_{i \in \mathbb{Z}_n} = \{\!\{(g(\omega^i \mathbb{V}))\}\!\}_{i \in \mathbb{Z}_n}$ , there is a permutation from cosets of f to cosets of g. Recall that  $S_f$  and  $S_g$  store the hash summation of f and g over cosets, respectively. If f and g are permuted over cosets, then  $S_f$ 

# $\mathsf{R}_{\mathsf{tp}} = \{ \bot, (\llbracket f \rrbracket, \llbracket g \rrbracket), (f, g) : \{\!\!\{(f(\omega^i \mathbb{V}))\}\!\!\}_{i \in [n]} = \{\!\!\{(g(\omega^i \mathbb{V}))\}\!\!\}_{i \in [n]} \}$

 $\mathsf{TuPerm}.\mathsf{P}(\bot,(\llbracket f\rrbracket,\llbracket g\rrbracket),(f,g))\leftrightarrow\mathsf{TuPerm}.\mathsf{V}(\bot,(\llbracket f\rrbracket,\llbracket g\rrbracket))$ 

- (1) P computes and sends the position-indexing polynomial I(X) and proves its well-formedness:
  - (a) V sends random challenges  $\beta \in \mathbb{F} \setminus \mathbb{H}$
  - (b) P computes and sends I defined over  $\mathbb{H}$  setting the evaluation of the  $j^{th}$  element of each coset to be j-th power-of- $\beta$  randomness,  $\beta^j$ :

$$\left[\left[I(\omega^{i}\gamma^{j})=\beta^{j}\right]_{i\in[n]}\right]_{j\in[m]}$$

- (c) P and V engage in  $\text{ZeroTest}(\mathbb{V},\mathbb{H})$  to prove  $L_{1,\mathbb{V}}(X)(I(X)-1)=0$  over  $\mathbb{V}$ .
- (d) P and V engage in ZeroTest( $\mathbb{V}, \mathbb{H}$ ) to prove  $(I(\gamma X) \beta \cdot I(X))(X \gamma^{m-1}) = 0$  over  $\mathbb{V}$ .
- (e) P and V engage in ZeroTest( $\mathbb{H},\mathbb{H}$ ) to prove  $(I(X) I(\omega X))Z_{\omega^{n-1}\mathbb{V}}(X) = 0$  over  $\mathbb{H}$
- (2) P computes and sends the summation polynomials  $S_f(X), S_g(X)$  for f(X), g(X), respectively, and proves their well-formedness:
  - (a) For  $p \in \{f, g\}$ , P computes and sends  $S_p$  defined over  $\mathbb{H}$  setting the evaluation to be constant in coset  $i \in \mathbb{Z}_n$ that represent the hash of  $[p(\omega^i \gamma^j)]_{j \in [m]}$ :

$$\left[ \left[ S_p(\omega^i \gamma^j) = \sum_{k \in [m]} \beta^k p(\omega^i \gamma^k) \right]_{i \in [n]} \right]_{j \in [m]}$$

- (b) P and V engage in  $\text{ZeroTest}(\mathbb{H},\mathbb{H})$  to prove every coset of  $\mathbb{V}$  in  $\mathbb{H}$  encodes some constant value:  $S_f(\gamma X) = C_f(\gamma X)$  $S_f(X), S_g(\gamma X) = S_g(X)$  over  $\mathbb{H}$
- (3) P computes and sends the induction polynomials  $B_f(X)$ ,  $B_q(X)$  for f(X), g(X), respectively, and prove their well-formedness:
  - (a) For  $p \in \{f, g\}$ , P computes and sends  $B_p$  defined over  $\mathbb{H}$  that accumulates the normalized hash:

$$\begin{split} \left[B_p(\omega^i \gamma^0) = 0\right]_{i \in [n]} \\ \left[\left[B_p(\omega^i \gamma^j) = \sum_{k \in [j]} \beta^k p(\omega^i \gamma^k) - j \cdot \frac{S_p(\omega^i \gamma^j)}{m}\right]_{i \in [n]}\right]_{j \in [m-1]} \end{split}$$

(b)  $\mathsf{P}$  and  $\mathsf{V}$  engage in  $\mathsf{ZeroTest}(\mathbb{H},\mathbb{H})$  to prove induction

$$B_f(\gamma X) = (B_f(X) + I(\gamma X) \cdot f(\gamma X)) - \frac{S_f(X)}{m},$$

$$B_g(\gamma X) = (B_g(X) + I(\gamma X) \cdot g(\gamma X)) - \frac{S_g(X)}{m} \text{ over } \mathbb{H}.$$

- (4) P computes and sends the ratio polynomial q(X) and prove it multiples to 1 over  $\mathbb{H}$ :
  - (a) V sends random challenges  $r \in \mathbb{F} \setminus \mathbb{H}$
  - (a) V sends random charges  $r \in \mathbb{R}^{n}$  (b) P computes and sends q defined over  $\mathbb{H}$  that encodes the ratio polynomial:  $\left[q(x) = \frac{r+S_f(x)}{r+S_g(x)}\right]_{x \in \mathbb{H}}$
  - (c) P and V engage in  $\text{ZeroTest}(\mathbb{H},\mathbb{H})$  to prove  $q(X)(S_g(X) + r) = S_f(X) + r$  over  $\mathbb{H}$
  - (d) P and V engage in ProductCheck $(\mathbb{H},\mathbb{H})$  to prove  $\prod_{x\in\mathbb{H}}q(x)=1$

Figure 29: TuPerm: Vector permutation protocol.

and  $S_g$  are also permuted over cosets. In particular, they are also permuted over the entire group  $\mathbb{H}$ . Since products are commutative, for any random element r, products  $\prod_{x \in \mathbb{H}} (S_f(x) + r) = \prod_{x \in \mathbb{H}} (S_g(x) + r)$  remain the same after permutation. Therefore, the product-check in 4(d) will proceed from its completeness.

<u>Knowledge soundness.</u> We bound the advantage of adversary  $\mathcal{A}$  by bounding the advantage of each of a series of game hops [BR06]. We define  $G_0 = \text{SOUND}_{\text{TuPerm}, \text{R}_{\text{tp}}, \text{X}, 4}^{\mathcal{A}}(\lambda)$ . The inequality above follows from the following claims that we will justify:

(1)

$$\begin{split} |\Pr[G_0 = 1] - \Pr[G_1 = 1]| &\leq 2\mathsf{Adv}_{\mathsf{ZeroTest}}^{sound}(\mathbb{W}, \mathbb{H}, |\mathbb{H}| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{x}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ZeroTest}}^{sound}(\mathbb{H}, \mathbb{H}, |\mathbb{H}| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{x}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(2)

$$\Pr[G_1 = 1] - \Pr[G_2 = 1]| \leq 3\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\mathbb{H},\mathbb{H},2|\mathbb{H}|),\mathsf{R}_{\mathsf{Zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda)$$

(3)

$$\begin{split} |\Pr[G_2 = 1] - \Pr[G_3 = 1]| &\leq \mathsf{Adv}_{\mathsf{ZeroTest}}^{sound}(\mathbb{H}, \mathbb{H}, 2|\mathbb{H}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{x}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{ProductCheck}(\mathbb{H}, \mathbb{H}), \mathsf{R}_{\mathsf{prod}}, \mathsf{x}_{\mathsf{prod}}, \mathcal{A}}(\lambda) + \frac{2|\mathbb{H}|}{|\mathbb{F}| - |\mathbb{H}|} + \frac{|\mathbb{V}|}{|\mathbb{F}| - |\mathbb{H}|} \end{split}$$

(4)  $\Pr[G_3 = 1] = 0$ 

The plan for the soundness proof is as follows: Claim 1 argues that polynomial I is constructed properly. Claim 2 argues that  $S_f$  and  $S_g$  encode hashes of each coset of f and g, respectively. Claim 3 argues that  $S_f$  and  $S_g$  are permutations over  $\mathbb{H}$ , and if so then f and g are coset permutations. Lastly, Claim 4 argues that the constructed extractor always succeeds for an accepting verifier.

*Claim 1 and 2:* Our first two steps are the same as Claim 1 and 2 in Theorem 3 except that we only have one *I* polynomial since everything is over the same group.

*Claim 3:* In the third claim, we want to prove that f and g are coset permutations. First, we argue that  $S_f$  and  $S_g$  are permutations of each other. In step 4(c) and 4(d), we check that

- $q(X)(S_g(X) + r) = S_f(X) + r$  over  $\mathbb{H}$  with advantage  $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{ZeroTest}(\mathbb{H},\mathbb{H},2|\mathbb{H}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda)$ : Checks q is the correct quotient polynomial of  $S_f$  and  $S_g$ .
- $\prod_{x \in \mathbb{H}} q(x) = 1$  with advantage  $\operatorname{Adv}_{\operatorname{ProductCheck}(\mathbb{H},\mathbb{H}),\operatorname{R}_{\operatorname{prod}},\operatorname{X}_{\operatorname{prod}},\mathcal{A}}(\lambda)$ : Checks the product of the quotient of  $S_f$  and  $S_g$  is equal to 1. This will allow us to argue as follows that  $S_f$  and  $S_g$  are permutations.

We define polynomials  $F, G \in \mathbb{F}[Y]$  as

$$F(Y) = \prod_{x \in \mathbb{H}} (S_f(x) + Y), \qquad G(Y) = \prod_{x \in \mathbb{H}} (S_g(x) + Y)$$

Observe that F(Y) and G(Y) are equivalent polynomials if and only if  $S_f$  and  $S_g$  are permutations over  $\mathbb{H}$ . Given a random verifier challenge r, we use the Schwartz-Zippel lemma to bound the probability that F(r) = G(r) (checked in the above zero test and product test) to  $\frac{|\mathbb{H}|}{|\mathbb{F} \setminus \mathbb{H}|}$ . G<sub>3</sub> employs the zero test extractor  $X_{zt}$  and the product check extractor  $X_{ProductCheck}$  to check the above tests and aborts if the extractor fails.

Lastly, since  $S_f$  and  $S_g$  encode the hashes of cosets of f and g, the coset hashes across f and g must be permutations of each other. By the collision resistance of the coset hash, we argue that f and g are thus coset permutations of each other. By an application of Schwartz-Zippel lemma (or Reed-Solomon encoding), we have that if  $\forall i \in [n]$  if  $S_f(\omega^i) = S_g(\omega^{\pi(i)})$  for permutation  $\pi$  then  $f(\omega^i \mathbb{V}) = g(\omega^{\pi(i)} \mathbb{V})$ , i.e.,

$$\left(f(\omega^{i}), f(\omega^{i}\gamma), \dots, f(\omega^{i}\gamma^{m-1})\right) = \left(g(\omega^{\pi(i)}), g(\omega^{\pi(i)}\gamma), \dots, g(\omega^{\pi(i)}\gamma^{m-1})\right).$$

In G<sub>3</sub>, a bad flag is set in a series of hybrids for each coset  $i \in [n]$  if this is not the case. We bound the probability of the flag being set to  $\frac{|\nabla|}{|\mathbb{F} \setminus \mathbb{H}|}$  in each hybrid with a total union bound of  $\frac{|\mathbb{H}|}{|\mathbb{F} \setminus \mathbb{H}|}$ .

Claim 4: Finally, we construct our extractor X that always succeeds on a verifying prover. X employs  $X_{zt}$  to retrieve and output f, g. By Claim 2, in  $G_2$ , if the verifier succeeds,  $X_{zt}$  always succeeds and so our extractor will always succeed.

**Corollary 26.** Using random linear combinations, k-TuPerm for  $R_{k-tp}$  satisfies perfect completeness and for any adversary A against knowledge soundness, we provide an extractor X using  $X_{zt}$ , an extractor for ZeroTest, and using  $X_{prod}$ , an extractor for ProductCheck such that

$$\begin{split} & \mathsf{Adv}^{\mathrm{sound}}_{k\text{-}\mathsf{TuPerm},\mathsf{R}_{k\text{-}\mathsf{tp}},\mathsf{X},\mathcal{A}}(\lambda) \leq 2\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H},|\mathbb{H}|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ & + 5\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}(\mathbb{H},\mathbb{H},2|\mathbb{H}|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ProductCheck}(\mathbb{H},\mathbb{H}),\mathsf{R}_{\mathsf{prod}},\mathsf{X}_{\mathsf{prod}},\mathcal{A}}(\lambda) \\ & + \frac{(2+k)|\mathbb{H}|}{|\mathbb{F}| - |\mathbb{H}|} + \frac{|\mathbb{V}|}{|\mathbb{F}| - |\mathbb{H}|}. \end{split}$$

Vector permutation with different groups. Our goal eventually is to construct a vector lookup protocol where the lookup polynomial f and the table polynomial t are defined over different groups. Towards that goal, we adjust the k-vector permutation protocol to work over different groups; our protocol which we call k-XGTuPerm is given in Figure 30.

For this setup, there are four sets of polynomials  $f_i, t_i, u_{1,i}, u_{2,i} \in \mathbb{F}[X]$  for  $i \in [k]$ . We define the vectors of  $f_i$  and  $u_{1,i}$  over the cosets of  $\mathbb{V}$  in group  $\mathbb{H}_0$ , and the vectors of  $t_i$  and  $u_{2,i}$  over the cosets of  $\mathbb{V}$  in a different group  $\mathbb{H}_1$ . The relation  $\mathsf{R}_{k-\mathsf{xgtp}}$  checks that the list of vectors across  $f_i$  and  $t_i$  are a permutation of the vectors in  $u_{1,i}$  and  $u_{2,i}$ , and further that it is the same permutation across all k sets of polynomials. The relation is defined as follows:

$$\mathsf{R}_{k\text{-xgtp}} = \begin{cases} \left( \begin{array}{c} \bot, \\ [\llbracket f_i \rrbracket, \llbracket t_i \rrbracket, \llbracket u_{1,i} \rrbracket, \llbracket u_{2,i} \rrbracket ]_{i \in [k]}, \\ [f_i, t_i, u_{1,i}, u_{2,i} ]_{i \in [k]} \end{array} \right) : & \begin{cases} \left( (f_i(\psi^j \gamma^l))_{i \in [k]})_{l \in [m]} \right\}_{j \in [d_1]} \\ = \left\{ \left( (u_{1,i}(\psi^j \gamma^l))_{i \in [k]})_{l \in [m]} \right\}_{j \in [d_0]} \\ \cup \left\{ \left( (u_{1,i}(\psi^j \gamma^l))_{i \in [k]})_{l \in [m]} \right\}_{j \in [d_0]} \\ \cup \left\{ ((u_{2,i}(\psi^j \gamma^l))_{i \in [k]})_{l \in [m]} \right\}_{j \in [d_1]} \end{cases} \right) \end{cases}$$

Intuitively, the construction splits the task into two parts (over two different groups) and generates vectors from each part separately. The completeness and soundness follow immediately from *k*-TuPerm and XGProductCheck, represented in the following corollary.

**Corollary 27.** *k*-XGTuPerm for  $R_{k-xgtp}$  (Figure 30) satisfies perfect completeness and for any adversary A against knowledge soundness, we provide an extractor X using  $X_{zt}$ , an extractor for ZeroTest, and using  $X_{xgprod}$ , an extractor for xgprod such that

$$\mathsf{Adv}_{k\mathsf{-XGTuPerm},\mathsf{R}_{k\mathsf{-tp2}},\mathsf{X},6,\mathcal{A}}^{\mathrm{sound}}(\lambda) \leq \frac{(17+k)|\mathbb{H}_0| + (16+k)|\mathbb{H}_1| + \max(|\mathbb{H}_0|,|\mathbb{H}_1|) + 4|\mathbb{V}| + 4|\mathbb{V}|$$

$$\begin{split} & \mathsf{Adv}^{\mathrm{sound}}_{k-\mathsf{XGTuPerm},\mathsf{R}_{k-\mathsf{tp2}},\mathsf{X},6,\mathcal{A}}(\lambda) \leq \mathsf{5}\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}}(\mathbb{V},\mathbb{H}_{0}\cup\mathbb{H}_{1},\max(|\mathbb{H}_{0}|+|\mathbb{H}_{1}|)+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ & + \mathsf{5}\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}}(\mathbb{H}_{0},\mathbb{H}_{0}\cup\mathbb{H}_{1},2|\mathbb{H}_{0}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \mathsf{5}\mathsf{Adv}^{\mathrm{sound}}_{\mathsf{ZeroTest}}(\mathbb{H}_{1},\mathbb{H}_{0}\cup\mathbb{H}_{1},2|\mathbb{H}_{1}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ & + \mathsf{Adv}^{\mathrm{sound}}_{\mathsf{xgprod}}(\mathbb{H}_{0},\mathbb{H}_{1},\mathbb{H}_{0}\cup\mathbb{H}_{1}),\mathsf{R}_{\mathsf{xgprod}},\mathsf{X}_{\mathsf{xgprod}},\mathcal{A}}(\lambda) \\ & + \frac{(3+k)|\mathbb{H}_{0}|}{|\mathbb{F}|-|\mathbb{H}_{0}|} + \frac{(2+k)|\mathbb{H}_{1}|}{|\mathbb{F}|-|\mathbb{H}|} + \frac{|\mathbb{V}|}{|\mathbb{F}|-|\mathbb{H}|}. \end{split}$$

$$\begin{aligned} & \left\{ \begin{pmatrix} 1, \\ [f_{1}], [u_{1}], [u_{1}, ], [u_{2,1}]]_{i \in [k]}, \\ [f_{1}, [i_{1}, u_{1,1}, u_{2,1}]_{i \in [k]}, \\ \end{bmatrix} : \begin{cases} \left\{ \left( (f_{1}(\psi^{j,\gamma}^{1})_{i \in [k]})_{i \in [k]} \right) \\ = \left\{ \left( (u_{1,i}(\psi^{j,\gamma}^{1})_{i \in [k]})_{i \in [m]} \right) \\ [f_{1}(u_{1,i}, u_{1,1}, u_{2,1})_{i \in [k]}, \\ \end{bmatrix} \end{cases} \right\} \\ & \left\{ k \times \text{XGTuPerm}, \left\{ \begin{pmatrix} 1, \\ [f_{1}, [f_{1}], [u_{1,1}], [u_{2,1}]]_{i \in [k]}, \\ [f_{1,i}, u_{1,1}, u_{2,1}, u_{2,1}]_{i \in [k]}, \\ \end{bmatrix} \right\} \\ & \leftrightarrow \text{XGTuPerm}, \left\{ \begin{pmatrix} 1, \\ [f_{1}, [f_{1}], [u_{1,1}], [u_{2,1}]]_{i \in [k]}, \\ [f_{1,i}, u_{1,1}, u_{2,1}, u_{2,1}]_{i \in [k]}, \\ \end{bmatrix} \right\} \\ & \leftrightarrow \text{XGTuPerm}, \left\{ \left( 1, [f_{1}], [u_{1,1}], [u_{1,1}], [u_{2,1}]]_{i \in [k]}, \\ [f_{1,i}, u_{1,1}, u_{2,1}, u_{2,1}]_{i \in [k]}, \\ \end{bmatrix} \right\} \\ & \leftrightarrow \text{XGTuPerm}, \left\{ \left( 1, [f_{1}], [u_{1,1}], [u_{2,1}]]_{i \in [k]}, \\ [f_{1,i}, u_{1,1}, u_{2,1}, u_{2,1}]_{i \in [k]}, \\ \end{bmatrix} \right\} \\ & \rightarrow \text{XGTuPerm}, \left\{ \left( 1, [f_{1,i}], [u_{1,i}], [u_{1,i}], [u_{2,i}]]_{i \in [k]}, \\ [f_{1,i}, u_{1,i}, u_{2,i}, u_{2,i}]_{i \in [k]}, \\ \end{bmatrix} \right\} \\ & \rightarrow \text{XGTuPerm}, \left\{ \left( 1, [f_{1,i}], [u_{1,i}], [u_{1,i}], [u_{2,i}]]_{i \in [k]}, \\ [f_{1,i}, u_{1,i}, u_{2,i}, u_{2,i}]_{i \in [k]}, \\ \end{bmatrix} \right\} \\ & \rightarrow \text{XGTuPerm}, \left\{ \left( 1, [f_{1,i}], [u_{1,i}], [u_{1,i}], [u_{1,i}], [u_{2,i}]]_{i \in [k]}, \\ \end{bmatrix} \right\} \\ & \rightarrow \text{XGTuPerm}, \left\{ 1, [f_{1,i}], [u_{1,i}], [u_{1,i}], [u_{1,i}], [u_{2,i}]]_{i \in [k]}, \\ \end{bmatrix} \\ & - \text{V such andom challeng } a \leftarrow 4^{k} \left\{ (0, (0, (0, 1)), \\ - \text{Yotop and scales dend be position-indextign polynomials } I_{1}(\chi) (\chi) (\chi) (\chi) = 10 \text{ our } Y, \\ - \text{V and V engge in ZeroTest(V, H_{0,i} \cup H_{1}) \text{ to prove } \\ I_{1,i}(\chi)(\chi)(\chi)(I_{i}(\chi)) = 1 \text{ our } Y, \\ - \text{P and V engge in ZeroTest(V, H_{0,i} \cup H_{1}) \text{ to prove } I_{1,i}(\chi)(\chi)(\chi)(I_{i}(\chi)) = 1 \text{ our } Y, \\ - \text{P and V engge in ZeroTest(V, H_{0,i} \cup H_{1}) \text{ to prove } I_{1,i}(\chi)(\chi)(\chi)(\chi)(\chi)(\chi)(\chi) = 0 \text{ our } H_{1}, \\ - \text{P and V engge in ZeroTest(H_{1,i}, H_{0,i} \cup H_{1}) \text{ to prove } I_{1,i}(\chi)(\chi)(\chi)(\chi)(\chi)(\chi) = 0 \text{ our } H_{1}, \\ \end{bmatrix} \\ \left\{ p = \text{ und } \text{ engge in ZeroTest(H_{0,i}, H_{0,i} \cup H_{1}) \text{ to prove } I_{1,i}(\chi)(\chi)(\chi)(\chi)(\chi)(\chi)(\chi) = 0 \text{ our } H$$

- P and V engage in Zero Test( $\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1$ ) to prove  $B_{p_2}(\gamma X) = (B_{p_2}(X) + I_2(\gamma X) \cdot p_2(\gamma X)) - \frac{S_{p_2}(X)}{m}$  over  $\mathbb{H}_1$ .

(5) P computes and sends the ratio polynomials  $q_1(X), q_2(X)$  and prove they multiply to 1 over  $\mathbb{H}_0, \mathbb{H}_1$ , respectively:

– V sends random challenges  $r \in \mathbb{F} \setminus \mathbb{H}$ .

- P computes and sends  $q_1$  defined over  $\mathbb{H}_0$  that encodes the fraction polynomial  $\frac{r+S_f(X)}{r+S_{u_1}(X)}$ , and  $q_2$  defined over  $\mathbb{H}_1$  that encodes the fraction polynomial  $\frac{r+S_{u_2}(X)}{r+S_t(X)}$ - P and V engage in ZeroTest( $\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1$ ), ZeroTest( $\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1$ ) respectively to prove  $q_1(X)(S_{u_1}(X) + r) = S_f(X) + r$  over  $\mathbb{H}_0$  and
- $q_2(X)(S_t(X) + r) = S_{u_2}(X) + r$  over  $\mathbb{H}_1$ .
- P and V engage in XGProductCheck to prove  $\prod_{x \in \mathbb{H}_0} q_1(x) = \prod_{x \in \mathbb{H}_1} q_2(x)$ .

Figure 30: k-XGTuPerm: Tuple permutation across different groups.

 $\mathsf{R}_{\mathsf{vlkup}} = \left\{ \bot, (\llbracket f \rrbracket, \llbracket t \rrbracket), (f, t) : \left\{ (f(\psi^i \mathbb{V})) \right\}_{i \in [d_0]} \subseteq \left\{ (t(\mu^i \mathbb{V})) \right\}_{i \in [d_1]} \right\}$ 

 $\mathsf{TuPlookup.P}(\bot,(\llbracket f\rrbracket,\llbracket t\rrbracket),(f,t))\leftrightarrow\mathsf{TuPlookup.V}(\bot,(\llbracket f\rrbracket,\llbracket t\rrbracket))$ 

(1) P computes and sends oracle polynomials  $u_1, u_2$  encoding the vector s where s is the canonically sorted vector of cosets from  $\{\{(f(\psi^i \mathbb{V}))\}\}_{i \in [d_0]} \cup \{\{(t(\mu^i \mathbb{V}))\}\}_{i \in [d_1]}\}$ .

 $\left[u_1(\psi^i \mathbb{V}) = s[i]\right]_{i \in [d_0]}, \quad \left[u_2(\mu^i \mathbb{V}) = s[d_0 + i]\right]_{i \in [d_1]}.$ 

(2) P computes and sends polynomials u'<sub>1</sub>, u'<sub>2</sub> that encode shifted versions of u<sub>1</sub>, u<sub>2</sub> wrapping the last coset from u<sub>2</sub> to u'<sub>1</sub> and wrapping the last coset of u<sub>1</sub> to u'<sub>2</sub>, then proving wellformedness.

(a) P interpolates  $u'_1, u'_2$  over  $\mathbb{H}_0, \mathbb{H}_1$  respectively with the following defined evaluations:

$$u_1'(\mathbb{V}) = u_2(\mu^{d_1-1}\mathbb{V}), \ \left[u_1'(\psi^i\mathbb{V}) = u_1(\psi^{i-1}\mathbb{V})\right]_{i \in [1,d_0)}, \ u_2'(\mathbb{V}) = u_1(\psi^{d_0-1}\mathbb{V}), \ \left[u_2'(\mu^i\mathbb{V}) = u_2(\mu^{i-1}\mathbb{V})\right]_{i \in [1,d_1]}.$$

(b) P and V engage in ZeroTest( $\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1$ ) to prove  $u'_1(X) = u_2(\mu^{d_1-1}X)$  over  $\mathbb{V}$ .

(c) P and V engage in  $\operatorname{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1)$  to prove  $(u_1'(X) - u_1(\psi^{-1}X))Z_{\mathbb{V}}(X) = 0$  over  $\mathbb{H}_0$ .

(d) P and V engage in  $\operatorname{\mathsf{ZeroTest}}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1)$  to prove  $u'_2(X) = u_1(\psi^{d_0-1}X)$  over  $\mathbb{V}$ .

(e) P and V engage in  $\text{ZeroTest}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1)$  to prove  $(u'_2(X) - u_2(\mu^{-1}X))Z_{\mathbb{V}}(X) = 0$  over  $\mathbb{H}_1$ .

(3) P and V engage in 2-XGTuPerm with oracles for  $(f, f), (t, t'), (u'_1, u_1), (u'_2, u_2)$  where t' is the virtual oracle  $t(\mu X)$  to prove

 $\{\!\!\{(f(\psi^{j}\mathbb{V}), f(\psi^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{0}]} \cup \{\!\!\{(t(\mu^{j}\mathbb{V}), t(\mu^{j+1}\mathbb{V}))\}\!\!\}_{j\in[d_{1}]} = \{\!\!\{(u_{1}'(\psi^{j}\mathbb{V}), u_{1}(\psi^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{0}]} \cup \{\!\!\{(u_{2}'(\mu^{j}\mathbb{V}), u_{2}(\mu^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{1}]} \cdot (u_{2}'(\mu^{j}\mathbb{V}), u_{2}(\mu^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{1}]} = \{\!\!\{(u_{1}'(\psi^{j}\mathbb{V}), u_{1}(\psi^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{0}]} \cup \{\!\!\{(u_{2}'(\mu^{j}\mathbb{V}), u_{2}(\mu^{j}\mathbb{V}))\}\!\}_{j\in[d_{1}]} \cdot (u_{2}'(\mu^{j}\mathbb{V}), u_{2}(\mu^{j}\mathbb{V}))\}\!\}_{j\in[d_{1}]} \cdot (u_{2}'(\mu^{j}\mathbb{V}), u_{2}(\mu^{j}\mathbb{V}))\}$ 

Figure 31: TuPlookup: Vector lookup argument

#### F.2 Vector Lookup from Vector Permutation

Given two vectors  $t \in \mathbb{F}^{d_1}$ ,  $f \in \mathbb{F}^{d_0}$ , Plookup [GW20, PFM<sup>+</sup>22] presents an argument for  $\{f[i]\}_{i \in [d_0]} \subseteq \{t[i]\}_{i \in [d_1]}$ . Again, we take inspiration from the existing approach and extend the result to support vectors defined over coset evaluations. The relation for TuPlookup as follows:

 $\mathsf{R}_{\mathsf{vlkup}} = \left\{ \bot, (\llbracket f \rrbracket, \llbracket t \rrbracket), (f, t) : \left\{ (f(\psi^i \mathbb{V})) \right\}_{i \in [d_0]} \subseteq \left\{ (t(\mu^i \mathbb{V})) \right\}_{i \in [d_1]} \right\}.$ 

We now give a high-level overview of our approach, detailed in Figure 31. For notational simplicity, for a polynomial p, let  $p_i$  denote the vector of evaluations of the *i*-th coset of p. For example, we let  $f_i = f(\psi^{i-1}\mathbb{V})$  and  $t_i = t(\mu^{i-1}\mathbb{V})$ . Consider the vector s containing vectors of f and t where t and s are sorted in the same canonical manner. [GW20] show that  $f \subset t$  if and only if the multiset of pairs  $(f_i, f_i)$  and  $(t_i, t_{i+1})$  is equal to the multiset of pairs  $(s_i, s_{i+1})$  of s. To use this fact, we must encode s as a polynomial; we do this by splitting s into lower and upper halves  $u_1$  and  $u_2$  such that  $u_1 = (s_0, \ldots, s_{d_0-1})$  and  $u_2 = (s_{d_0}, \ldots, s_{d_0+d_1-1})$ . We then construct shifted polynomials  $u'_1, u'_2$  where  $u'_1 = (s_{d_0+d_1-1}, s_0, \ldots, s_{d_0-2})$  and  $u'_2 = (s_{d_0-1}, \ldots, s_{d_0+d_1-2})$ . In this way, we can use  $(u_1, u'_1)$  and  $(u_2, u'_2)$  along with a cross-group vector permutation argument to compare the multiset of pairs  $(s_i, s_{i+1})$  to a similarly constructed multiset of pairs derived from f and t.

We provide the following theorem for the completeness, knowledge soundness and efficiency of our vector lookup argument.

**Theorem 28.** TuPlookup for  $R_{vlkup}$  (Figure 31) satisfies perfect completeness and for any adversary A against knowledge soundness, we provide an extractor X using  $X_{zt}$ , an extractor for ZeroTest, and using  $X_{xgprod}$ , an extractor for xgprod such that

$$\begin{split} &\mathsf{Adv}^{sound}_{\mathsf{TuPlookup},\mathsf{R}_{\mathsf{vlkup}},\mathsf{X},7,\mathcal{A}}(\lambda) \leq 8\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{V},\mathbb{H}_0 \cup \mathbb{H}_1,\max(|\mathbb{H}_0|,|\mathbb{H}_1|) + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ 6\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{H}_0,\mathbb{H}_0 \cup \mathbb{H}_1,2|\mathbb{H}_0| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 6\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{H}_1,\mathbb{H}_0 \cup \mathbb{H}_1,2|\mathbb{H}_1| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{xgprod}}(\mathbb{H}_0,\mathbb{H}_1,\mathbb{H}_0 \cup \mathbb{H}_1), \mathsf{R}_{\mathsf{xgprod}},\mathsf{X}_{\mathsf{xgprod}},\mathcal{A}}(\lambda) + \frac{5|\mathbb{H}_0|}{|\mathbb{F}| - |\mathbb{H}_0|} + \frac{4|\mathbb{H}_1|}{|\mathbb{F}| - |\mathbb{H}_1|} + \frac{|\mathbb{V}|}{|\mathbb{F}| - |\mathbb{H}|} \end{split}$$

*Proof.* We argue completeness and knowledge soundness separately.

<u>Completeness</u>. The zero tests proving wellformedness in steps (2-4) follow directly from the construction of the polynomials and will succeed by the completeness of the zero test protocol. Next, we argue that the vector permutation

argument in step (5) will succeed for valid witnesses using the same argument as [GW20]. By the construction of polynomials  $u_1, u_2, u'_1, u'_2$  with respect to the sorted vector s, we have that the vector permutation checks the following:  $\{(f(\psi^i \mathbb{V}), f(\psi^i \mathbb{V}))\}_{i \in [d_0]} \cup \{(t(\mu^i \mathbb{V}), t(\mu^{i+1} \mathbb{V})\}_{i \in [d_1]} = \{(s_i, s_{i+1})\}_{i \in [d_0+d_1-1]} \cup (s_{d_0+d_1-1}, s_0)$ . First, consider  $x = f(\psi^i \mathbb{V})$  wlog for some i. If x has multiplicity  $\ell$  in f and  $x \in t$ , then the vector s has  $\ell + 1$  (assuming x has multiplicity 1 in t, though the argument extends if the multiplicity is > 1 as well). This means that for the  $\ell$  pairs of  $(f(\psi^i \mathbb{V}), f(\psi^i \mathbb{V}))$  contributed to the multiset, the same  $\ell$  pairs are contributed to the  $(s_i, s_{i+1})$  multiset by having  $\ell + 1$  consecutive sorted values of x.

Next, consider wlog a pair  $(t(\mu^i \mathbb{V}), t(\mu^{i+1} \mathbb{V}))$  for some *i*. Since *t* and *s* are sorted in the same canonical manner, if *f* does not contain any vectors not present in *t*, then *s* will also include a consecutive pair matching  $(t(\mu^i \mathbb{V}), t(\mu^{i+1} \mathbb{V}))$ . Thus, the two multisets above are equivalent and the vector permutation argument will succeed by the completeness of 2-XGTuPerm.

<u>Knowledge soundness</u>. We bound the advantage through a series of game hops. First define  $G_0 = \text{SOUND}_{\mathsf{TuPlookup},\mathsf{R}_{\mathsf{vlkup}},\mathsf{X},7}(\lambda)$ . The inequality above follows from the following claims that we will justify:

$$\begin{split} |\Pr[G_0 = 1] - \Pr[G_1 = 1]| &\leq 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1, \max(|\mathbb{H}_0|, |\mathbb{H}_1|)), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1, |\mathbb{H}_0| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) + \mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1, |\mathbb{H}_1| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \end{split}$$

(2)

$$\begin{split} &|\Pr[G_1=1] - \Pr[G_2=1]| \leq 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1, |\mathbb{H}_0| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1, |\mathbb{H}_1| + |\mathbb{V}|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) + 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1, \max(|\mathbb{H}_0| + |\mathbb{H}_1|) + 1), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ 5\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1, 2|\mathbb{H}_0|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) + 5\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1, 2|\mathbb{H}_1|), \mathsf{R}_{\mathsf{zero}}, \mathsf{X}_{\mathsf{zt}}, \mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}^{sound}_{\mathsf{xgprod}}(\mathbb{H}_0, \mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1), \mathsf{R}_{\mathsf{xgprod}}, \mathsf{X}_{\mathsf{xgprod}}, \mathcal{A}}(\lambda) + \frac{5|\mathbb{H}_0|}{|\mathbb{F}| - |\mathbb{H}_0|} + \frac{4|\mathbb{H}_1|}{|\mathbb{F}| - |\mathbb{H}_1|} + \frac{|\mathbb{V}|}{|\mathbb{F}| - |\mathbb{H}|} \end{split}$$

(3)  $\Pr[G_2 = 1] = 0$ 

Claim 1 argues for the wellformedness of  $u'_1$  and  $u'_2$ . Claim 2 argues that the vector permutation is correct. Lastly, Claim 3 argues that this implies the vector lookup relation is satisfied and the constructed extractor always succeeds for an accepting verifier.

*Claim 1:* In the first step, we argue for the wellformedness of  $u'_1, u'_2$  in steps (4bcde):

- u'<sub>1</sub>(X) = u<sub>2</sub>(μ<sup>d<sub>1</sub>-1</sup>X) over V with advantage Adv<sup>sound</sup><sub>ZeroTest</sub>(V, H<sub>0</sub>∪H<sub>1</sub>,max(|H<sub>0</sub>|,|H<sub>1</sub>|)), R<sub>zero</sub>, X<sub>zt</sub>, A(λ): Checks the first coset of u'<sub>1</sub> is set to last coset of u<sub>2</sub>.
- $(u'_1(X) u_1(\psi^{-1}X))Z_{\mathbb{V}}(X) = 0$  over  $\mathbb{H}_0$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1, |\mathbb{H}_0| + |\mathbb{V}|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks that  $u'_1$  matches with the shifted  $u_1$  everywhere except for the first coset.
- $u'_2(X) = u_1(\psi^{d_0-1}X)$  over  $\mathbb{V}$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_0|,|\mathbb{H}_1|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}(\lambda)$ : Checks the first coset of  $u'_2$  matches the last coset of  $u_1$ .
- $(u'_2(X) u_2(\mu^{-1}X))Z_{\mathbb{V}}(X) = 0$  over  $\mathbb{H}_1$  with advantage  $\operatorname{Adv}_{\operatorname{ZeroTest}}^{\operatorname{sound}}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1, |\mathbb{H}_1| + |\mathbb{V}|), \mathbb{R}_{\operatorname{Zero}}, \mathsf{X}_{\operatorname{zt}}, \mathcal{A}}(\lambda)$ : Checks that  $u'_2$  matches with the shifted  $u_2$  everywhere except for the first coset.

In  $G_1$ , we invoke the zero test extractor  $X_{zt}$  to extract the witnesses and check if the above hold, aborting otherwise. We bound the probability of the bad flag being set by the advantage against the soundness of the zero test.

Claim 2: We then argue that the vector relation holds:

• 2-XGTuPerm for  $f, t, u'_1, u_1, u'_2, u_2$  with advantage

$$\begin{split} & 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_0\cup\mathbb{H}_1,|\mathbb{H}_0|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_0\cup\mathbb{H}_1,|\mathbb{H}_1|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ & + 2\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{V},\mathbb{H}_0\cup\mathbb{H}_1,\max(|\mathbb{H}_0|+|\mathbb{H}_1|)+1),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 5\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_0,\mathbb{H}_0\cup\mathbb{H}_1,2|\mathbb{H}_0|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ & + 5\mathsf{Adv}^{sound}_{\mathsf{ZeroTest}(\mathbb{H}_1,\mathbb{H}_0\cup\mathbb{H}_1,2|\mathbb{H}_1|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + \mathsf{Adv}^{sound}_{\mathsf{xgprod}(\mathbb{H}_0,\mathbb{H}_1,\mathbb{H}_0\cup\mathbb{H}_1),\mathsf{R}_{\mathsf{xgprod}},\mathsf{X}_{\mathsf{xgprod}},\mathcal{A}}(\lambda) \\ & + \frac{5|\mathbb{H}_0|}{|\mathbb{F}| - |\mathbb{H}_0|} + \frac{4|\mathbb{H}_1|}{|\mathbb{F}| - |\mathbb{H}|} + \frac{|\mathbb{V}|}{|\mathbb{F}| - |\mathbb{H}|} \end{split}$$

This checks the relation

$$\{\!\!\{(f(\psi^{j}\mathbb{V}), f(\psi^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{0}]} \cup \{\!\!\{(t(\mu^{j}\mathbb{V}), t(\mu^{j+1}\mathbb{V}))\}\!\!\}_{j\in[d_{1}]} \\ = \{\!\!\{(u_{1}'(\psi^{j}\mathbb{V}), u_{1}(\psi^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{0}]} \cup \{\!\!\{(u_{2}'(\mu^{j}\mathbb{V}), u_{2}(\mu^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{1}]}$$

Our extractor X employs  $X_{2-XGTuPerm}$  to check the above holds, and aborts if the extractor fails. The probability of the bad flag being set is bounded by the soundness advantage of the 2-XGTuPerm protocol.

Claim 3: Next, we argue that since the vector permutation holds, the vector lookup relation is satisfied. Let's first define s such that  $[u_1(\psi^i \mathbb{V}) = s[i]]_{i \in [d_0]}, [u_2(\mu^i \mathbb{V}) = s[d_0 + i]]_{i \in [d_1]}$ . Notice that  $u'_1$  is shifted  $u_1$  and  $u'_2$  is shifted  $u_2$ , and  $u_1$ 's last coset is connected to the first coset of  $u_2$ . Then the pairs we get are exactly the consecutive pairs of s:

$$\{ (u'_1(\psi^j \mathbb{V}), u_1(\psi^j \mathbb{V})) \}_{j \in [d_0]} \cup \{ (u'_2(\mu^j \mathbb{V}), u_2(\mu^j \mathbb{V})) \}_{j \in [d_1]} \\ = \{ (u_2(\mu^{d_1-1}\mathbb{V}), u_1(\mathbb{V})), (u_1(\mathbb{V}), u_1(\phi\mathbb{V})), \dots, (u_1(\phi^{d_0-2}\mathbb{V}), u_1(\phi^{d_0-1}\mathbb{V})) \} \\ \cup \{ (u_1(\phi^{d_0-1}\mathbb{V}), u_2(\mathbb{V})), (u_2(\mathbb{V}), u_2(\mu\mathbb{V})), \dots, (u_2(\mu^{d_1-2}\mathbb{V}), u_2(\mu^{d_1-1}\mathbb{V})) \} \\ = \{ (s_i, s_{i+1}) \}_{i \in [d_0+d_1-1]} \cup (s_{d_0+d_1-1}, s_0)$$

The argument from [GW20] then naturally generalizes to cosets: if

$$\{\!\!\{(f(\psi^{j}\mathbb{V}), f(\psi^{j}\mathbb{V}))\}\!\!\}_{j\in[d_{0}]} \cup \{\!\!\{(t(\mu^{j}\mathbb{V}), t(\mu^{j+1}\mathbb{V}))\}\!\!\}_{j\in[d_{1}]} \\ = \{\!\!\{(s_{i}, s_{i+1})\}\!\!\}_{i\in[d_{0}+d_{1}-1]} \cup (s_{d_{0}+d_{1}-1}, s_{0}),$$

then s must be sorted over cosets and  $\{(f(\psi^i \mathbb{V}))\}_{i \in [d_0]} \subseteq \{(t(\mu^i \mathbb{V}))\}_{i \in [d_1]}$ . The reason is that by permutation, each consecutive pair  $(s_i, s_{i+1})$  must be either  $(f(\psi^j \mathbb{V}), f(\psi^j \mathbb{V}))$  or  $(t(\mu^{j'} \mathbb{V}), t(\mu^{j'+1} \mathbb{V}))$  for some j, j'. Whenever  $s_i \neq s_{i+1}$ , it corresponds to  $(t(\mu^j \mathbb{V}), t(\mu^{j+1} \mathbb{V}))$  and we move to the next value in the table. All values of f must be values of t to remain consistency in consecutive pairs of s.

Finally, we construct our extractor X that always succeeds on a verifying prover. X employs  $X_{2-XGTuPerm}$  to extract and output f,t. By Claim 2, if the verifier succeeds then  $X_{2-XGTuPerm}$  succeeds and so X always succeeds.

Vector lookup across multiple tables. Lastly, as with k-TuPerm, it will be useful in our machine execution application to be able to perform a lookup across w lookup polynomial and table polynomial pairs ensuring that the same rows are read across all w pairs. To do this, we use the same trick of combining the w pairs using a random linear combination and performing a single TuPlookup.

**Corollary 29.** w-TuPlookup for  $R_{w-v|kup}$  (Figure 32) satisfies perfect completeness and for any adversary A against knowledge soundness, we provide an extractor X using  $X_{zt}$ , an extractor for ZeroTest, and using  $X_{xgprod}$ , an extractor

| $ \left[ R_{w\text{-vlkup}} = \left\{ \bot, ([\llbracket f_i \rrbracket]_{i \in [w]}, [\llbracket t_i \rrbracket]_{i \in [w]}), ([f_i, t_i]_{i \in [w]}) : \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_0]} \subseteq \left\{ (t_i(\mu^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} \right\}_{j \in [d_1 - 1]} \right] = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]} \right\}_{j \in [d_1 - 1]} = \left\{ (f_i(\psi^j \mathbb{V}))_{i \in [w]$ |
|---|
| $w-TuPlookup.P(\bot,([\llbracket f_i \rrbracket, \llbracket t_i \rrbracket]_{i \in [w]}), ([f_i, t_i]_{i \in [w]})) \leftrightarrow w-TuPlookup.V(\bot, ([\llbracket f_i \rrbracket, \llbracket t_i \rrbracket]_{i \in [w]}))$  |
| (1) P and V derive polynomials $f$ and $t$ as the linear combinations of $f_i$ 's and $t_i$ 's.   |
| – V sends a random challenge $\alpha \leftarrow \mathbb{F}$   |
| - Through additive homomorphism, P and V derive $t(X) = \sum_{i \in [w]} t_i(X)\alpha^i$ and $f(X) = \sum_{i \in [w]} f_i(X)\alpha^i$   |
| (2) P and V engage in TuPlookup to prove $\{(f(\psi^j \mathbb{V}))\}_{j \in [d_0]} \subseteq \{(t(\mu^j \mathbb{V}))\}_{j \in [d_1-1]}$   |

Figure 32: w-TuPlookup: Vector lookup argument enforcing same row lookups for w tables

for xgprod, such that

$$\begin{split} &\mathsf{Adv}_{w\text{-}\mathsf{TuPlookup},\mathsf{R}_{w\text{-}\mathsf{vlkup}},\mathsf{X},\mathsf{S},\mathcal{A}}(\lambda) \leq \mathsf{8Adv}_{\mathsf{ZeroTest}}^{\mathrm{sound}}(\mathbb{V},\mathbb{H}_0\cup\mathbb{H}_1,\max(|\mathbb{H}_0|,|\mathbb{H}_1|)+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ 6\mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathrm{sound}}(\mathbb{H}_0,\mathbb{H}_0\cup\mathbb{H}_1,2|\mathbb{H}_0|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) + 6\mathsf{Adv}_{\mathsf{ZeroTest}}^{\mathrm{sound}}(\mathbb{H}_1,\mathbb{H}_0\cup\mathbb{H}_1,2|\mathbb{H}_1|+|\mathbb{V}|),\mathsf{R}_{\mathsf{zero}},\mathsf{X}_{\mathsf{zt}},\mathcal{A}}(\lambda) \\ &+ \mathsf{Adv}_{\mathsf{xgprod}}^{\mathrm{sound}}(\mathbb{H}_0,\mathbb{H}_1,\mathbb{H}_0\cup\mathbb{H}_1),\mathsf{R}_{\mathsf{xgprod}},\mathsf{X}_{\mathsf{xgprod}},\mathcal{A}}(\lambda) + \frac{(5+w)|\mathbb{H}_0|}{|\mathbb{F}|-|\mathbb{H}_0|} + \frac{(4+w)|\mathbb{H}_1|}{|\mathbb{F}|-|\mathbb{H}_1|} + \frac{|\mathbb{V}|}{|\mathbb{F}|-|\mathbb{H}|}. \end{split}$$