MUXProofs: Succinct Arguments for Machine Computation from Tuple Lookups

Zijing Di
Stanford University

Lucas Xia
Stanford University

Wilson Nguyen
Stanford University

Nirvan Tyagi
Cornell University

Abstract. Proofs for machine computation allow for proving 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 program execution. This approach incurs prover cost per execution step on the order of the sum of instruction constraints for instructions in the set despite only a single instruction being executed. Existing approaches that avoid the universal cost per step (and incur only the cost of a single instruction's constraints per step) either fail to provide zero-knowledge of program execution or rely on recursive proof composition techniques where security derives from heuristic non-black-box random oracle instantiation.

We present a new protocol for proving machine execution that resolves the above limitations, allowing for prover efficiency on the order of executed instructions while achieving zero-knowledge and avoiding the use of proof recursion. Our core technical contribution is a new primitive that we call a tuple lookup argument which is used to allow a prover to build up a machine execution "on-the-fly". Our tuple lookup argument relies on univariate polynomial commitments in which tuples are encoded as evaluations on cosets of a multiplicative subgroup. We instantiate our protocol by combining our tuple lookup with the popular Marlin succinct non-interactive proof system.

1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) [Kil92, Mic94, GW11, BCCT12] allow a *prover* to produce a certificate that convinces a *verifier* of knowledge of a satisfying witness for an NP statement; the succinctness property means that the size and verification cost of the certificate are both sublinear in the length of the witness. Typically existing SNARKs for NP statements operate by proving knowledge of a witness for some NP-complete problem, e.g., arithmetic circuit satisfiability or rank-1 constraint satisfiability (R1CS) [Gro10, GGPR13, Gro16, BBHR19, GWC19, CHM^{+}20, Set20]. In practice, creating a proof for a statement first requires compiling the statement-witness relation to a low-level constraint system. While there exist a number of tools for optimizing this compilation step [BCCT12, BCG^{+}13, GGPR13, BFR^{+}13, FL14, WSR^{+}15, CFH^{+}15, KPS18, OBW20], the concrete blow-up in relation size when described as low-level constraints has limited the applications in which SNARKs can be practically deployed due to a corresponding blow-up in prover computation time. Thus, a key challenge in the design of SNARKs is improving prover time. In this work, we improve prover time for a particular class of statements known as *machine computation* [BFR^{+}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 knowledge of which clause is satisfied remains hidden. A standard approach to proving 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 ℓ 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 taking advantage of similar structure 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, ℓ is the number of instructions in the instruction set, and C is the constraint size of an 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 started to be deployed in the context of improving scalability of blockchain auditing, in particular, with respect to the auditing of smart contract execution. Instead of requiring auditors to execute smart contracts locally to determine and verify new blockchain state, auditors can 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 prover cost 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+20, XCZ+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+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 and recursively verify a proof for 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+21] and further employing techniques in offline memory checking [BEG+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. Outside of recent exploratory work [CCS22, CCG+23], we do not have practical techniques for producing SNARKs on computations that query random oracles. Thus, in practice, this means that the security of these protocols is based on a heuristic step in which a concrete hash function is used in place of the random oracle to encode the recursive computation in a manner suitable for existing SNARKs; the protocols have not

Protocols	Prover computation?	Execution leakage?	Recursion heuristic?
IVC with universal constraints [BCTV14a]	$\tilde{O}(n\ell C)$	no leakage	yes
non-uniform IVC SuperNova [KS22], Protostar [BC23]	$\tilde{O}((n+\ell)C)$	no leakage	yes
Unroll with universal constraints [BCTV14b], Arya [BCG+18], [BCG+19]	$\tilde{O}(n\ell C)$	instruction upper bound	no
Unroll executed instructions			
Pantry [BFR+13], Buffet [WSR+15]	$\tilde{O}((n+\ell)C)$	full execution	no
vRAM [ZGK+18]	$\tilde{O}((n+\ell)C)$	instruction multiplicity	no
Mux-Marlin (This work)	$\tilde{O}((n+\ell)C)$	instruction upper bound	no
Sublonk [CGG+23] (Concurrent work)	$\tilde{O}(nC)$	instruction upper bound	no

Figure 1: Summary of strategy and characteristics for machine execution proof protocols. The asymptotic execution time is given in terms of the number of executed instructions n, the number of instructions in the instruction set ℓ, and the constraint size of a single instruction C. Execution leakage refers to aspects of the program execution that are revealed to the verifier. Recursion heuristic refers to the heuristic step used in recursive proof composition to prove a computation that queries a random oracle by instantiating the oracle as a concrete hash function.

been shown to be secure in the random oracle model.

As we describe next, an alternate strategy avoids IVC (and its associated heuristic proof step) by unrolling and proving the full program execution as one.

Unrolled proof systems for machine computation. 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+18, BCG+19]. Other unroll approaches including Pantry [BFR+13], Buffet [WSR+15], and vRAM [ZGK+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. Existing 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: it is not known ahead of time which instruction will be executed at each execution step.

Our contributions. 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 other 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 with heuristic random oracle instantiation during recursion. To do this, our main technical contribution is a new proof system for what we call *tuple lookups*. A tuple lookup allows proving that a commitment to a vector of tuples contains only tuples that exist in a reference table of tuples, i.e, that every tuple was "looked up" from some index in the table. Equipped with our new tuple lookup, we proceed to construct an unrolled proof system for machine computation. Our approach combines tuple 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 polynomial commitment in a preprocessing step. In our approach, the polynomial commitments representing each instruction in the instruction set are encoded together in a table that represents the machine. Then a tuple lookup is used to construct a polynomial commitment on-the-fly that represents the unrolled machine execution. We instantiate our approach with a popular polyIOP system, Marlin [CHM+20] and can be compiled to a zero-knowledge, non-interactive protocol using the Fiat-Shamir transform in the random oracle model along with an appropriate polynomial commitment scheme. For example, when instantiated with the Marlin-KZG polynomial commitment scheme [CHM+20], this results in a proof system for machine computation that admits zero-knowledge, constant-size proofs, logarithmic verifier computation, and prover computation on the order of the executed instructions.

Concurrent work. There exist a number of concurrent works targeting similar improvements in prover time for proving correct machine execution. We have already compared to SuperNova [KS22] and Protostar [BC23] above. Both these works achieve the same asymptotic prover time but take the IVC approach with a heuristic recursion step.

The work most closely related to ours is Sublonk [CGG$^+$23] which achieves similar asymptotic prover time and takes the same unrolled approach as in Mux-Marlin. Further, to achieve this, Sublonk follows the same high level recipe as Mux-Marlin as discussed in Section 2, in first constructing a tuple lookup argument for cosets (referred to as a segment lookup argument [CGG$^+$23]), and then applying the lookup argument to a suitable polyIOP, Plonk [GWC19]. Two differences in instantiation of the shared recipe:

(1) Performance characteristics of the tuple lookup argument: Our tuple lookup argument builds off the Plookup lookup argument [GW20], while Sublonk builds off the cached quotients lookup argument [EFG22]. The performance tradeoffs of the resulting tuple lookup arguments mirror that of the underlying lookup arguments: Plookup versus cached quotients. For table size ℓ (i.e., instruction set size) and lookup size n (i.e., number of executed instructions), to perform a lookup, Plookup incurs prover costs $\tilde{O}(n+\ell)$ while cached quotients incurs cost only on the order of the lookups, $\tilde{O}(n)$. When $n \ll \ell$, cached quotients is more performant as the online prover runtime does not depend on ℓ. However when $n \geq \ell$, Plookup may be preferable due to its lower constant factor costs and that it does not require a precomputed state of size $O(\ell)$ as in cached quotients. In machine execution, it is reasonable to expect the number of executed instructions n to eclipse the instruction set size ℓ.

(2) Choice of polyIOP application: We apply our tuple lookup argument to Marlin [CHM$^+$20] to produce Mux-Marlin, while Sublonk is the result of applying their tuple lookup argument to Plonk [GWC19]. Using our terminology from Section 2, to apply the tuple lookup argument in the context of machine execution, the polyIOP computation commitment must have a global structure that can be recovered through post-processing. Sublonk provides this post-processing for the Plonk polyIOP.

Together, our work and the work of Choudhuri et al. [CGG$^+$23] demonstrate the generality and modularity of this high-level approach to proving machine execution using tuple lookup arguments. The tuple lookup argument of Sublonk may be combined with our Marlin post-processing to build "SubMarlin", and similarly, our tuple lookup argument may be combined with the Plonk post-processing of [CGG$^+$23] to build "Mux-Plonk".

2 Technical Overview

A standard approach to constructing succinct zero knowledge proof systems employs *holography* in which the computation to be proved is encoded within a *computation commitment* in an initial preprocessing phase [CHM$^+$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 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 tuple lookup argument, for proving correspondence between coset evaluations of two polynomials. The second contribution is to show how to compose our new tuple lookup argument with an existing proof system, Marlin [CHM$^+$20], 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 execution of a program, a computation commitment for the executed computation (i.e., the particular sequence of 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 ℓ instructions each of which are represented as a computation over an input state $(inst_{in}, mem_{in})$ and produce an output state $(inst_{out}, mem_{out})$ [1]. 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 $inst \in [0, \ell)$ specifies which of the ℓ 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).

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 [CHM$^+$20, GWC19], 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 is then further encoded as a univariate polynomial $f : \mathbb{F} \to \mathbb{F}$ interpolated over evaluation points where the elements of the vector are set as the evaluations over a canonical ordered subgroup $\mathbb{H} \subseteq \mathbb{F}$. This approach of preprocessing computation commitments as polynomials is used by a popular class of proof systems known as *polynomial interactive oracle proofs* [BFS20]. We will take the same approach modeling a machine commitment as a polynomial specified by evaluations over a specific subgroup.

Say each instruction can be described by a vector of field elements of size m. We can compute a polynomial that represents all ℓ instructions by defining the evaluations of the polynomial over a multiplicative 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 has useful properties. 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 ω:

$$\mathbb{H} = \left\{1, \omega, \omega^2, \ldots, \omega^{\ell m - 1}\right\}.$$

Then we define multiplicative subgroup $\mathbb{V} \leqslant \mathbb{H}$ of size $|\mathbb{V}| = m$ where $\mathbb{V}$ is generated by $\mathbb{V} = \langle \gamma = \omega^{\ell} \rangle$:

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

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

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

In this way, we define 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 instruction set as a machine commitment, our goal is to produce a computation commitment polynomial for the unrolled execution. An unrolled execution consists of applying a number of instruction computations in sequence.

At a high level what we want is to be able to produce a polynomial f interpolated over a subgroup $\mathbb{G}$ (where $|\mathbb{G}| = mn$ and has generator $\mathbb{G} = \langle \mu \rangle$) to represent an unrolled execution of n instructions. Analogous to our encoding of ℓ distinct instructions in the machine commitment polynomial t, we can encode the n executed instructions in f as evaluations over the n cosets of $\mathbb{V}$ in $\mathbb{G}$. More precisely, we want:

$$\forall j \in [0, n) \;\; \exists i \in [0, \ell) \;\; \text{s.t.} \;\; f(\mu^j \mathbb{V}) = t(\omega^i \mathbb{V}).$$

This type of *lookup* relation has been proposed in prior work for the related problem of individual field element

[1]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 $(inst_{in}, mem_{in})$.

evaluations [BCG+18, GW20, ZBK+22, PK22, ZGK+22, GK22, EFG22]. In the individual element lookup, the task is to prove every evaluation of f over $\mathbb{G}$ exists in (i.e., is *looked up* from) the evaluations of the table polynomial t on $\mathbb{H}$. Our first key technical contribution is the construction of a new lookup argument for cosets which we call a *tuple lookup* (Section 2.2).

The tuple lookup will prove the computation commitment f indeed includes valid instructions encoded within its coset evaluations. 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 a two additional hurdles to overcome (Section 2.3). First, f is constructed as a stitching together of the computation commitments 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 commitments results in a valid computation commitment 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 f to f' to recover the global structure required in the Marlin proof system [CHM+20].

Lastly, applying Marlin 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: Tuple Lookup Argument

In the prior section, we described how a tuple lookup argument may be used to prove the wellformedness of an executed computation commitment from a machine commitment. Here we will describe the main ideas behind our construction of a tuple lookup argument.

We begin by sketching the solution of the related element lookup argument problem [GW20] that forms the starting point of our work. Given a lookup polynomial f and a table polynomial t, the task is to prove that each evaluation of f on a subgroup $\mathbb{G} = \langle \mu \rangle$ ($|\mathbb{G}| = n$) matches some evaluation of t on $\mathbb{H} = \langle \omega \rangle$ ($|\mathbb{H}| = \ell$). That is, $\forall j \in [0, n), \exists i \in [0, \ell)$ such that $f(\mu^j) = t(\omega^i)$. The prover commits to a sorted polynomial s that is defined over $\mathbb{K} = \langle \psi \rangle$ (where $|\mathbb{K}| = \ell + n$) in which the evaluations of s over the canonical ordering of $\mathbb{K}$ are set to the sorted vector of f's evaluations over $\mathbb{G}$ and t's evaluations over $\mathbb{H}$ (sorted in the same order that they appear in t). For notational convenience, define $f_j = f(\mu^j)$, $t_i = t(\omega^i)$, and $s_k = s(\psi^k)$. The lookup argument completes using a permutation argument to prove multiset equality of the following two multisets:

$$\{\{(f_j, f_j)\}\}_{j=0}^{n-1} \cup \{\{(t_i, t_{i+1})\}\}_{i=0}^{\ell-1} = \{\{(s_k, s_{k+1})\}\}_{k=0}^{\ell+n-1},$$

by checking the following multiset hash given verifier challenge r [2]:

$$\left(\prod_{j=0}^{n-1} f_j + r \cdot f_j\right) \cdot \left(\prod_{i=0}^{\ell-1} t_i + r \cdot t_{i+1}\right) = \prod_{k=0}^{\ell+n-1} s_k + r \cdot s_{k+1}\,.$$

Intuitively, this tests the lookup requirement because for each f_j, if there exists a corresponding $t_i = f_j$, then (f_j, f_j) in the multiset will correspond to an (s_k, s_{k+1}) in the multiset of the sorted polynomial where $s_k = f_j$ and $s_{k+1} = t_i$. The remainder of the (t_i, t_{i+1}) and (s_k, s_{k+1}) pairs cancel out.

As a strawman, one could build a tuple lookup argument directly from an element lookup argument by encoding a table polynomial for each position of the tuple and performing a multi-table lookup over the position tables. Prior work has shown how to perform such multi-table lookups using a linear combination over the single table lookup argument [GW20]. Unfortunately, this approach incurs verifier costs on the order of the tuple size (i.e., the number of tables) and would not be succinct.

Instead, we construct a tuple lookup protocol where each tuple is encoded over a coset. The core technical challenge of the tuple lookup argument then reduces to building a tuple permutation argument over cosets. Redefine $\mathbb{G}$ to size $|\mathbb{G}| = mn$ containing n cosets of $\mathbb{V} = \langle \gamma \rangle$ with $|\mathbb{V}| = m$. Here, we overview the main ideas for a tuple permutation argument in a simplified setting between two polynomials f and g for cosets of $\mathbb{V}$ over the same group $\mathbb{G}$. Ultimately,

[2]Given a random challenge r, a collision-resistant hash for a vector of elements $(a_0, \ldots, a_n)$ is $H(a_0, \ldots, a_n) = \sum_{i=0}^{n} r^i a_i$. A multiset hash for a multiset S produces a collision-resistant hash for a multiset taking into account multiplicity but not order: $\prod_{s \in S} H(s)$ is a collision-resistant multiset hash is H if a collision-resistant hash to a group.

we want to check:

$$\{\!\{ \left(f(\mu^j), f(\mu^j\gamma), \ldots, f(\mu^j\gamma^{m-1})\right) \}\!\}_{j=0}^{n-1} = \{\!\{ \left(g(\mu^j), g(\mu^j\gamma), \ldots, g(\mu^j\gamma^{m-1})\right) \}\!\}_{j=0}^{n-1}$$

To do this permutation check, we can compute a hash of each tuple and then compare a multiset hash over the tuple hashes. Given a verifier challenges $\beta, r \leftarrow\!\!\$\, \mathbb{F}$:

$$\prod_{j=0}^{n-1}\left(r + \sum_{i=0}^{m-1} \beta^i \cdot f(\mu^j\gamma^i)\right) \stackrel{?}{=} \prod_{j=0}^{n-1}\left(r + \sum_{i=0}^{m-1} \beta^i \cdot g(\mu^j\gamma^i)\right)$$

With this goal, we provide a protocol to prove that the above check holds. The prover interpolates a polynomial S_f (respectively, S_g) that encodes the claimed hash of the coset for each coset evaluation. That is, for all $j \in [0,n)$, the evaluation $S_f(\mu^j\mathbb{V}) = \sum_{i=0}^{m-1} \beta^i \cdot f(\mu^j\gamma^i)$. Given S_f and S_g, we can employ a product test (see preliminaries in Section 3.3) for checking $\prod_{\mu\in\mathbb{G}} r + S_f(\mu) = \prod_{\mu\in\mathbb{G}} r + S_g(\mu)$. Such a check repeats each coset hash evaluation m times in the multiset hash, but even with repetitions the permutation check holds.

All that remains is to prove that S_f (respectively, S_g) is interpolated as claimed. To do this, the prover interpolates polynomials I and B_f (and B_g). Polynomial I encodes the powers of verifier challenge β. Namely, the i^{th} element of each coset is set to evaluate to β^i: for all $j \in [0,n)$ and for all $i \in [0,m)$, the evaluation $I(\mu^j\gamma^i) = \beta^i$. Then we define the induction polynomial B_f which builds up the coset hash summation. Here, the i^{th} element of each coset is set to evaluate to the partial coset hash summation of f normalized by the claimed complete summation encoded in S_f:

$$\forall j \in [0,n),\ \forall i \in [0,m),\ B_f(\mu^j\gamma^i) = \left(\sum_{k=0}^{i} \beta^k f(\mu^j\gamma^k)\right) - i \cdot \frac{S_f(\mu^j\gamma^i)}{m}.$$

If the following polynomial identities hold, then S_f (respectively, S_g) must contain the correct coset hash summation:

- $I(1) = 1$: The first element of I is anchored to equal to 1.
- $(I(\gamma X) - \beta \cdot I(X))(X - \gamma^{m-1}) = 0$ over $\mathbb{V}$: The next element in the coset $\mathbb{V}$ is equal to β times the previous element (excluding the last element). Since the first element was anchored to 1 and the last element is excluded, this sets the evaluations of $\mathbb{V}$ to be equal to the powers of β.
- $(I(\mu X) - I(X)) \cdot Z_{\mu^{n-1}\mathbb{V}}(X) = 0$ over $\mathbb{G}$: The next element in $\mathbb{G}$ is equal to the previous element in $\mathbb{G}$ (excluding the last coset where Z is the "vanishing polynomial" that evaluates to 0 on $\mu^{n-1}\mathbb{V}$). This ensures that the i^{th} element of every coset is the same, so since the powers of β are encoded in coset $\mathbb{V}$, they are encoded in every coset.
- $S_f(\gamma X) = S_f(X)$ over $\mathbb{G}$: Every element of a coset contains the same claimed summation for the coset hash.
- $B_f(\gamma X) = B_f(X) + I(\gamma X) \cdot f(\gamma X) - \frac{S_f(X)}{m}$ over $\mathbb{G}$: The induction statement requires that the next element in the partial summation sums the previous partial summation with the contribution of the next element in the coset, namely $\beta^k \cdot f(\mu^j\gamma^k)$, and again subtracts out the normalized claimed summation.

These polynomial identities are checked via standard zero test protocols (see preliminaries Section 3.3). Due to our insight of encoding tuples in cosets, we are able to take advantage of the algebraic structure to succinctly prove the multiset hash evaluation.

Section 4 presents this protocol as a starting point and extends it to support permutation arguments (and lookup arguments) over different groups, e.g., f, t, and u are defined over different groups in the lookup example above. These extensions are relevant for our machine execution application because while the number of instructions ℓ in the machine commitment is fixed, the number of unrolled instructions n is variable and will determine a group to use for the lookup argument on-the-fly. We do not want our protocol to be fixed ahead of time to only one choice of n.

2.3 Contribution: Marlin for Machine Execution

We described our high level strategy of proving the wellformedness of the executed computation commitment through a tuple lookup argument using the machine commitment as a table of valid instructions. There are two further challenges to overcome to apply a proof system like Marlin [CHM+20] to the 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 statement for the computation commitment is not succinct.

Recovering global structure of the computation commitment. Marlin is a proof system for rank-1 constraint satisfiability (R1CS). An R1CS computation is defined by coefficient matrices $(A, B, C) \in \mathbb{F}^m \times \mathbb{F}^m$. Given a statement

$x \in \mathbb{F}^k$ specifying a partial assignment to variables and witness $w \in \mathbb{F}^{m-k}$ specifying assignment for the remainder of variables, an R1CS computation is satisfied if $Az \circ Bz = Cz$ for $z \leftarrow (x, w) \in \mathbb{F}^m$. In Marlin, the computation commitment consists of polynomials row_M, col_M, val_M that encode a description for each coefficient matrix $M \in [A, B, C]$. Take for example row_A which is defined via evaluations over $\mathbb{V}$. Given a canonical mapping ϕ between $\mathbb{V}$ and the non-zero elements of A, row_A is defined to set the evaluation of $row_A(\gamma^i) = \gamma^j$ where j is the row of element $\phi(\gamma^i)$ in A.

Notice that this computation commitment has global structure in that it is defined over group $\mathbb{V}$ that has size equal to the dimension of the matrices. In an executed computation of n instructions where each instruction is defined by an R1CS instance of size m (over $\mathbb{V}$), to apply Marlin, we need computation commitment $row_{A'}$ for matrix A' that represents the executed computation of size mn over group $\mathbb{G}$ ($|\mathbb{G}| = mn$). However, if we were to stitch together the evaluations from the computation commitments for the individual instructions, the semantics of the row mapping would be lost: all evaluations would be in $\mathbb{V}$.

We observe that, in fact, not all semantics are lost. The desired matrix A' for the executed computation of applying n instructions with matrices $A_0, \ldots, A_{n-1}$ consists of each instruction matrix offset along the diagonal (row indices denoted in blue):

$$A' = \left[\begin{array}{ccc} A_0 & 0 & \dots \\ 0 & \ddots & 0 \\ \vdots & 0 & A_{n-1} \end{array}\right] \begin{array}{c} \mathbb{V} \\ \vdots \\ \mu^{n-1}\mathbb{V} \end{array}$$

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 instruction j need to be offset by μ^j to instead map to coset $\mu^j\mathbb{V}$. Consider polynomial $row'_{A'}$ defined over $\mathbb{G}$ that consists of the stitched together computation commitments where each coset $\mu^j\mathbb{V}$ evaluates to the row mapping of instruction j in $\mathbb{V}$, and take polynomial $row_{A'}$ as the correct description of A' where each coset $\mu^j\mathbb{V}$ evaluates to the row mapping of instruction j in $\mathbb{V}$ offset to $\mu^j\mathbb{V}$. We provide a protocol for proving the correct offset shift of $row_{A'}$ with respect to $row'_{A'}$. 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 $row_{A'}$ 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 μ 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 μ.
- $row_{A'}(X) = row'_{A'}(X) \cdot s(X)$ over $\mathbb{G}$: This directly checks the offset shift between $row_{A'}$ and $row'_{A'}$.

With this protocol, we show that we can recover the global structure of the computation commitment to allow application of Marlin to an on-the-fly executed computation commitment generated from the machine commitment. Section 5 provides the full details of how Marlin's computation commitments may be recovered: col_M follows the same strategy as row_M, while val_M takes a different approach.

Compressing the 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: $[(inst_{in,j}, mem_{in,j}, inst_{out,j}, mem_{out,j})]_{j=0}^{n-1}$. 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 $(inst_{out,j}, mem_{out,j}) = (inst_{in,j+1}, mem_{in,j+1})$ for all $j \in [0, n-1)$. Then the verifier only need hold the starting state $(inst_{in,0}, mem_{in,0})$ and the ending state $(inst_{out,n-1}, mem_{out,n-1})$.

In Marlin, the statement and witness for the unrolled computation are encoded together in a polynomial z defined over $\mathbb{G}$ where the statement and witness for each instruction are encoded within a coset of $\mathbb{V}$ in $\mathbb{G}$: for instruction $j \in [0, n-1)$, the evaluation of $z(\mu^j\mathbb{V}) = (inst_{in,j}, mem_{in,j}, inst_{out,j}, mem_{out,j}, w_j)$. More precisely, we consider two further multiplicative subgroups $\mathbb{V}_{in} \leqslant \mathbb{V}_x \leqslant \mathbb{V}$, where $\mathbb{V}_x = \langle\psi\rangle$ encodes the statement within a further subgroup

$\mathbb{V}_{in}$ and its single coset $\mathbb{V}_{out} = \psi\mathbb{V}_{in}$:

$$\forall j \in [0,n),\ \left[z(\mu^j\mathbb{V}_{in}) = [inst_{in,j}, mem_{in,j}]\right]_{j=0}^{n-1} \quad \left[z(\mu^j\mathbb{V}_{out}) = [inst_{out,j}, mem_{out,j}]\right]_{j=0}^{n-1}$$

Thus, proving consistency of intermediate program states reduces to proving equality between certain cosets. Namely, for $j \in [1,n)$, $z(\mu^j\mathbb{V}_{in}) = z(\mu^{j-1}\mathbb{V}_{out})$. 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 $\mathbb{G}_{in} = \bigcup_{j=0}^{n-1}\mu^j\mathbb{V}_{in}$. Unfortunately, we cannot directly apply a zero test to this polynomial identity as the vanishing polynomial for $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$ does not have a succinct form so the verifier cannot compute it on its own. Nevertheless, we observe that the vanishing polynomial for $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$ consists of the product of vanishing polynomials for cosets of $\mathbb{V}_{in}$ each of which have succinct form. As such, in Section 5, we provide a protocol for the prover to provide and prove the wellformedness of a claimed vanishing polynomial of $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$ that builds up the product in a manner similar to the product test (see preliminaries Section 3.3).

This completes the application of Marlin to the unrolled computation commitment. All of the polynomial identities can be checked succinctly and with zero-knowledge resulting in the first zero-knowledge protocol for machine execution that does not rely on recursive proving techniques.

3 Preliminaries

3.1 Polynomial Notation

Let λ be our security parameter. For a positive number n, let $[n]$ denote the set $\{0,\ldots,n-1\}$. For polynomials $f_1,\ldots,f_n \in F[X]$ and some multiplicative group or coset $\mathbb{V} = \langle\gamma\rangle$ of size m, let $f_i(\mathbb{V})$ denote expanding all f_i's evaluation points $f_i(\gamma^0),\ldots,f_i(\gamma^{m-1})$. Alternatively, we also use $[f_i(\gamma^j)]_{j\in[m]}$ to expand evaluation points of f_i. In general, we use $f(\mathbb{V})$ to expand evaluations in domain $\mathbb{V}$, $[\cdot]$ as an operator that expands subscripts and integers, and $(\cdot)$ to expand subscripts into an ordered tuple. Let $\{\!\{\cdot\}\!\}$ denote a multiset, i.e., a set with repetitions allowed.

Define $\mathbb{F}$ to be a scalar field of large prime order p. Let $\mathbb{H} = \langle\omega\rangle$ and $\mathbb{V} = \langle\gamma\rangle$ be multiplicative subgroups of $\mathbb{F}^*$ of size nm and m, respectively, and both have order of a power of 2 in order to perform FFT (size can be adapted with padding). In particular, we have $\mathbb{V}$ to be a subgroup of $\mathbb{H}$, denoted $\mathbb{V} \leqslant \mathbb{H}$.

In our approach for constructing a tuple lookup, we will use polynomials from different groups. In this setting, we define multiplicative subgroups $\mathbb{H}_0 = \langle\psi\rangle$ of size d_0m and $\mathbb{H}_1 = \langle\mu\rangle$ of size d_1m. We define $\{\psi^i\mathbb{V}_0\}_{i\in\mathbb{Z}_{d_0}}$ to be d_0 cosets of $\mathbb{V}_0$ in $\mathbb{H}_0$ and $\{\mu^i\mathbb{V}_0\}_{i\in\mathbb{Z}_{d_1}}$ to be d_1 cosets of $\mathbb{V}_0$ in $\mathbb{H}_1$. We require that all multiplicative groups we use are FFT-friendly and have smooth sizes [BCG+13]. That is, we want $2^L|\mathsf{p}-1$ 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. Then given any $m = 2^i$ for some i, we can always let $d_0 = 2^j, d_1 = 2^k$ such that $i+j, i+k \leq L$. Many common curves are FFT-friendly including BN382 and BLS12-381 ($GF(q)$) [AHG22].

Claim 1. *For multiplicative subgroups $\mathbb{H} = \langle\omega\rangle, \mathbb{V} = \langle\gamma\rangle$ and $\mathbb{V} \leqslant \mathbb{H}$, $[\omega^i\mathbb{V}_0]_{i\in\mathbb{Z}_n}$ consists of n distinct cosets of $\mathbb{V}$ in $\mathbb{H}$, or equivalently, $\cup_i(\omega^i\mathbb{V}_0) = \mathbb{H}$ and $\omega^i\mathbb{V}_0 \cap \omega^j\mathbb{V}_0 = \emptyset, \forall i,j \in \mathbb{Z}_n$.*

Proof. When $n = 1$, this claim is trivially true. We can move to the case where $n > 1$ and $|\mathbb{V}| < |\mathbb{H}|$. Since the coset decomposition is unique, it is sufficient to show that $\omega^i\mathbb{V}_0 \neq \omega^j\mathbb{V}_0$ for $i,j \in \mathbb{Z}_n$ and $i \neq j$. Suppose by contradiction, $\omega^i\mathbb{V}_0 = \omega^j\mathbb{V}_0$. Then it must be the case that $\omega^{i-j} \in \mathbb{V}$, i.e., $\omega^{i-j} = \gamma^k$ for some k. In particular, we have $\omega = \gamma^k$ and γ^k generates $\mathbb{H}$. This is a contradiction since $|\mathbb{V}| < |\mathbb{H}|$. As a result, for $i \in \mathbb{Z}_n$, $\cup_i\omega^i\mathbb{V}_0 = \mathbb{H}$ since $N = nm$ and all $\omega^i\mathbb{V}_0$ are disjoint. □

Let $\mathbb{F}^{\leq d}[X_1,\ldots,X_\mu]$ be the set of μ-variate polynomials in indeterminate $X_1,\ldots,X_\mu$ with coefficients in $\mathbb{F}$ with degree less than or equal to d. In this work, we will primarily be working with univariate polynomials, denoted $\mathbb{F}^{\leq d}[X]$. If the degree bound of the polynomial is not specified, we may drop the superscript. 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 vanishes on $S\setminus\{x\}$ and has $L_{x,S}(x) = 1$. For a group or coset $\mathbb{V}$, it has the sparse form $L_{x,\mathbb{V}}(X) = \frac{c_x Z_{\mathbb{V}}(X)}{X-x}$ and can be evaluated at any point in $\log(\mathbb{V})$ time. 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 all $c_x, \forall x \in \mathbb{V}$ can be precomputed in $O(|\mathbb{V}|)$ time.

Claim 2. *For a coset $\omega^i\mathbb{V}$ of order m, its vanishing polynomial has the succinct form $Z_{\omega^i\mathbb{V}}(X) = X^m - \omega^{im}$.*

```
$\langle \mathsf{P}(pp, x, w, aux_{\mathsf{P}}) \leftrightarrow \mathsf{V}(vp, x, aux_{\mathsf{V}}) \rangle_n$
Parse oracles $[(f_i, d_i)]_{i=0}^{\ell} \leftarrow vp$ ; $[(f_i, d_i)]_{i=\ell}^{\ell+m} \leftarrow x$
$vp \leftarrow [d_i]_{i=0}^{\ell}$ ; $x \leftarrow [d_i]_{i=\ell}^{\ell+m}$ ; $ctr \leftarrow \ell + m$
$(st_{\mathsf{P}}, m) \leftarrow\!\!\$\ \mathsf{P.Init}(pp, x, w, aux_{\mathsf{P}})$
$st_{\mathsf{V}} \leftarrow\!\!\$\ \mathsf{V.Init}(vp, x, aux_{\mathsf{V}})$
Repeat $n$ times:
    $(st_{\mathsf{V}}, m, dec) \leftarrow\!\!\$\ \mathsf{V.Round}^{\textsc{Poly}}(st_{\mathsf{V}}, m)$
    If $dec = \texttt{accept}$ then return $m$
    $(st_{\mathsf{P}}, m) \leftarrow\!\!\$\ \mathsf{P.Round}(st_{\mathsf{P}}, m)$
    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

Oracle $\textsc{Poly}(g, [i_j]_j^m, \alpha)$
Require $g \in \mathbb{F}^1[X_1, \ldots, X_m]$
Require $\forall_j^m\ i_j \in [0, ctr]$
Return $g(f_{i_0}(\alpha), \ldots, f_{i_{m-1}}(\alpha))$
```

Figure 2: 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.

Proof. $\forall \gamma^j \in \mathbb{V}$, $Z_{\omega^i \gamma^j} = (\omega^i \gamma^j)^m - \omega^{im} = 0$ since the generator γ has order m and $\gamma^{jm} = 1$. Since $X^m - \omega^{im}$ is of degree m and $|\omega^i \mathbb{V}| = m$, it is the unique monic polynomial of degree at most m that is zero everywhere in $\omega^i \mathbb{V}$, i.e, it is the vanishing polynomial of $\omega^i \mathbb{V}$. In this case it is succinct, and can be evaluated in $O(\log(m))$ field operations. □

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 [CBBZ22] 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+20] is defined over triples $(\mathfrak{i}, x, w)$ where $\mathfrak{i}$ 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 Π is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Index}, \mathsf{P}, \mathsf{V})$ for an indexed relation R. The algorithms are defined as follows:

- $gp \leftarrow\!\!\$\ \mathsf{Setup}(\lambda)$: The setup algorithm takes a security parameter λ and outputs the global parameters gp.
- $(vp, pp) \leftarrow \mathsf{Index}(gp, \mathfrak{i})$: The deterministic indexing algorithm takes as input the index $\mathfrak{i}$ 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.
- $\mathsf{P}(pp, x, w) \leftrightarrow \mathsf{V}(vp, x)$: 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: $(st_{\mathsf{P}}, m_{\mathsf{out}}) \leftarrow\!\!\$\ \mathsf{P.Round}(st_{\mathsf{P}}, m_{\mathsf{in}})$. The verifier algorithm additionally outputs a decision in $\{\texttt{accept}, \texttt{cont}\}$: $(st_{\mathsf{V}}, m_{\mathsf{out}}, dec) \leftarrow\!\!\$\ \mathsf{V.Round}(st_{\mathsf{V}}, m_{\mathsf{in}})$. If the verifier accepts, the output message is parsed as $m_{\mathsf{out}} \in \{0, 1\}$ indicating whether verification succeeded. The state for the prover algorithm is initialized with the prover parameters, statement, and witness, $(st_{\mathsf{P}}, m_{\mathsf{out}}) \leftarrow\!\!\$\ \mathsf{P.Init}(pp, x, w)$, and the verifier algorithm is initialized with the verifier parameters and statement, $st_{\mathsf{V}} \leftarrow\!\!\$\ \mathsf{V.Init}(vp, x)$. 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, the setup algorithm outputs the prover and verifier parameters directly.

```
Game $\textsc{Sound}^{\mathcal{A}}_{\Pi,\mathsf{R},\mathsf{X},n}(\lambda)$
$gp \leftarrow\!\!\$\ \Pi.\mathsf{Setup}(\lambda)$
$(i, x, st_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_1(gp)$
$(vp, pp) \leftarrow \Pi.\mathsf{Index}(gp, i)$
$w \leftarrow\!\!\$\ \mathsf{X}^{\mathcal{A}_1,\mathcal{A}_2}(gp, i, x)$
Return $\bigwedge \begin{pmatrix} \langle \mathcal{A}_2(pp, x, \perp, st_{\mathcal{A}}) \leftrightarrow \Pi.\mathsf{V}(vp, x)\rangle_n \\ (i, x, w) \notin \mathsf{R} \end{pmatrix}$
```

```
Game $\mathrm{ZK}^{\mathcal{A},b}_{\Pi,\mathsf{R},\mathsf{S},n}(\lambda)$
$gp_1 \leftarrow\!\!\$\ \Pi.\mathsf{Setup}(\lambda)$ ; $(gp_0, st_{\mathsf{S}}) \leftarrow\!\!\$\ \mathsf{S}.\mathsf{Setup}(\lambda)$
$(i, x, w, st_{\mathcal{A}}) \leftarrow\!\!\$\ \mathcal{A}_1(gp_b)$
$(vp, pp) \leftarrow \Pi.\mathsf{Index}(gp_b, i)$
$vw_1 \leftarrow \mathsf{View}\langle \Pi.\mathsf{P}(pp, x, w, \perp) \leftrightarrow \Pi.\mathsf{V}(vp, x, \perp)\rangle_n$
$vw_0 \leftarrow \mathsf{S}.\mathsf{SimView}(pp, x, st_{\mathsf{S}})$
Return $\bigwedge \begin{pmatrix} \mathcal{A}_2(vw_b, st_{\mathcal{A}}) \\ (i, x, w) \in \mathsf{R} \end{pmatrix}$
```

Figure 3: Knowledge soundness (left) and zero knowledge (right) security games for interactive argument systems.

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

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

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

Knowledge soundness. 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 $(i, x, w) \in \mathsf{R}$. This is modeled via an extractor algorithm X that can learn the witness given oracle access to the prover. Here, oracle access X^{P} 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 $\textsc{Sound}$ in Figure 3. An n-round protocol Π 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 λ: $\mathsf{Adv}^{\mathrm{sound}}_{\Pi,\mathsf{R},\mathsf{X},n,\mathcal{A}}(\lambda) = \Pr\left[\textsc{Sound}^{\mathcal{A}}_{\Pi,\mathsf{R},\mathsf{X},n}(\lambda) = 1\right]$.

Zero knowledge. An argument system is *zero-knowledge* 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 3 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 messages. An n-round protocol Π 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 λ:

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

Public coin. An interactive argument is considered *public coin* if all of the verifier messages are uniform random samples from some predefined challenge space. Importantly, the verifier messages should not depend on the results of other 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 $f \in \mathbb{F}^{\leq d}[X]$, we denote $[\![f]\!]^{\leq d}$ to be its oracle (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 2. Our treatment is with respect to univariate polynomial oracles, though can be readily extended to the multivariate setting (see [CBBZ22] for a multivariate treatment). We additionally require the following properties for polynomial oracles:

Oracle degree admissibility. 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.

Virtual oracles for linear combinations. 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+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 2).

Domain-restricted admissibility. We define a further restricted form of polyIOPs following the treatment of [CBBZ22] (which defines a restriction of multivariate polyIOPs to sum-checks over the boolean hypercube). Every polynomial oracle provided is accompanied by an *evaluation domain*. 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+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+20, BFS20, CBBZ22].

Zero-knowledge compiler. A sound domain-admissible polyIOP whose witness contains only specified evaluations of oracle polynomials on the restricted evaluation domain compiles to a zero-knowledge sound polyIOP by careful application of a bounded independence argument to the polynomial oracles [BCR+19, CHM+20]. The compiler works as follows. Define $\mathbb{G}$ as the union of all oracle restricted domains and $\mathbf{b}$ as the maximum query bound to a single oracle (including derived virtual oracles). We construct a new prover $\hat{\mathsf{P}}$ such that for every oracle f_i with restricted domain $\mathbb{H}_i$ that P sends, $\hat{\mathsf{P}}$ samples $\mathbf{b}$ random points in $\mathbb{F} \setminus \mathbb{G}$, interpolates and sends polynomial $\hat{f}_i \in \mathbb{F}^{\leq d_i + \mathbf{b}}[X]$ that agrees with f_i over $\mathbb{H}_i$. The compilation does not affect completeness and soundness since $\hat{f}_i$ agrees with f_i over the evaluation domain. At the same time, $\hat{f}_i$ reveals no information up to $\mathbf{b}$ queries outside $\mathbb{G}$ since it is $\mathbf{b}$-wise independent over $\mathbb{F} \setminus \mathbb{G}$. Since all verifier challenges fall outside $\mathbb{G}$, all messages sent by $\hat{\mathsf{P}}$ appear uniformly random and can be simulated. The simulator selects random verifier challenges ahead of time and provides random polynomial oracles of the appropriate blinded degree. Since the simulator selects the random challenges ahead of time, it sets the evaluations of polynomial oracles for the verifier challenge evaluation points to meet the verification checks. Note that this compiler bumps the degree of polynomials by $\mathbf{b}$ and increase the soundness errors by constant number of $\frac{\mathbf{b}}{|\mathbb{F} \setminus \mathbb{G}|}$.

Oracle instantiation compiler. 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+20, Theorem 8.1-8.4].

Non-interaction compiler. Further, if the polyIOP is public-coin and the above hold, it can be compiled to a zero-knowledge 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+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, and product check—that we will make extensive use of in building higher level protocols. We note that all of these protocols are domain admissible and may be compiled to zero-knowledge.

Zero test. We provide a polyIOP for the relation $\mathsf{R}_{\mathsf{zero}}$ that we refer to as ZeroTest in Figure 4 following [CHM+20] and [GWC19]. The protocol allows for testing whether a polynomial $F(X)$ evaluates to zero over a domain $\mathbb{K}$. For generality, $F(X)$ is defined with respect to polynomial oracles $[\![f_i]\!]_i^k$ that a verifier may have access to. To ensure domain admissibility, we also specify a restricted domain $\mathbb{G}$ 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+20]. For simplicity, we will use the following shorthand for applying the zero test when the rest of the parameters are clear from context: $\mathsf{ZeroTest}(\mathbb{K}, \mathbb{G})$, specifying the test will be evaluated over domain $\mathbb{K}$, with the

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

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

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

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

$\mathsf{R}_{\mathsf{sum}} = \{\bot, (\mathbb{K} = \langle \omega \rangle, \mathbb{G}, \llbracket f \rrbracket, H), f : \sum_{x \in \mathbb{K}} f(x) = H\}$

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

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

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

Figure 5: PolyIOP for checking that the sum of evaluations of $f(X)$ over domain $\mathbb{K}$ is equal to H.

randomness sampled outside of the set $\mathbb{G}$.

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 5. 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+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 $\mathsf{SumCheck}(\mathbb{K}, \mathbb{G})$ to denote the test will be evaluated over $\mathbb{K}$ and the randomness will be sampled outside $\mathbb{G}$.

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 6. The protocol satisfies completeness and has soundness advantage at most $\frac{3|\mathbb{K}|+B}{|\mathbb{F} \setminus \mathbb{G}|}$ where B is the degree bound of f [GWC19].

Cross-group product check. Lastly, looking forward, our tuple 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 7.

Theorem 3. *$\mathsf{XGProductCheck}$ for $\mathsf{R}_{\mathsf{prod}}$ satisfies perfect completeness and for any adversary $\mathcal{A}$ against knowledge soundness, we provide an extractor X such that*

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

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.

Completeness. 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.

Knowledge soundness. 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 $i \in [|\mathbb{H}_0|]$, $T_0(\psi^i) =$

$\mathsf{R_{prod}} = \left\{\bot, (\mathbb{K} = \langle\omega\rangle, \mathbb{G}, [\![f]\!]), f : \prod_{x\in\mathbb{K}} f(x) = 1\right\}$

$\mathsf{ProductCheck.P}(\bot, (\mathbb{K}, \mathbb{G}, [\![f]\!]), f) \leftrightarrow \mathsf{ProductCheck.V}(\bot, (\mathbb{K}, \mathbb{G}, [\![f]\!]))$

(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 $\mathsf{ZeroTest}(\mathbb{K}, \mathbb{G})$ to prove $L_{1,\mathbb{K}}(X)(T(X) - 1) = 0$ over $\mathbb{K}$.

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

Figure 6: PolyIOP for checking that the product of evaluations of $f(X)$ over domain $\mathbb{K}$ is equal to 1.

$\mathsf{R_{xgprod}} = \left\{\bot, (\mathbb{H}_0 = \langle\psi\rangle, \mathbb{H}_1 = \langle\mu\rangle, [\![f_0]\!], [\![f_1]\!]), (f_0, f_1) : \prod_{x\in\mathbb{H}_0} f_0(x) = \prod_{x\in\mathbb{H}_1} f_1(x)\right\}$

$$\mathsf{XGProductCheck.P}\begin{pmatrix} \bot, \\ (\mathbb{H}_0, \mathbb{H}_1, [\![f_0]\!], [\![f_1]\!]), \\ (f_0, f_1) \end{pmatrix} \leftrightarrow \mathsf{XGProductCheck.V}\begin{pmatrix} \bot, \\ (\mathbb{H}_0, \mathbb{H}_1, [\![f_0]\!], [\![f_1]\!]) \end{pmatrix}$$

(1) P computes and sends product polynomials T_0, T_1 and proves that they are well-formed.

(a) P interpolates and sends polynomials T_0 over $\mathbb{H}_0$ and T_1 over $\mathbb{H}_1$ such that:

$$\left[T_0(\psi^i) = \prod_{k=i}^{|\mathbb{H}_0|-1} f_0(\psi^{|\mathbb{H}_0|-k-1})\right]_{i=0}^{|\mathbb{H}_0|-1} \qquad \left[T_1(\mu^i) = \prod_{k=i}^{|\mathbb{H}_1|-1} f_1(\mu^{|\mathbb{H}_1|-k-1})\right]_{i=0}^{|\mathbb{H}_1|-1}.$$

(b) P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $(X - \psi^{|\mathbb{H}_0|-1})(T_0(\psi X)f_0(X) - T_0(X)) = 0$ over $\mathbb{H}_0$.

(c) P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $(X - \mu^{|\mathbb{H}_1|-1})(T_1(\mu X)f_1(X) - T_1(X)) = 0$ over $\mathbb{H}_1$.

(d) P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $L_{\psi^{|\mathbb{H}_0|-1},\mathbb{H}_0}(X)(T_0(X) - f_0(X)) = 0$ over $\mathbb{H}_0$.

(e) P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $L_{\mu^{|\mathbb{H}_1|-1},\mathbb{H}_1}(X)(T_1(X) - f_1(X)) = 0$ over $\mathbb{H}_1$.

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

Figure 7: 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 $\mathbb{H}_1$

$\prod_{k=i}^{|\mathbb{H}_0|-1} f_0(\psi^{|\mathbb{H}_0|-k-1})$, and in particular, $T_0(1) = \prod_{k=0}^{|\mathbb{H}_0|-1} f_0(\psi^k)$. Respectively, by (1c) and (1e), we have that $T_1(1) = \prod_{k=0}^{|\mathbb{H}_1|-1} f_1(\mu^k)$. 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 $\mathsf{X_{ZeroTest}}$ to retrieve f_0, f_1. By the soundness bounds of each of the zero-tests, we complete the argument.

□

4 TuPlookup: A Lookup Argument for Tuples

In this section, we present TuPlookup, defined by the following relation:

$$\mathsf{R_{tl}} = \left\{\bot, ([\![f]\!], [\![t]\!]), (f, t) : \left\{(f(\psi^i\mathbb{V}))\right\}_{i\in[d_0]} \subseteq \left\{(t(\mu^i\mathbb{V}))\right\}_{i\in[d_1-1]}\right\}.$$

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 tuples in f is a subset of the set of coset tuples in t.

We build up to the argument through a series of steps. First, we construct a tuple permutation argument TuPerm that shows the coset tuples 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 tuples for polynomials even when the coset tuples are evaluated over different groups on the different polynomials. This will be important to build our desired tuple lookup argument where the lookup polynomial f and the table polynomial t may be of different sizes and are defined over different groups $\mathbb{H}_0, \mathbb{H}_1$.

4.1 Tuple Permutation

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

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

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 tuple setting: constructing a multiset hash where each element in the set is a tuple (or looking forward, a hash encoding of a tuple).

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 8.

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

Theorem 4. *TuPerm for R_{tp} (Figure 8) satisfies perfect completeness and for any adversary $\mathcal{A}$ against knowledge soundness, we provide an extractor X such that $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{TuPerm},\mathsf{R}_{\mathsf{tp}},\mathsf{X},4,\mathcal{A}}(\lambda) \le \frac{16|\mathbb{H}|+2|\mathbb{V}|+1}{|\mathbb{F}\setminus\mathbb{H}|}$.*

Proof. We argue completeness and knowledge soundness separately.

Completeness. 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 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.

Knowledge soundness. We bound the advantage of adversary $\mathcal{A}$ by bounding the advantage of each of a series of game hops [BR06]. We define $\mathrm{G}_0 = \mathrm{SOUND}^{\mathcal{A}}_{\mathsf{TuPerm},\mathsf{R}_{\mathsf{tp}},\mathsf{X},4}(\lambda)$. The inequality above follows from the following claims that we will justify:

(1) $|\Pr[\mathrm{G}_0 = 1] - \Pr[\mathrm{G}_1 = 1]| \le \frac{2|\mathbb{H}|+2|\mathbb{V}|+1}{|\mathbb{F}\setminus\mathbb{H}|}$

(2) $|\Pr[\mathrm{G}_1 = 1] - \Pr[\mathrm{G}_2 = 1]| \le \frac{6|\mathbb{H}|}{|\mathbb{F}\setminus\mathbb{H}|}$

(3) $|\Pr[\mathrm{G}_2 = 1] - \Pr[\mathrm{G}_3 = 1]| \le \frac{8|\mathbb{H}|}{|\mathbb{F}\setminus\mathbb{H}|}$

(4) $\Pr[\mathrm{G}_3 = 1] = 0$

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: For the first step, we show that polynomial I encodes powers of β over cosets, i.e., $I(\omega^i\gamma^j) = \beta^j, \forall i \in [n], j \in [m]$.

- $L_{1,\mathbb{V}}(X)(I(X)-1) = 0$ over $\mathbb{V}$ with advantage $\frac{|\mathbb{H}|+|\mathbb{V}|}{|\mathbb{F}\setminus\mathbb{H}|}$: Checks base case that $I(1) = 1$.
- $(I(\gamma X) - \beta\cdot I(X))(X-\gamma^{m-1}) = 0$ over $\mathbb{V}$ with advantage $\frac{|\mathbb{H}|+1}{|\mathbb{F}\setminus\mathbb{H}|}$: Checks inductive step that for all $j \in [0,m)$, $I(\gamma^j) = \beta\cdot I(\gamma^{j-1}) = \beta^j$.
- $(I(X) - I(\omega X))Z_{\omega^{n-1}\mathbb{V}}(X) = 0$ over $\mathbb{H}$ with advantage $\frac{|\mathbb{H}|+|\mathbb{V}|}{|\mathbb{F}\setminus\mathbb{H}|}$: Checks $I(\omega^i\mathbb{V}) = I(\omega^j\mathbb{V}), \forall i,j$. Since from the previous check we have that $I(\omega^0\mathbb{V})$ encodes the powers-of-β, this check ensures that every coset $\omega^i\mathbb{V}$ encodes the powers-of-β.

G_1 employs the zero test extractor $\mathsf{X}_{\mathsf{ZeroTest}}$ 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.

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

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

(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-β randomness, β^j:

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

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

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

(e) P and V engage in $\mathsf{ZeroTest}(\mathbb{H},\mathbb{H})$ 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 $\mathsf{ZeroTest}(\mathbb{H},\mathbb{H})$ to prove every coset of $\mathbb{V}$ in $\mathbb{H}$ encodes some constant value: $S_f(\gamma X) = S_f(X)$, $S_g(\gamma X) = S_g(X)$ over $\mathbb{H}$

(3) P computes and sends the induction polynomials $B_f(X), B_g(X)$ 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:

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

(b) P and V engage in $\mathsf{ZeroTest}(\mathbb{H},\mathbb{H})$ to prove induction
$B_f(\gamma X) = (B_f(X) + I(\gamma X)\cdot f(\gamma X)) - \frac{S_f(X)}{m}$,
$B_g(\gamma X) = (B_g(X) + I(\gamma X)\cdot g(\gamma X)) - \frac{S_g(X)}{m}$ 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 $r \in \mathbb{F} \setminus \mathbb{H}$

(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 $\mathsf{ZeroTest}(\mathbb{H},\mathbb{H})$ to prove $q(X)(S_g(X)+r) = S_f(X)+r$ over $\mathbb{H}$

(d) P and V engage in $\mathsf{ProductCheck}(\mathbb{H},\mathbb{H})$ to prove $\prod_{x\in\mathbb{H}} q(x) = 1$

Figure 8: TuPerm: Tuple permutation protocol.

$$\mathsf{R}_{k\text{-tp}} = \left\{ \left(\begin{array}{l} \bot, \\ [\![f_i]\!], [\![g_i]\!]_{i\in[k]} \\ [f_i, g_i]_{i\in[k]} \end{array} \right) : \begin{array}{l} \{\!\{((f_i(\omega^j\gamma^l)_{i\in[k]})_{l\in\mathbb{Z}_m})\}\!\}_{j\in[n]} \\ \quad = \{\!\{((g_i(\omega^j\gamma^l)_{i\in[k]})_{l\in\mathbb{Z}_m})\}\!\}_{j\in[n]} \end{array} \right\}$$

k-$\mathsf{TuPerm.P}(\bot, [\![f_i]\!], [\![g_i]\!]_{i\in[k]}, [f_i, g_i]_{i\in[k]}) \leftrightarrow k$-$\mathsf{TuPerm.V}(\bot, [\![f_i]\!], [\![g_i]\!]_{i\in[k]})$

(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 $f(X) = \sum_{i\in[k]} f_i(X)\alpha^i$ and $g(X) = \sum_{i\in[k]} g_i(X)\alpha^i$

(2) P and V engage in TuPerm to prove $\{\!\{(f(\omega^i\mathbb{V}))\}\!\}_{i\in[n]} = \{\!\{(g(\omega^i\mathbb{V}))\}\!\}_{i\in[n]}$

Figure 9: k-TuPerm: Tuple permutation enforcing the same permutation across k pairs.

Claim 2: In the second claim, we argue that S_f and S_g encode hashes of each coset of f and g. First, in step 2(b) and 3(b) we check that for $p \in \{f, g\}$:

- $S_p(\gamma X) = S_p(X)$ over $\mathbb{H}$ with advantage $\frac{|\mathbb{H}|}{|\mathbb{F}\setminus\mathbb{H}|}$: Checks $\forall i \in [n]$, $S_p(\omega^i\mathbb{V})$ encodes some constant value.
- $B_p(\gamma X) = (B_p(X) + I(\gamma X)\cdot p(\gamma X)) - \frac{S_p(X)}{m}$ with advantage $\frac{2|\mathbb{H}|}{|\mathbb{F}\setminus\mathbb{H}|}$: Checks $\forall i \in [n]$, $\sum_{j\in[m]} I(X)p(X) = \sum_{j\in[m]} \beta^j p(X) = \sum_{j\in[m]} \frac{S_p(X)}{m}$. Since $S_p(X)$ is constant over $\omega^i\mathbb{V}$, we have that $S_p(\omega^i\mathbb{V})$ must encode the hash: $S_p(\omega^i\mathbb{V}) = \sum_{j\in[m]} \beta^j p(\omega^i\gamma^j)$.

Again, G_2 employs the zero test extractor $\mathsf{X}_{\mathsf{ZeroTest}}$ 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 g.

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 $\frac{2|\mathbb{H}|}{|\mathbb{F}\setminus\mathbb{H}|}$: Checks q is the correct quotient polynomial of S_f and S_g.
- $\prod_{x\in\mathbb{H}} q(x) = 1$ with advantage $\frac{4\mathbb{H}}{|\mathbb{F}\setminus\mathbb{H}|}$: 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_3 employs the zero test extractor $\mathsf{X}_{\mathsf{ZeroTest}}$ and the product check extractor $\mathsf{X}_{\mathsf{ProductCheck}}$ 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 π 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_3, 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 $\frac{|\mathbb{V}|}{|\mathbb{F}\setminus\mathbb{H}|}$ 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 $\mathsf{X}_{\mathsf{ZeroTest}}$ to retrieve and output f, g. By Claim 2, in G_2, if the verifier succeeds, $\mathsf{X}_{\mathsf{ZeroTest}}$ always succeeds and so our extractor will always succeed.

□

k-Tuple permutation. We can extend the permutation argument to work across k pairs of polynomials (f_i, g_i) for

$i \in [0, k)$ checking that the same tuple permutation 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. The protocol is given in Figure 9. For polynomials $[f_i]_{i\in[k]}$, we use the following shorthand:

$$((f_i(\omega^j\gamma^l)_{i\in[k]})_{l\in\mathbb{Z}_m}) = \Big(\big(f_1(\omega^j\gamma^0),\ldots,f_k(\omega^j\gamma^0)\big),\ldots,\big(f_1(\omega^j\gamma^{m-1}),\ldots,f_k(\omega^j\gamma^{m-1})\big)\Big).$$

The relation is then captured as:

$$\mathsf{R}_{k\text{-tp}} = \left\{ \begin{pmatrix} \perp, \\ [\![f_i]\!],[\![g_i]\!]_{i\in[k]} \\ [f_i,g_i]_{i\in[k]} \end{pmatrix} : \begin{array}{l} \{\!\{((f_i(\omega^j\gamma^l)_{i\in[k]})_{l\in[m]})\}\!\}_{j\in[n]} \\ \quad = \{\!\{((g_i(\omega^j\gamma^l)_{i\in[k]})_{l\in[m]})\}\!\}_{j\in[n]} \end{array} \right\}.$$

We provide the following theorem for the completeness and knowledge soundness of our k-tuple permutation argument.

Theorem 5. *k-TuPerm for $\mathsf{R}_{k\text{-tp}}$ (Figure 9) satisfies perfect completeness and for any adversary $\mathcal{A}$ against knowledge soundness, we provide an extractor X such that $\mathsf{Adv}^{\text{sound}}_{k\text{-TuPerm},\mathsf{R}_{k\text{-tp}},\mathsf{X},6,\mathcal{A}}(\lambda) \le \frac{(16+k)|\mathbb{H}|+2|\mathbb{V}|+1}{|\mathbb{F}\backslash\mathbb{H}|}$.*

Proof. We argue completeness and knowledge soundness separately.

Completeness. By linear combinations, if

$$\{\!\{((f_i(\omega^j\gamma^l)_{i\in[k]})_{l\in[m]})\}\!\}_{j\in[n]} = \{\!\{((g_i(\omega^j\gamma^l)_{i\in[k]})_{l\in[m]})\}\!\}_{j\in[n]}\,,$$

then

$$\{\!\{(f(\omega^j\gamma^l)_{l\in[m]})\}\!\}_{j\in[n]} = \{\!\{(g(\omega^j\gamma^l)_{l\in[m]})\}\!\}_{j\in[n]}\,.$$

Then the completeness holds because of the completeness of TuPerm.

Knowledge soundness. We bound the advantage through a series of game hops. First define $\mathrm{G}_0 = \textsc{Sound}^{\mathcal{A}}_{k\text{-TuPerm},\mathsf{R}_{k\text{-tp}},\mathsf{X},6}(\lambda)$. The inequality above follows from the following claims that we will justify:

(1) $|\Pr[\mathrm{G}_0 = 1] - \Pr[\mathrm{G}_1 = 1]| \le \frac{16|\mathbb{H}|+2|\mathbb{V}|+1}{|\mathbb{F}\backslash\mathbb{H}|}$

(2) $|\Pr[\mathrm{G}_1 = 1] - \Pr[\mathrm{G}_2 = 1]| \le \frac{k|\mathbb{H}|}{|\mathbb{F}\backslash\mathbb{H}|}$

(3) $\Pr[\mathrm{G}_2 = 1] = 0$

Claim 1 argues for the tuple permutation of f and g. Claim 2 argues for the tuple permutation of f_i's and g_i's given that. Lastly, Claim 3 argues that the constructed extractor always succeeds for an accepting verifier.

Claim 1: In this step, we argue that f and g are tuple permutations by the soundness of TuPerm. The probability of the bad flag being set is bounded by the soundness advantage of the tuple permutation protocol,

- TuPerm for f, g with advantage $\frac{16|\mathbb{H}|+2|\mathbb{V}|+1}{|\mathbb{F}\backslash\mathbb{H}|}$

G_1 employs the extractor $\mathsf{X}_{\mathsf{TuPerm}}$ and aborts if the extractor fails.

Claim 2: In the second step, we argue that each f_i and g_i is a tuple permutation of the other by considering the random linear combination. Each evaluation of f (respectively g) can be considered as a Reed-Solomon encoding of the k evaluations of $[f_i]_{i\in[k]}$. For each evaluation $j \in [n]$ and $l \in [m]$, if $\sum_{i\in[k]} \alpha^i f_i(\omega^j\gamma^l) = \sum_{i\in[k]} \alpha^i g_i(\omega^j\gamma^l)$, then we can bound the probability that $(f_0(\omega^j\gamma^l),\ldots,f_{k-1}(\omega^j\gamma^l)) \ne (g_0(\omega^{\pi(j)}\gamma^l),\ldots,g_{k-1}(\omega^{\pi(j)}\gamma^l))$ using an application of Schwartz-Zippel with random verifier challenge to $\frac{k}{|\mathbb{F}\backslash\mathbb{H}|}$. We set a bad flag in each of a series of hybrids for each of $|\mathbb{H}| = mn$ evaluations resulting in a union bound of $\frac{k\mathbb{H}}{|\mathbb{F}\backslash\mathbb{H}|}$.

Claim 3: For $p \in \{f, g\}$, our extractor can query p_i's at $(\max_i(\text{degree}(p_i)) + 1)$ points to extract the polynomials from oracles. Since this uniquely determines the polynomial, it has to be the witness polynomial for the oracle if the verifier accepts for the index and oracle pair. □

Tuple permutation with different groups. Our goal eventually is to construct a tuple lookup protocol where the lookup polynomial f and the table polynomial t are defined over different groups. Towards that goal, we adjust the k-tuple permutation protocol to work over different groups; our protocol which we call k-$\mathsf{XGTuPerm}$ is given in Figure 10.

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 tuples of f_i and $u_{1,i}$ over the cosets of $\mathbb{V}$ in group $\mathbb{H}_0$, and the tuples of t_i and $u_{2,i}$ over the cosets of $\mathbb{V}$ in a different group $\mathbb{H}_1$. The relation $\mathsf{R}_{k\text{-xgtp}}$ checks that the list of tuples across f_i and t_i are a permutation of the tuples 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:

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

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

Corollary 6. *k-XGTuPerm for $\mathsf{R}_{k\text{-xgtp}}$ (Figure 10) satisfies perfect completeness and for any adversary $\mathcal{A}$ against knowledge soundness, we provide an extractor X such that*

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

.

4.2 Tuple Lookup from Tuple Permutation

Given two vectors $t \in \mathbb{F}^{d_1}, f \in \mathbb{F}^{d_0}$, Plookup [GW20, PFM+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 tuples defined over coset evaluations. The relation for TuPlookup as follows:

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

We now give a high-level overview of our approach, detailed in Figure 11. For notational simplicity, for a polynomial p, let p_i denote the tuple 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 tuples 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 tuple 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 tuple lookup argument.

Theorem 7. *TuPlookup for R_{tl} (Figure 11) satisfies perfect completeness and for any adversary $\mathcal{A}$ against knowledge soundness, we provide an extractor X such that $\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{TuPlookup},\mathsf{R}_{\mathsf{tl}},\mathsf{X},7,\mathcal{A}}(\lambda) \leq \frac{20|\mathbb{H}_0|+19|\mathbb{H}_1|+3\max(|\mathbb{H}_0|,|\mathbb{H}_1|)+5|\mathbb{V}|+4}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}$.*

Proof. We argue completeness and knowledge soundness separately.

Completeness. 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 tuple 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 tuple permutation checks the following:

$$\left\{(f(\psi^i\mathbb{V}), f(\psi^i\mathbb{V}))\right\}_{i\in[d_0]} \cup \left\{(t(\mu^i\mathbb{V}), t(\mu^{i+1}\mathbb{V})\right\}_{i\in[d_1]} = \{(s_i, s_{i+1})\}_{i\in[d_0+d_1-1]} \cup (s_{d_0+d_1-1}, s_0)$$

First, consider $x = f(\psi^i\mathbb{V})$ wlog for some i. If x has multiplicity ℓ 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 ℓ pairs of $(f(\psi^i\mathbb{V}), f(\psi^i\mathbb{V}))$ contributed to the multiset, the same ℓ 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 tuples 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 tuple permutation argument will succeed by the completeness of 2-XGTuPerm.

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

$$k\text{-}\mathsf{XGTuPerm.P}\left(\begin{array}{l} \perp, \\ [\![f_i]\!], [\![t_i]\!], [\![u_{1,i}]\!], [\![u_{2,i}]\!]_{i\in[k]}, \\ [f_i, t_i, u_{1,i}, u_{2,i}]_{i\in[k]} \end{array} \right) \leftrightarrow k\text{-}\mathsf{XGTuPerm.V}\Big(\perp, [\![f_i]\!], [\![t_i]\!], [\![u_{1,i}]\!], [\![u_{2,i}]\!]_{i\in[k]}\Big)$$

(1) P and V derive polynomial f, t, u_1, u_2 as the linear combinations of f_i's, t_i's, $u_{1,i}$'s and $u_{2,i}$'s:

- V sends random challenge $\alpha \leftarrow\!\!\$\ \mathbb{F} \setminus (\mathbb{H}_0 \cup \mathbb{H}_1)$.
- Through additive homomorphism, P and V derive $f = \sum_{i\in[k]} \alpha^i f_i$, $t = \sum_{i\in[k]} \alpha^i t_i$, $u_1 = \sum_{i\in[k]} \alpha^i u_{1,i}$ and $u_2 = \sum_{i\in[k]} \alpha^i u_{2,i}$.

(2) P computes and sends the position-indexing polynoimals $I_1(X)$ over $\mathbb{H}_0$, $I_2(X)$ over $\mathbb{H}_1$ and proves their well-formedness:

- V sends random challenge $\beta \leftarrow\!\!\$\ \mathbb{F} \setminus (\mathbb{H}_0 \cup \mathbb{H}_1)$.
- P computes and sends I_1 defined over $\mathbb{H}_0$, I_2 over $\mathbb{H}_1$ setting the evaluation of the j^{th} element of each coset to be j-th randomness β^j:
$$[I_1(\psi^i\gamma^j) = I_1(\mu^i\gamma^j) = \beta^j]_{i\in[n], j\in[m]}.$$
- P and V engage in $\mathsf{ZeroTest}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $L_{1,\mathbb{V}}(X)(I_1(X) - 1) = 0$ over $\mathbb{V}$.
- P and V engage in $\mathsf{ZeroTest}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $L_{1,\mathbb{V}}(X)(I_2(X) - 1) = 0$ over $\mathbb{V}$.
- P and V engage in $\mathsf{ZeroTest}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $(I_1(\gamma X) - \beta \cdot I_1(X))(X - \gamma^{m-1}) = (I_2(\gamma X) - \beta \cdot I_2(X))(X - \gamma^{m-1}) = 0$ over $\mathbb{V}$.
- P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $(I_1(X) - I_1(\psi X))Z_{\psi^{d_0-1}\mathbb{V}}(X) = 0$ over $\mathbb{H}_0$.
- P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $(I_2(X) - I_2(\mu X))Z_{\mu^{d_1-1}\mathbb{V}}(X) = 0$ over $\mathbb{H}_1$.

(3) P computes and sends the summation polynomials $S_f(X), S_t(X), S_{u_1}(X), S_{u_2}(X)$ for $f(X), t(X), u_1(X), u_2(X)$, respectively, and proves their well-formedness:

- For $p_1 \in \{f, u_1\}$, P computes and sends S_{p_1} defined over $\mathbb{H}_0$ setting the evalution to be a constant in each coset $i \in [n]$ representing the hash of $[p_1(\psi^i\gamma^j)]_{j\in[m]}$. S_{p_2} is defined in a similar way for $p_2 \in \{t, u_2\}$ but defined over $\mathbb{H}_1$.
$$\Big[S_f(\psi^l\gamma^j) = \textstyle\sum_{k\in[m]} \beta^k f(\psi^l\gamma^k)\Big]_{l\in[d_0], j\in[m]}, \qquad \Big[S_{u_1}(\psi^l\gamma^j) = \textstyle\sum_{k\in[m]} \beta^k u_1(\psi^l\gamma^k)\Big]_{l\in[d_0], j\in[m]},$$
$$\Big[S_t(\mu^l\gamma^j) = \textstyle\sum_{k\in[m]} \beta^k t(\mu^l\gamma^k)\Big]_{l\in[d_1], j\in[m]}, \qquad \Big[S_{u_2}(\mu^l\gamma^0) = \textstyle\sum_{k\in[m]} \beta^k u_2(\mu^l\gamma^k)\Big]_{l\in[d_1], j\in[m]}.$$
- P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $S_{p_1}(\gamma X) = S_{p_1}(X)$ over $\mathbb{H}_0$, and $\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $S_{p_2}(\gamma X) = S_{p_2}(X)$ over $\mathbb{H}_1$ showing every coset of $\mathbb{V}$ evaluates to a constant.

(4) P computes and sends the normalized induction polynomials $B_f(X), B_t(X), B_{u_1}(X), B_{u_2}(X)$ for $f(X), t(X), u_1(X), u_2(X)$, respectively, and proves their well-formedness:

- For $p_1 \in \{f, u_1\}, p_2 \in \{t, u_2\}$, P computes and sends B_{p_1} over $\mathbb{H}_0$, B_{p_2} over $\mathbb{H}_1$ that accumulates the hash summation:
$$[B_{p_1}(\psi^l\gamma^0) = 0]_{l\in[d_0]}, \qquad \left[B_{p_1}(\omega^i\gamma^j) = \textstyle\sum_{k\in[j]} \beta^k p_1(\omega^i\gamma^k) - j \cdot \frac{S_{p_1}(\omega^i\gamma^j)}{m}\right]_{i\in[n], j\in[m-1]},$$
$$[B_{p_2}(\psi^l\gamma^0) = 0]_{l\in[d_0]}, \qquad \left[B_{p_2}(\omega^i\gamma^j) = \textstyle\sum_{k\in[j]} \beta^k p_2(\omega^i\gamma^k) - j \cdot \frac{S_{p_2}(\omega^i\gamma^j)}{m}\right]_{i\in[n], j\in[m-1]}.$$
- P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $B_{p_1}(\gamma X) = (B_{p_1}(X) + I_1(\gamma X) \cdot p_1(\gamma X)) - \frac{S_{p_1}(X)}{m}$ over $\mathbb{H}_0$.
- P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1)$ 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 $\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1)$, $\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1)$ 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 $\mathsf{XGProductCheck}$ to prove $\prod_{x\in\mathbb{H}_0} q_1(x) = \prod_{x\in\mathbb{H}_1} q_2(x)$.

Figure 10: k-XGTuPerm: Tuple permutation across different groups.

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

$\mathsf{TuPlookup.P}(\perp, ([\![f]\!], [\![t]\!]), (f,t)) \leftrightarrow \mathsf{TuPlookup.V}(\perp, ([\![f]\!], [\![t]\!]))$

(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_1', u_2' that encode shifted versions of u_1, u_2 wrapping the last coset from u_2 to u_1' and wrapping the last coset of u_1 to u_2', then proving wellformedness.

(a) P interpolates u_1', u_2' over $\mathbb{H}_0, \mathbb{H}_1$ respectively with the following defined evaluations:

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

(b) P and V engage in $\mathsf{ZeroTest}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $u_1'(X) = u_2(\mu^{d_1-1}X)$ over $\mathbb{V}$.

(c) P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_0, \mathbb{H}_0 \cup \mathbb{H}_1)$ 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 $\mathsf{ZeroTest}(\mathbb{V}, \mathbb{H}_0 \cup \mathbb{H}_1)$ to prove $u_2'(X) = u_1(\psi^{d_0-1}X)$ over $\mathbb{V}$.

(e) P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_1, \mathbb{H}_0 \cup \mathbb{H}_1)$ 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

$$\{\!\{(f(\psi^j \mathbb{V}), f(\psi^j \mathbb{V}))\}\!\}_{j\in[d_0]} \cup \{\!\{(t(\mu^j \mathbb{V}), t(\mu^{j+1}\mathbb{V}))\}\!\}_{j\in[d_1]} = \{\!\{(u_1'(\psi^j \mathbb{V}), u_1(\psi^j \mathbb{V}))\}\!\}_{j\in[d_0]} \cup \{\!\{(u_2'(\mu^j \mathbb{V}), u_2(\mu^j \mathbb{V}))\}\!\}_{j\in[d_1]}.$$

Figure 11: TuPlookup: Tuple lookup argument

Knowledge soundness. We bound the advantage through a series of game hops. First define $\mathrm{G}_0 = \mathrm{SOUND}^{\mathcal{A}}_{\mathsf{TuPlookup},\mathsf{R}_{\mathsf{tl}},\mathsf{X},7}(\lambda)$. The inequality above follows from the following claims that we will justify:

(1) $|\Pr[\mathrm{G}_0 = 1] - \Pr[\mathrm{G}_1 = 1]| \le \frac{|\mathbb{H}_0|+|\mathbb{H}_1|+2\max(|\mathbb{H}_0|,|\mathbb{H}_1|)+|\mathbb{V}|}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}$

(2) $|\Pr[\mathrm{G}_1 = 1] - \Pr[\mathrm{G}_2 = 1]| \le \frac{19|\mathbb{H}_0|+18|\mathbb{H}_1|+\max(|\mathbb{H}_0|,|\mathbb{H}_1|)+4|\mathbb{V}|+4}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}$

(3) $\Pr[\mathrm{G}_2 = 1] = 0$

Claim 1 argues for the wellformedness of u_1' and u_2'. Claim 2 argues that the tuple permutation is correct. Lastly, Claim 3 argues that this implies the tuple 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):

- $u_1'(X) = u_2(\mu^{d_1-1}X)$ over $\mathbb{V}$ with advantage $\frac{\max(|\mathbb{H}_0|,|\mathbb{H}_1|)}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}$: Checks the first coset of u_1' is set to last coset of u_2.
- $(u_1'(X) - u_1(\psi^{-1}X))Z_{\mathbb{V}}(X) = 0$ over $\mathbb{H}_0$ with advantage $\frac{|\mathbb{H}_0|+\mathbb{V}}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}$: 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 $\frac{\max(|\mathbb{H}_0|,|\mathbb{H}_1|)}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}$: 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 $\frac{|\mathbb{H}_1|+\mathbb{V}}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}$: 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 $\mathsf{X}_{\mathsf{ZeroTest}}$ 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 tuple relation holds:

- 2-XGTuPerm for $f, t, u_1', u_1, u_2', u_2$ with advantage $\frac{19|\mathbb{H}_0|+18|\mathbb{H}_1|+\max(|\mathbb{H}_0|,|\mathbb{H}_1|)+4|\mathbb{V}|+4}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}$: 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 $\mathsf{X}_{\text{2-XGTuPerm}}$ 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 tuple permutation holds, the tuple lookup relation is satisfied. Let's first define s such that $\left[u_1(\psi^i \mathbb{V}) = s[i]\right]_{i\in[d_0]}, \left[u_2(\mu^i \mathbb{V}) = s[d_0+i]\right]_{i\in[d_1]}$. Notice that u_1' is shifted u_1 and u_2' is shifted u_2, and

$$\mathsf{R}_{w\text{-tl}} = \left\{ \bot, ([\![f_i]\!]_{i\in[w]}, [\![t_i]\!]_{i\in[w]}), ([f_i, t_i]_{i\in[w]}) : \left\{(f_i(\psi^j \mathbb{V}))_{i\in[w]}\right\}_{j\in[d_0]} \subseteq \left\{(t_i(\mu^j \mathbb{V}))_{i\in[w]}\right\}_{j\in[d_1-1]} \right\}$$

w-$\mathsf{TuPlookup.P}(\bot, ([\![f_i]\!], [\![t_i]\!]]_{i\in[w]}), ([f_i, t_i]_{i\in[w]})) \leftrightarrow w$-$\mathsf{TuPlookup.V}(\bot, ([\![f_i]\!], [\![t_i]\!]]_{i\in[w]}))$

(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 $\left\{(f(\psi^j \mathbb{V}))\right\}_{j\in[d_0]} \subseteq \left\{(t(\mu^j \mathbb{V}))\right\}_{j\in[d_1-1]}$

Figure 12: w-TuPlookup: Tuple lookup argument enforcing same row lookups for w tables

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:

$$\begin{aligned}
&\{\!\{(u_1'(\psi^j\mathbb{V}), u_1(\psi^j\mathbb{V}))\}\!\}_{j\in[d_0]} \cup \{\!\{(u_2'(\mu^j\mathbb{V}), u_2(\mu^j\mathbb{V}))\}\!\}_{j\in[d_1]} \\
=&\{\!\{(u_2(\mu^{d_1-1}\mathbb{V}), u_1(\mathbb{V})), (u_1(\mathbb{V}), u_1(\phi\mathbb{V})), \ldots, (u_1(\phi^{d_0-2}\mathbb{V}), u_1(\phi^{d_0-1}\mathbb{V}))\}\!\} \\
&\cup \{\!\{(u_1(\phi^{d_0-1}\mathbb{V}), u_2(\mathbb{V})), (u_2(\mathbb{V}), u_2(\mu\mathbb{V})), \ldots, (u_2(\mu^{d_1-2}\mathbb{V}), u_2(\mu^{d_1-1}\mathbb{V}))\}\!\} \\
=&\{\!\{(s_i, s_{i+1})\}\!\}_{i\in[d_0+d_1-1]} \cup (s_{d_0+d_1-1}, s_0)
\end{aligned}$$

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 $\left\{(f(\psi^i\mathbb{V}))\right\}_{i\in[d_0]} \subseteq \left\{(t(\mu^i\mathbb{V}))\right\}_{i\in[d_1]}$. 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 $\mathsf{X}_{\text{2-XGTuPerm}}$ to extract and output f, t. By Claim 2, if the verifier succeeds then $\mathsf{X}_{\text{2-XGTuPerm}}$ succeeds and so X always succeeds.

□

Tuple 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 8. *w-TuPlookup for $\mathsf{R}_{w\text{-tl}}$ (Figure 12) satisfies perfect completeness and for any adversary $\mathcal{A}$ against knowledge soundness, we provide an extractor* X *such that*

$$\mathsf{Adv}^{\text{sound}}_{w\text{-TuPlookup},\mathsf{R}_{w\text{-tl}},\mathsf{X},8,\mathcal{A}}(\lambda) \leq \frac{(20+w)|\mathbb{H}_0|+19|\mathbb{H}_1|+3\max(|\mathbb{H}_0|,|\mathbb{H}_1|)+5|\mathbb{V}|+4}{|\mathbb{F}\setminus(\mathbb{H}_0\cup\mathbb{H}_1)|}.$$

5 Mux-Marlin: Proofs for Unrolled Machine Execution from Tuple Lookups

We model a machine execution of a machine with ℓ instructions using ℓ 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:

$$x = (inst_{in}, mem_{in}, inst_{out}, mem_{out}),$$

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:

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

In the following sections we build unrolled machine execution proof systems for instructions encoded as rank-1 constraint systems ($\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{r1cs}}]$) derived from the Marlin proof system [CHM+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 Additional Marlin Preliminaries

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

Rank-1 constraint satisfiability (R1CS). A common arithmetization used in proofs for relations in NP is rank-1 constraint satisfiability (R1CS). An R1CS relation is indexed by the tuple $(\mathbb{F}, A, B, C, d_x)$ where $A, B, C \in \mathbb{F}^{d \times d}$. The statement and witness $\left[\begin{smallmatrix} w \\ x \end{smallmatrix}\right] \in \mathbb{F}^d$ together form a vector (with the length of the statement $d_x \leq d$ specified) where the following algebraic relation is satisfied:

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

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 13 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 $\left[\begin{smallmatrix} w \\ x \end{smallmatrix}\right]$ as evaluations over a multiplicative subgroup $\mathbb{H} \leqslant \mathbb{F}$ where $|\mathbb{H}| = d$ and, more precisely, further encodes x as evaluations over a subgroup $\mathbb{H}_x \leqslant \mathbb{H}$ 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:

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

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 $x = y$.

5.2 Construction

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

$$\mathsf{R}_{\mathsf{r1cs}} = \left\{ \left(\begin{array}{l} (\mathbb{F}, \mathbb{H}, \mathbb{H}_{\mathsf{x}}, A, B, C), \\ (\llbracket x \rrbracket, \mathbb{K}), \\ (\mathsf{w}, x) \end{array} \right) : \begin{array}{l} A \left[\begin{smallmatrix} \mathsf{w} \\ \mathsf{x} \end{smallmatrix}\right] \circ B \left[\begin{smallmatrix} \mathsf{w} \\ \mathsf{x} \end{smallmatrix}\right] = C \left[\begin{smallmatrix} \mathsf{w} \\ \mathsf{x} \end{smallmatrix}\right] \\ x(\mathbb{H}_{\mathsf{x}}) = \mathsf{x} \end{array} \right\}$$

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

$\mathsf{Marlin.Index}(\perp, (\mathbb{F}, \mathbb{H}, \mathbb{H}_{\mathsf{x}}, A, B, C))$

(1) For each matrix $M \in [A, B, C]$, compute the polynomials row_M, col_M, val_M using evaluations defined over $\mathbb{H}$. Assume a canonical mapping ϕ between the non-zero elements of M and the elements of $\mathbb{H} = \langle \omega \rangle$.
   - Set $\left[row_M(\omega^i) = \omega^j\right]_{i=0}^{d-1}$ where j is the row of the element $\phi(\omega^i)$ in M.
   - Set $\left[col_M(\omega^i) = \omega^j\right]_{i=0}^{d-1}$ where j is the column of the element $\phi(\omega^i)$ in M.
   - Set $\left[val_M(\omega^i) = \frac{\phi(\omega^i)}{u_{\mathbb{H}}(row_M(\omega^i), row_M(\omega^i)) \cdot u_{\mathbb{H}}(col_M(\omega^i), col_M(\omega^i))}\right]_{i=0}^{d-1}$.

(2) Return $\left(pp \leftarrow [(row_M, col_M, val_M)]_{M \in [A,B,C]}, vp \leftarrow [(\llbracket row_M \rrbracket, \llbracket col_M \rrbracket, \llbracket val_M \rrbracket)]_{M \in [A,B,C]} \right)$

$\mathsf{Marlin.P}([(row_M, col_M, val_M)]_{M \in [A,B,C]}, (\llbracket x \rrbracket, \mathbb{K}), (\mathsf{w}, x)) \leftrightarrow \mathsf{Marlin.V}([(\llbracket row_M \rrbracket, \llbracket col_M \rrbracket, \llbracket val_M \rrbracket)]_{M \in [A,B,C]}, (\llbracket x \rrbracket, \mathbb{K}))$

(1) P computes and sends polynomial z that evaluates to the vector $\left[\begin{smallmatrix} \mathsf{w} \\ \mathsf{x} \end{smallmatrix}\right]$ where x is the vector that x evaluates to over $\mathbb{H}_{\mathsf{x}}$.

(2) For each matrix $M \in [A, B, C]$, P computes and sends the polynomial z_M that evaluates to the vector $M \cdot \left[\begin{smallmatrix} \mathsf{w} \\ \mathsf{x} \end{smallmatrix}\right]$ over $\mathbb{H}$.

(3) V sends random challenge $r_1 \in \mathbb{F} \setminus (\mathbb{H} \cup \mathbb{K})$.

(4) For each matrix $M \in [A, B, C]$, P proves well-formedness of polynomial z_M:

   (a) P computes and sends polynomial q_M as follows:
   $$q_M(X) = \left(z(X) \cdot \sum_{\omega \in \mathbb{H}} u_{\mathbb{H}}(r_1, row_M(\omega)) \cdot u_{\mathbb{H}}(X, col_M(\omega)) \cdot val_M(\omega) \right) - z_M(X) \cdot |\mathbb{H}|^{-1}$$

   (b) P and V engage in $\mathsf{SumCheck}(\mathbb{H}, \mathbb{H} \cup \mathbb{K})$ protocol to prove q_M sums to 0 over $\mathbb{H}$.

   (c) V sends random challenge $r_M \in \mathbb{F} \setminus (\mathbb{H} \cup \mathbb{K})$.

   (d) P proves well-formedness of q_M:
      - P computes and sends polynomial f_M defined by the following evaluations over $\mathbb{H}$:
      $$\left[f_M(\omega^i) = \frac{Z_{\mathbb{H}}(r_1) \cdot Z_{\mathbb{H}}(r_M) \cdot val_M(\omega^i)}{(r_1 - row_M(\omega^i))(r_M - col_M(\omega^i))} \right]_{i=0}^{d-1}$$
      - V queries q_M, z_M, z on r_M and computes $\sigma = \frac{q_M(r_M) + z_M(r_M) \cdot |\mathbb{H}|^{-1}}{z(r_M)}$.
      - P and V engage in $\mathsf{SumCheck}(\mathbb{H}, \mathbb{H} \cup \mathbb{K})$ to prove f_M sums to σ over $\mathbb{H}$.
      - P and V engage in $\mathsf{ZeroTest}(\mathbb{H}, \mathbb{H} \cup \mathbb{K})$ protocol to prove $(r_1 - row_M)(r_M - col_M)f - Z_{\mathbb{H}}(r_1) \cdot Z_{\mathbb{H}}(r_M) \cdot val_M$ evaluates to 0 over $\mathbb{H}$.

(5) P and V engage in $\mathsf{ZeroTest}(\mathbb{H}, \mathbb{H} \cup \mathbb{K})$ protocol to prove $z_A \cdot z_B - z_C$ evaluates to 0 over $\mathbb{H}$.

(6) To prove correctness of z, P and V engage in $\mathsf{ZeroTest}(\mathbb{H}_{\mathsf{x}}, \mathbb{H} \cup \mathbb{K})$ protocol to prove $z - x$ evaluates to 0 over $\mathbb{H}_{\mathsf{x}}$.

Figure 13: Marlin: Marlin polyIOP [CHM+20].

- Define $\mathbb{V} = \langle \gamma \rangle$ as the multiplicative subgroup of size d where the R1CS instruction index has matrices of dimension d.
- Define $\mathbb{V}_{\mathsf{x}} \leqslant \mathbb{V}$ as the multiplicative subgroup of size d_{x} of the R1CS instruction index where $d/d_{\mathsf{x}} = a$.
- Define $\mathbb{V}_{in} \leqslant \mathbb{V}_x$ as the multiplicative subgroup of size $d_{\mathsf{x}}/2$ generated by $\mu^{\frac{2nd}{d_{\mathsf{x}}}}$ and $\mathbb{V}_{out} = \mu^{\frac{2nd}{d_{\mathsf{x}}}} \mathbb{V}_{in}$ as the other coset of $\mathbb{V}_{in}$ in $\mathbb{V}_x$ encoding the two parts of the machine execution statement.
- Define $\mathbb{H} = \langle \omega \rangle$ as the multiplicative subgroup of size $d\ell$ where ℓ is the number of instructions. Denote the ℓ cosets of $\mathbb{V}$ in $\mathbb{H}$ as $[\omega^i \mathbb{V}]_{i=0}^{\ell-1}$.
- Define $\mathbb{G} = \langle \mu \rangle$ as the multiplicative subgroup of size dn where n is the number of unrolled execution steps. Denote the n cosets of $\mathbb{V}$ in $\mathbb{G}$ as $[\mu^i \mathbb{V}]_{i=0}^{n-1}$.

We discuss some specific aspects of our construction beyond the details in Section 2.3. Figure 14 and Figure 15 provide the details for our adaptation of the Marlin proof system for unrolled machine execution.

$$\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{r1cs}}] = \left\{ \begin{pmatrix} \left(\mathbb{F},\mathbb{G},\mathbb{H},\mathbb{V},\mathbb{V}_x,\mathbb{V}_{in},[A_i,B_i,C_i]_{i=0}^{\ell-1}\right), \\ ([\![x_0]\!],[\![x_n]\!]), \\ \left([inst_j,mem_j,w_j]_{j=0}^{n},x_0,x_n\right) \end{pmatrix} : \begin{array}{l} x_0(\mathbb{V}_{in}) = [inst_0, mem_0] \\ x_n(\mathbb{V}_{out}) = [inst_n, mem_n] \\ \bigwedge_{j=0}^{n-1} \begin{pmatrix} \left(\mathbb{F}, A_{inst_j}, B_{inst_j}, C_{inst_j}, |\mathbb{V}_x|\right), \\ [inst_j, mem_j, inst_{j+1}, mem_{j+1}], \\ w_j \end{pmatrix} \in \mathsf{R}_{\mathsf{r1cs}} \end{array} \right\}$$

<u>$\mathsf{Mux\text{-}Marlin.Setup}(\lambda)$</u>: Return $\perp$

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

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

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

(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 $\mathbb{V}$.

$$\left[\left[tval_M(\omega^i\gamma^k) = \phi_i(\gamma^k)\right]_{k=0}^{d}\right]_{i=0}^{\ell-1}$$

(4) Return $\left(pp \leftarrow [(trow_M, tcol_M, tval_M)]_{M\in[A,B,C]}, vp \leftarrow [([\![trow_M]\!],[\![tcol_M]\!],[\![tval_M]\!])]_{M\in[A,B,C]}\right)$

Figure 14: Setup for Mux-Marlin: Adapted Marlin polyIOP for unrolled machine execution.

<u>*Recovering global structure of val_M.*</u> In the Marlin proof system, 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:

$$\begin{aligned} \forall \mu \in \mathbb{G}, \quad val_M(\mu) &= \frac{val'_M(\mu)}{u_{\mathbb{G}}(row_M(\mu), row_M(\mu)) \cdot u_{\mathbb{G}}(col_M(\mu), col_M(\mu))} \\ &= \frac{val'_M(\mu)}{|\mathbb{G}|^2 \cdot row_M(\mu)^{|\mathbb{G}|-1} \cdot col_M(\mu)^{|\mathbb{G}|-1}} \\ &= \frac{val'_M(\mu) \cdot row_M(\mu) \cdot col_M(\mu)}{|\mathbb{G}|^2} \end{aligned} \tag{1}$$

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 $\mathbb{G}$.

<u>*Outsourcing the computation of vanishing polynomial $Z_{\mathbb{G}_{in}\setminus\mathbb{V}_{in}}$.*</u> In our Mux-Marlin construction, it is crucial to perform a ZeroTest over the set $\mathbb{G}_{in} \setminus \mathbb{V}_{in}$ to check that current instruction defined at $\mathbb{V}_{in}$ is connected (copied to) its previous instruction defined at $\mathbb{V}_{out}$: $x(\mu^j\mathbb{V}_{in}) = x(\mu^{j-1}\mathbb{V}_{out})$ for $j \in [1,n)$. Notice that $Z_{\mathbb{G}_{in}\setminus\mathbb{V}_{in}}(X) = \prod_{j\in\mathbb{Z}_n} Z_{\mu^j\mathbb{V}_{in}}(X)$ cannot be 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 $[Z_{\mu^j\mathbb{V}_{in}}(X) = X^{\frac{d_x j}{2}} - \mu^{\frac{d_x j}{2}}]_{j\in\mathbb{Z}_n}$ since $\mathbb{V}_{in}$ is of order $\frac{d_x}{2}$. To prove wellformedness, the

$$\mathsf{Mux\text{-}Marlin.P}\left(\begin{array}{l}[(trow_M, tcol_M, tval_M)]_{M\in[A,B,C]},\\ ([\![x_0]\!],[\![x_n]\!]),\\ \left([inst_j, mem_j, w_j]_{j=0}^{n}, x_0, x_n\right)\end{array}\right) \leftrightarrow \mathsf{Mux\text{-}Marlin.V}\left(\begin{array}{l}[([\![trow_M]\!],[\![tcol_M]\!],[\![tval_M]\!])]_{M\in[A,B,C]},\\ ([\![x_0]\!],[\![x_n]\!])\end{array}\right)$$

(1) For $M \in [A,B,C]$, P computes polynomials row'_M, col'_M, val'_M and proves their well-formedness.

(a) Set polynomials row'_M, col'_M, val'_M using evaluations over $\mathbb{G}$ setting the evaluation of the j^{th} coset of $\mathbb{V}$ in $\mathbb{G}$ to the corresponding preprocessed evaluations for instruction $inst_j$:

$$\left[row'_M(\mu^j\mathbb{V}) = trow_M(\omega^{\mathsf{inst}_j}\mathbb{V})\right]_{j=0}^{n-1} \quad \left[col'_M(\mu^j\mathbb{V}) = tcol_M(\omega^{\mathsf{inst}_j}\mathbb{V})\right]_{j=0}^{n-1} \quad \left[val'_M(\mu^j\mathbb{V}) = tval_M(\omega^{\mathsf{inst}_j}\mathbb{V})\right]_{j=0}^{n-1}$$

(b) P sends polynomials row'_M, col'_M, val'_M to V.

(c) P and V engage in 9-TuPlookup to prove well-formedness of row'_M, col'_M, val'_M with respect to $trow_M, tcol_M, tval_M$.

(2) For $M \in [A,B,C]$, P computes and sends "shifted" polynomials row_M and col_M, and proves their well-formedness.

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

– P computes and sends s over $\mathbb{G}$ setting the evaluation of the j^{th} coset of $\mathbb{V}$ in $\mathbb{G}$ to be the shift μ^j: $\left[s(\mu^j\mathbb{V}) = \mu^j\right]_{j=0}^{n-1}$.

– P and V engage in ZeroTest$(\mathbb{V}, \mathbb{H}\cup\mathbb{G})$ to prove $s(X) = s(\gamma X)$ over $\mathbb{V}$.

– P and V engage in ZeroTest$(\mathbb{G}, \mathbb{H}\cup\mathbb{G})$ to prove $(\mu\cdot s(X) - s(\mu X))(Z_{\mu^{n-1}\mathbb{V}}(X)) = 0$ over $\mathbb{G}$.

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

(b) For $M \in [A,B,C]$, P computes and sends "shifted" polynomials row_M and col_M, and proves their well-formedness.

– P computes and sends row_M, col_M set to shifted evaluations of row'_M, col'_M over $\mathbb{G}$:

$$\left[row_M(\mu^j\mathbb{V}) = \mu^j\cdot row'_M(\mu^j\mathbb{V})\right]_{j=0}^{n-1} \qquad \left[col_M(\mu^j\mathbb{V}) = \mu^j\cdot col'_M(\mu^j\mathbb{V})\right]_{j=0}^{n-1}$$

– P and V engage in ZeroTest$(\mathbb{G}, \mathbb{H}\cup\mathbb{G})$ to prove $row_M(X) = row'_M(X)s(X)$ and $col_M(X) = col'_M(X)s(X)$ over $\mathbb{G}$.

(3) For $M \in [A,B,C]$, P computes and sends "normalized" polynomial val_M, and proves its well-formedness.

(a) P computes and sends val_M set as evaluations of val'_M normalized over $\mathbb{G}$:

$$\left[val_M(\mu) = \frac{val'_M(\mu)}{u_{\mathbb{G}}(row_M(\mu), row_M(\mu))\cdot u_{\mathbb{G}}(col_M(\mu), col_M(\mu))} = \frac{val'_M(\mu)\cdot row_M(\mu)\cdot col_M(\mu)}{|\mathbb{G}|^2}\right]_{\mu\in\mathbb{G}}$$

(b) P and V engage in ZeroTest$(\mathbb{G}, \mathbb{H}\cup\mathbb{G})$ to prove $|\mathbb{G}|^2\cdot val_M(X) = val'_M(X)\cdot row_M(X)\cdot col_M(X)$ over $\mathbb{G}$.

(4) 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 a single step of machine execution.

$$\left[x(\mu^j\mathbb{V}) = [inst_j, mem_j, inst_{j+1}, mem_{j+1}, w_j]\right]_{j=0}^{n-1} \text{ s.t. } \left[x(\mu^j\mathbb{V}_{in}) = [inst_j, mem_j]\right]_{j=0}^{n-1} \quad \left[x(\mu^j\mathbb{V}_{out}) = [inst_{j+1}, mem_{j+1}]\right]_{j=0}^{n-1}$$

(b) P and V prove initial input with 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 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 $x(\mu^j\mathbb{V}_{in}) = x(\mu^{j-1}\mathbb{V}_{out})$ for $j \in [1,n)$.

– Define $\mathbb{G}_{in} = \bigcup_{j=0}^{n-1}\mu^j\mathbb{V}_{in}$. P computes and sends $f = Z_{\mathbb{G}_{in}\setminus\mathbb{V}_{in}}$ to V.

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

– Define $\mathbb{G}_n = \langle\mu^d\rangle$ as the multiplicative subgroup of $\mathbb{G}$ of size n. P computes and sends polynomial g defined using the following evaluations over $\mathbb{G}_n$: $g(1) = 1$ and $[g(\mu^{dj}) = \prod_{k=1}^{j} Z_{\mu^k\mathbb{V}_{in}}(r)]_{j=1}^{n-1}$.

– P computes and sends polynomial h defined using the following evaluations over $\mathbb{G}_n$ which encodes the constant term of the polynomials $[Z_{\mu^j\mathbb{V}_{in}}]_j^n$: $h(1) = 1$ and $[h(\mu^{dj}) = \mu^{d_x j/2}]_{j=1}^{n-1}$.

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

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

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

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

(5) P and V engage in Marlin proving protocol with preprocessed index polynomials $[row_M, col_M, val_M]_{M\in[A,B,C]}$.

$$\mathsf{Marlin.P}\left(\begin{array}{l}[(row_M, col_M, val_M)]_{M\in[A,B,C]},\\ ([\![x]\!],\mathbb{G}),\\ (\perp, [inst_j, mem_j, inst_{j+1}, mem_{j+1}, w_j]_{j=0}^{n})\end{array}\right) \leftrightarrow \mathsf{Marlin.V}([([\![row_M]\!],[\![col_M]\!],[\![val_M]\!])]_{M\in[A,B,C]}, ([\![x]\!],\mathbb{G}))$$

Figure 15: Mux-Marlin: Adapted Marlin polyIOP for unrolled machine execution.

prover create new polynomials to accumulate the products of $Z_{\mu^j \mathbb{V}_{in}}(r)$ 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 $\mu^{\frac{d_X i}{2}}$ of $Z_{\mu^i \mathbb{V}_{in}}(r)$ at μ^{di}, and a polynomial g to encode the intermediate product up to j^{th} item in multiplication $\prod_{i=1}^{j} Z_{\mu^i \mathbb{V}_{in}}(r)$ at μ^{di}. Then we use standard 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 starting of the induction.
- $(\mu^{\frac{d_X}{2}} \cdot h(X) - h(\mu^d X))(X - \mu^{d(n-1)}) = 0$ over $\mathbb{G}_n$: The next element in $\mathbb{G}_n$ is equal to $\mu^{\frac{d_X}{2}}$ 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.
- $(X-1)(g(X) - g(X/\mu^d) \cdot (r^{\frac{d_X}{2}} - h(X)) = 0$ over $\mathbb{G}_n$: The j^{th} element in $\mathbb{G}_n$ is equal to $r^{\frac{d_X}{2}} - h(X) = r^{\frac{d_X}{2}} - \mu^{\frac{d_X j}{2}}$ times $(j-1)^{th}$ element excluding the last and the first one. Since the first element is set to 1, this ensures that j^{th} element is set to the accumulated product $\prod_{i=1}^{j} Z_{\mu^i \mathbb{V}_{in}}(r)$ up to j.

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

Security. We prove the completeness and knowledge soundness of Mux-Marlin in the following theorem . The zero-knowledge of Mux-Marlin is achieved through the generic compiler observing that Mux-Marlin is domain-restriction admissible.

Theorem 9. Mux-Marlin *for* $\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{r1cs}}]$ *(Figure 15) satisfies perfect completeness and for any adversary* $\mathcal{A}$ *against knowledge soundness, we provide an extractor* X *such that*

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathsf{Mux\text{-}Marlin},\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{r1cs}}],\mathsf{X},19,\mathcal{A}}(\lambda) \leq \frac{76|\mathbb{G}| + 19|\mathbb{H}| + 3\max(|\mathbb{G}|,|\mathbb{H}|) + 5|\mathbb{V}| + 4}{|\mathbb{F} \setminus (\mathbb{G} \cup \mathbb{H})|}.$$

Proof of Theorem 9.

Proof. We consider each of completeness and soundness separately.

Completeness. 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:

$$\forall i, j \in [n] \quad M(\mu^i \mathbb{V}, \mu^j \mathbb{V}) = \begin{cases} M_{\mathit{inst}_j} & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

Since x evaluated on each coset $x(\mu^j \mathbb{V}) = [\mathit{inst}_j, \mathit{mem}_j, \mathit{inst}_{j+1}, \mathit{mem}_{j+1}, w_j]$, we have that for a valid prover

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

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

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

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.

Knowledge soundness. We bound the advantage of adversary $\mathcal{A}$ by bounding the advantage of each of a series of game hops. We define $\mathrm{G}_0 = \mathrm{SOUND}^{\mathcal{A}}_{\mathsf{Mux\text{-}Marlin},\mathsf{R}_{\mathsf{MExe},n}[\mathsf{R}_{\mathsf{r1cs}}],\mathsf{X},19}(\lambda)$. The inequality above follows from the following claims that we will justify:

(1) $|\Pr[\mathrm{G}_0 = 1] - \Pr[\mathrm{G}_1 = 1]| \leq \frac{14|\mathbb{G}|}{\mathbb{F} \setminus (\mathbb{G} \cup \mathbb{H})}$

(2) $|\Pr[\mathrm{G}_1 = 1] - \Pr[\mathrm{G}_2 = 1]| \leq \frac{54|\mathbb{G}| + 19|\mathbb{H}| + 3\max(|\mathbb{G}|,|\mathbb{H}|) + 5|\mathbb{V}| + 4}{|\mathbb{F} \setminus (\mathbb{G} \cup \mathbb{H})|}$

(3) $|\Pr[\mathrm{G}_2 = 1] - \Pr[\mathrm{G}_3 = 1]| \leq \frac{8|\mathbb{G}| + 1}{|\mathbb{F} \setminus (\mathbb{G} \cup \mathbb{H})|}$

(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: 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 (4bc):

- $x(X) = x_0(X)$ over $\mathbb{V}_{in}$ with advantage $\frac{|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: Checks correct first statement.
- $x(\mu^{n-1}X) = x_n(X)$ over $\mathbb{V}_{out}$ with advantage $\frac{|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: Checks correct last statement.

In G_1, we invoke the zero test extractor $\mathsf{X}_{\mathsf{ZeroTest}}$ 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 (4d):

- $L_{1,\mathbb{G}_n}(X)(h(X)-1) = 0$ over $\mathbb{G}_n$ with advantage $\frac{2|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: 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 $\frac{2|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: 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 $\frac{2|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: 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 $\frac{2|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: 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 $\frac{2|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: 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 $\frac{|\mathbb{G}_{in}\setminus\mathbb{V}_{in}|}{|\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})|} \leq \frac{|\mathbb{G}|}{|\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})|}$.

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 $\frac{|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: 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 $\mathsf{X}_{\mathsf{ZeroTest}}$ 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 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 $\mathsf{X}_{\mathsf{TuPlookup}}$ 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,

- 9-TuPlookup for row'_M, col'_M, val'_M with table polynomials $trow_M$, $tcol_M$, $tval_M$ for $M \in \{A, B, C\}$ with advantage $\frac{(20+9)|\mathbb{G}|+19|\mathbb{H}|+3\max(|\mathbb{G}|,|\mathbb{H}|)+5|\mathbb{V}|+4}{|\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})|}$.

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 $\frac{|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: Checks base case $s(1) = 1$.
- $s(X) = s(\gamma X)$ over $\mathbb{V}$ with advantage $\frac{|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: 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 $\frac{2|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$: 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 2. 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 $\frac{2|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$.
- $col_M(X) = col'_M(X)s(X)$ over $\mathbb{G}$ with advantage $\frac{2|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$.

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 $\frac{3|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}$.

As before G_2 employs the zero test extractor $\mathsf{X}_{\mathsf{ZeroTest}}$ 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. All together, we have the following probability bound which completes Claim 2:

$$\frac{(20+9)|\mathbb{G}|+19|\mathbb{H}|+3\max(|\mathbb{G}|,|\mathbb{H}|)+5|\mathbb{V}|+4}{|\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})|}+\frac{4|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}+3\cdot\left(\frac{7|\mathbb{G}|}{\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})}\right)=\frac{54|\mathbb{G}|+19|\mathbb{H}|+3\max(|\mathbb{G}|,|\mathbb{H}|)+5|\mathbb{V}|+4}{|\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})|}$$

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 2. G_3 employs the Marlin extractor $\mathsf{X}_{\mathsf{Marlin}}$ 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 $\frac{8|\mathbb{G}|+1}{|\mathbb{F}\setminus(\mathbb{G}\cup\mathbb{H})|}$.

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:

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

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 $\mathsf{X}_{\mathsf{Marlin}}$ to extract x and outputs,

$$\left([inst_j, mem_j, w_j]_{j=0}^{n}, x_0 = [inst_0, mem_0], x_n = [inst_n, mem_n]\right).$$

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

□

6 Evaluation

In this section, we evaluate the efficiency of our proposed protocol. First, we detail our compilation to a zkSNARK including a number of optimizations. Then we provide a precise accounting of the costs associated with Mux-Marlin in comparison to alternate state-of-the-art approaches to proving machine execution.

6.1 Compilation

Using a sequence of standard compilers for univariate polyIOPs described in Section 3.2, the Mux-Marlin polyIOP (Figure 15) can be compiled to a zkSNARK.

The first compilation step compiles the polyIOP to a zero-knowledge polyIOP. The Mux-Marlin polyIOP satisfies the requirements for the zero-knowledge polyIOP compiler by specifying the evaluation domains for each prover-provided polynomial as the domain the polynomial was interpolated over, namely subsets of $\mathbb{H} \cup \mathbb{G}$. With the specified domains, Mux-Marlin (1) has its witness completely encoded within the evaluation domain of a provided polynomial oracle x, and (2) is domain-admissible in that, apart from the domain-admissible queries in Marlin (Figure 13 Step 4d) and the domain-admissible query in Mux-Marlin (Figure 15 Step 4d), it is fully composed of subprotocols that each reduce

Protocol	Prover computation	Verifier computation	Proof size
IVC-Univ: Nova	$(2n+22)$ v-MSM$(C\ell)$	2 pairings	13 $\mathbb{G}_1$, 8 $\mathbb{F}_q$
IVC-NonUniv: SuperNova	$2n$ v-MSM$(C)+22$ v-MSM$(C\ell)$	2 pairings	13 $\mathbb{G}_1$, 8 $\mathbb{F}_q$
Unroll-Univ: Phalanx-Marlin	7v-MSM$(C\ell)+15$v-MSM$(nC\ell)$	2 pairings	13 $\mathbb{G}_1$, 8 $\mathbb{F}_q$
Unroll-noZK: Marlin	22v-MSM(nC)	2 pairings	13 $\mathbb{G}_1$, 8 $\mathbb{F}_q$
Unroll-ZK: Mux-Marlin (This work)	102v-MSM(nC)+50v-MSM$(C\ell)$ +5v-MSM$(n\max(C,\ell))$+13v-MSM(n)	2 pairings	125 $\mathbb{G}_1$, 116 $\mathbb{F}_q$

Figure 16: Accounting of dominant costs in proof protocols for machine execution where ℓ 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. $\mathbb{G}_1$ refers to one group in a pairing-friendly group and $\mathbb{F}_q$ 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.

down to ZeroTest subprotocols. The zero test protocols are domain-admissible when the randomness is restricted to be chosen from $\mathbb{F} \setminus \mathbb{H} \cup \mathbb{G}$.

Then, the zero-knowledge polyIOP can be compiled to a zkSNARK by instantiating the oracles with an appropriate polynomial commitment scheme and applying the Fiat-Shamir transform. For our accounting purposes, we consider the Marlin-KZG polynomial commitment scheme [CHM+20].

Compilation optimizations. Lastly, we apply a number of common optimization tricks, enumerated below, to further improve the efficiency characteristics of the protocol. All together, the optimizations reduced proving time by nearly $3\times$, proof size by over $2\times$, and verifier cost from 132 pairings to 2 pairings.

Parallelize zero tests. A random linear combination is used to combine multiple zero tests over the same group at the expense of a slight increase in soundness error. This reduces the number of quotient polynomials that are committed to one per group.

Batch verification on the same point. Under a single degree bound D, for n polynomials $(p_i)_{i\in[n]}$, the prover can open all evaluations to all n polynomials at the same point z using a random linear combination. This allows the prover to only send a single evaluation proof instead of n proofs, and allows the verifier to perform only two pairings (one pairing equation) for each evaluation point.

Batch verification on different points. Marlin-KZG opening verificaton consists of a pairing equation. A random linear combination is used to batch together pairing equations for opening verification to reduce the verifier cost to only two pairings in total.

Reducing the cost of hiding commitments. In our protocol, a polynomial is evaluated at no more than four different points. Hence we can relax the hiding property to hide only four evaluations per polynomial. This means for a polynomial p, the shifted polynomial $\bar{p}$ for degree bound check is of degree 4 and we can eliminate the one additional commitment and opening since they are negligible compared to v-MSM of $deg(p)$.

6.2 Accounting Comparison

We provide precise accounting of the prover computation, verifier computation, and proof size in Figure 16. 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+20]; we do not report on the additional scalar FFTs required. To keep the comparisons relatively on similar grounds, we choose baselines highlighting alternate state-of-the-art strategies for machine execution instantiated with Marlin-based proof systems (using the Marlin-KZG polynomial commitment), meaning the prover operations similarly include MSMs (and FFTs), result in constant-size proofs, and use a trusted setup. We do not attempt to compare to proof systems based on other multivariate or FRI-derived polynomial commitments.

Comparison to IVC baselines. We compare to two IVC approaches. First we consider the standard recipe of applying IVC to a universal constraint system for the instruction set [BCTV14a]; we instantiate this approach with the state-of-the-art Nova protocol for IVC [KST22] paired with Marlin to achieve constant-size succinct proofs. Second we consider the SuperNova protocol for non-uniform IVC [KS22] to dispense with the overhead of universal constraints; again, Marlin is used to achieve succinctness.

Focusing on SuperNova as it achieves the same asymptotic costs as Mux-Marlin, the constant-factor costs of proving are $5-50\times$ lower than Mux-Marlin (depending on ratio of n and ℓ) due to lower costs for folding instances over directly proving. Another benefit of IVC-based solutions, not evaluated concretely, is the memory requirement is on the order of $\tilde{O}(C+\ell)$ as opposed to $\tilde{O}((n+\ell)C)$. On the other hand, the security of IVC solutions rely on a heuristic step that does not follow in the random oracle model. Further, when using folding techniques like SuperNova which require homomorphic polynomial commitments, efficiency is greatly improved when using cycles of elliptic curves [BCTV14a] which are currently non-standard and incur additional overheads not captured in our accounting.

Comparison to unroll baselines. We similarly compare to two unroll approaches. First we consider an unroll approach using universal constraints in which the universal constraint system is repeated for each execution step [BCTV14b]. As in previous protocols addressing this setting, a more efficient proving protocol can be designed to take advantage of the constraint repetition [BCG+18, BCG+19]. We compare against a state-of-the-art protocol to proving constraints with repetition called Phalanx [TKPS22] which we adapt to work with Marlin. Lastly, we consider the baseline of simply proving the unrolled execution (without universal constraints) using Marlin; this does not provide program execution zero-knowledge but avoids the asymptotic costs associated with universal constraints. Mux-Marlin incurs $\approx 8\times$ overhead over the non-zero-knowledge unroll baseline. However, the key contribution of Mux-Marlin is that it supports zero-knowledge. When comparing Mux-Marlin to Phalanx with universal constraints—the only prior unroll approach to providing zero-knowledge—Mux-Marlin achieves similar proving costs at ≈ 100 executed instructions, and scales significantly better beyond due to avoiding universal constraints. For example, with $\ell = 200$, $n = 1000$, Mux-Marlin is $\approx 15\times$ more prover efficient.

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); part of this work was completed while he was a visiting student at Stanford University. 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 *CRYPTO (4)*, volume 12828 of *Lecture Notes in Computer Science*, pages 65–91. Springer, 2021.

[AFK22] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In *TCC (1)*, volume 13747 of *Lecture Notes in Computer Science*, pages 113–142. Springer, 2022.

[AHG22] Diego F. Aranha, Youssef El Housni, and Aurore Guillevic. A survey of elliptic curves for proof systems. *IACR Cryptol. ePrint Arch.*, page 586, 2022.

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2002.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *FOCS*, pages 2–13. IEEE Computer Society, 1992.

[Bab85] László Babai. Trading group theory for randomness. In *STOC*, pages 421–429. ACM, 1985.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO (3)*, volume 11694 of *Lecture Notes in Computer Science*, pages 701–732. Springer, 2019.

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. *IACR Cryptol. ePrint 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 *ITCS*, pages 326–349. ACM, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC*, pages 111–120. ACM, 2013.

[BCG+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 *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[BCG+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 *ASIACRYPT (1)*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626. Springer, 2018.

[BCG+19] Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner. Linear-size constant-query iops for delegating computation. In *TCC (2)*, volume 11892 of *Lecture Notes in Computer Science*, 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 *IEEE Symposium on Security and Privacy*, pages 947–964. IEEE, 2020.

[BCL+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+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+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+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.

[CBBZ22] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. *IACR Cryptol. ePrint Arch.*, page 1355, 2022.

[CCG+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.

[CFH+15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 253–270. IEEE Computer Society, 2015.

[CGG+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+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.

[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.

[FL14] Matthew Fredrikson and Benjamin Livshits. Zø: An optimizing distributing zero-knowledge compiler. In *USENIX Security Symposium*, pages 909–924. USENIX Association, 2014.

[GGHK22] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose ς-protocols for disjunctions. In *EUROCRYPT (2)*, volume 13276 of *Lecture Notes in Computer Science*, pages 458–487. Springer, 2022.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[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$^+$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.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304. ACM, 1985.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer, 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[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 oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, page 953, 2019.

[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.

[KPS18] Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. xjsnark: A framework for efficient verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 944–961. IEEE Computer Society, 2018.

[KS22] Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *IACR Cryptol. ePrint Arch.*, page 1758, 2022.

[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.

[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.

[OBW20] Alex Ozdemir, Fraser Brown, and Riad S. Wahby. Unifying compilers for snarks, smt, and more. *IACR Cryptol. ePrint Arch.*, page 1586, 2020.

[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.

[TKPS22] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath Setty. Transparency dictionaries with succinct proofs of correct operation. In *NDSS*. The Internet Society, 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$^+$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$^+$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.

[ZBK$^+$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$^+$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`.