RAMenPaSTA: Parallelizable Scalable Transparent Arguments for RAM Programs

Khai Hanh Tang[1], Nhat Minh Pham[2], and Chan Nam Ngo[3]

[1] Nanyang Technological University, 50 Nanyang Ave, Singapore 639798, Singapore
[2] Orochi Network Co. Ltd., 88 B2 Street, An Loi Dong Ward, Ho Chi Minh City, Vietnam
[3] Privacy + Scaling Explorations

Abstract. Recent advances in Zero-Knowledge Proof and Argument (ZKP/ZKA) Systems allow efficiently verifiable computation in the circuit-based model (where programs can be represented as Boolean or arithmetic circuits). However, the circuit-based model is not widely popular due to its unsuitability for big programs (as the circuit size is linear to the program's size). Hence, the research community began looking into the Random-Access Machine model of programs, namely RAM programs, which is better suited for general-purpose programming. Consequently, a flurry of work began researching to construct ZKP protocols for the correct execution of RAM programs. In this paper, we also develop ZKP/ZKA for RAM programs, with a particular focus on two properties: (i) *parallelizability*, which significantly reduces prover (and verifier) computation, and (ii) *genericity*, which makes the construction requires minimal assumptions and can be instantiated with any existing ZKP protocols. To our knowledge, previous ZKP constructions for RAM programs either (i) are not known to support proving or verifying in parallel, (ii) require making non-black-box use of specific SNARK constructions, or (iii) even require proving computation of random oracle.
To this end, we propose the so-called Conditional Folding Scheme (CFS) as a building block to construct ZKP/ZKA for RAM programs. We provide a non-trivial practical construction for our CFS that is natively parallelizable, which significantly brings the runtime of proof generation down to $\mathcal{O}(W \log N)$ (compared to $\Omega(W \cdot N)$ in previous works), where W is the witness size of the heaviest operation, and N is the number of execution steps. We rigorously prove the security of our CFS (also for the case of folding in parallel). Our scheme can be made non-interactive using the Fiat-Shamir transform. Our CFS is generic and does not require a trusted setup (yielding a "transparent" argument). It can be adapted to construct Parallelizable Scalable Transparent Arguments of Knowledge for RAM Programs that we dub RAMenPaSTA.

Keywords: Zero-Knowledge Proofs · RAM Programs · Folding Scheme

Table of Contents

1 Introduction

Zero-knowledge proof for RAM programs. Zero-knowledge proofs (ZKP) allow a party, called the prover, to convince another party, called the verifier, that the prover knows a witness of a certain statement without disclosing any information beyond its validity. Over the past few years, extensive research has been conducted to construct efficient ZKP protocols [IKOS07, GGPR13, Gro16, BBB+18, XZZ+19, GWC19, BSBHR19, BSCR+19, CHM+20, MAGABMMT23] for programs. These ZKP constructions mainly focus on proving computation of *circuit-based programs*, i.e., those that can be represented as Boolean or arithmetic circuits. However, circuit-based programs are only suitable for small programs (otherwise, the circuits would be huge, especially when the program has loops or branches), are not developer-friendly, and thus yield a bad development experience. On the contrary, *RAM program*s, where the programs can be represented as a sequence of computations in the random-access machine (RAM) model, are better for big programs and much more friendly to those that are used to programming the general-purpose CPU. Some results [BSCG+13, WSR+15, MRS17, BCG+18, FKL+21, DdSGOTV22, GHAK23] set their focus on building ZKP protocols for proving the correctness of RAM programs instead. Our goal of this work is also to design a ZKP for RAM programs, with a particular interest in two properties: *genericity* and *parallelizabiity*.

Genericity and Standard Assumptions. Although there are various attempts to build ZKP for RAM programs, many of them have to *make non-black-box use of primitives* such as Succinct Non-Interactive Argument of Knowledge (SNARKs) [BCCT12, GGPR13, Gro16], and global random oracles [CJS14]. Several constructions using Incremental Verifiable Computation (IVC) [KS22, BC24b, AS24] even require proving computation of random oracle (RO). Thus, they need to prove the RO's computation as a circuit of the employed hash function heuristically realizing RO (a.k.a *recursion heuristic*) [KS22, BC23, BC24b]. (Note that proving the output of RO instantiated by a hash function is not a standard way of guaranteeing security since RO is an ideal object that any function cannot represent. Usually, we heuristically realize RO by a hash function via Fiat-Shamir transform [FS87] since we do not need to prove the correct execution of the hash function.) Consequently, all the constructions mentioned above have to rely on non-standard assumptions found in SNARK and IVC, such as trusted setup, Algebraic Group Model (AGM) [FKL18], recursion heuristic, and might not be compatible with other ZKP techniques. For example, those using sumcheck [LFKN92, BSCR+19] have to make a non-black-box use of techniques and assumptions found in SNARKs for sumcheck constraints and is only proven secure in the AGM. This is due to employing polynomial commitment schemes whose instantiations are based on AGM. On the other hand, a generic construction (i.e., making only black-box use of cryptographic primitive) would be more beneficial since it can be deployed with any ZKP and does not have to rely on specific or non-standard assumptions.

Notably, only [FKL+21, DdSGOTV22, YH24] and Dora [GHAK23] have managed to propose efficient constructions achieving zero-knowledge while requiring only a homomorphic *commitment scheme* and a *ZKAoK* protocol as building blocks, and do not require any non-standard assumption, which is considered *optimal* in sense of genericity. However, these constructions suffer from a common efficiency issue as follows. Consider a RAM program running in N steps for some positive integer N. To prove the correctness of this RAM program, the prover and verifier have to engage in an interactive protocol that requires $\Theta(N)$ rounds of communication in sequential. Therefore, their protocols suffer from $\Omega(N)$ communication, computation, and especially network latency cost. Additionally, requiring sequential execution means it is uncertain whether these protocols could benefit from possible optimizations like producing the proofs in parallel, to be clarified below.

Parallelizability. A program's execution is usually a sequence of computations. One way to optimize the efficiency is to construct a ZKP for RAM program executions by making it parallelizable, e.g., generating the proof in parallel by separating the proofs of instructions and then unifying them into a single proof.

ZKPs for RAM programs with parallelizability have been mentioned and even considered in several constructions, e.g., SuperNova [KS22] Mangrove [NDC+24] and SPARK [EFKP20]. Moreover, there exist several distributed ZKPs [WZC+18, XZC+22, LXZ+24] and the Proof-Carrying Data (PCD) supporting parallelization [NDC+24] for general arithmetic circuits. Note that if we employ an existing distributed ZKP protocol, on input a RAM program's execution trace, and view each prover as a thread, we could get a parallelizable ZKP for a RAM program execution. Unfortunately, except [EFKP20], all the above protocols either have to make non-black-box use

on existing SNARK constructions, such as [Gro16, ZXZS20, GWC19], or require recursion heuristic in the case of [KS22, NDC+24], and hence, they all suffer from having to rely on non-standard assumptions. Finally, while [EFKP20] only relies on hash functions, it mentions nothing about ZK, and it is unknown whether [EFKP20] could achieve this property. Also, SPARK's parallel strategy only reduces the prover time to $\mathcal{O}(WN)$. This is less efficient for prover time compared to [NDC+24] and distributed ZKP since their strategy allows reducing the prover time down to $\mathcal{O}(W \log N)$ or $\mathcal{O}(W \log W)$.

Research Questions. As analyzed above, there has not been an efficient ZKP for RAM programs (with ZK) that achieves both parallelizability and genericity or relies only on standard assumptions. Therefore, we ask the following questions.

Q.1 Can we construct a parallelizable ZKP/ZKA for RAM programs that require only generic primitives, i.e., a homomorphic commitment scheme and a ZKAoK protocol, as building blocks?
Q.2 Can we rely only on standard assumptions (including a transparent setup)?
Q.3 And what would be the achievable efficiency when parallelizing?

1.1 Our Contributions

We answer Q.1 and Q.2 affirmatively by proposing a ZKAoK, dubbed RAMenPaSTA, for RAM programs, that is parallelizable, generic with standard assumptions, and possibly transparent in the setup[4]. RAMenPaSTA only requires black-box use of a homomorphic commitment scheme and a compatible ZKAoK.

RAMenPaSTA is efficient and scalable (to answer Q.3). For "scalable", we follow the definition in [BSBHR19], requiring that the prover cost should be at most $\tilde{\mathcal{O}}(WN)$ and the verifier cost should be at most $\mathsf{poly}(\log(WN))$. Later, when analyzing the cost, we will see that, RAMenPaSTA is indeed scalable when executed in parallel. Compared to other constructions with minimal assumptions, [FKL+21, DdSGOTV22, YH24, GHAK23], RAMenPaSTA has better efficiency since it achieves parallelizability in proving the RAM programs. Consider a RAM program of N computation steps. To achieve genericity and efficiency, instead of running a ZKAoK and directly proving all these N steps [BCG+17, FKL+21, DXNT23], we follow the Folding Scheme [KST22] paradigm and propose *Conditional Folding Scheme* (a variant of folding scheme in [KST22]) to fold (in parallel) all N instance-witness pairs *with some conditions enforced between them*, representing the N steps *and the necessary memory*, into a single folded instance-witness pair. Then, we only need to apply the ZKAoK to prove the validity of the folded pair, implying the validity of all N instance-witness pairs. Details of our contributions are as follows.

Conditional Folding Scheme (CFS) and Generic Construction. Folding schemes such as Nova [KST22] and subsequent works [BCMS20, BCL+21, BC23, KST22, KS22, KS23b, LXZ+24, ZGGX23] do not enforce any condition between two instance-witness pairs when folding them. In such schemes, to enforce some conditions between the folded pairs, they employ recursion heuristics and possibly cycles of curves [H+] for consistency check of conditions validity, limiting the choices of assumptions and proof techniques.

To avoid those issues, we propose *Conditional Folding Scheme* (CFS). Our CFS requires some conditions between two to-be-folded instance-witness pairs to hold in order to fold them successfully. This is captured by the notion of knowledge soundness in our definition of CFS, i.e., the extractor can extract valid witnesses with conditions satisfied. We then provide a generic construction of CFS (called generic CFS) for folding two relaxed R1CS (rR1CS) instance-witness pairs [KST22] *with some conditions between them captured by an additional rR1CS*. Our generic CFS is non-trivial and does not rely on recursion heuristics and cycles of curves. We will discuss more in detail in Section 2.2. Interestingly, compared with [KST22], we only fold by leveraging the homomorphism of commitment schemes. When applying a ZKAoK for the folded instance, the validity of the proof implies the validity of both to-be-folded instances (by knowledge extractors).

With the above generic CFS, we can extend to fold instance-witness pairs of an N-step computation. Here, an N-step computation is a sequence of N consecutive computation steps that, between any two consecutive steps, there are some relationships between them, e.g., the output of

[4] The term RAMenPaSTA is from parallelizable scalable transparent arguments for RAM programs, i.e., "RAM and PaSTA". Hence, we named RAMenPaSTA.

the former step and the input of the latter one must be consistent. The folding process follows a binary tree structure in a way that, by viewing the N computations as the N leaf nodes of the tree, we can fold the nodes by appropriately picking two nodes and folding them into a new node following the tree structure until reaching the root node.

Supporting Tuple Permutation and Tuple Lookup. We adapt our CFS for folding permutation and lookup arguments. For permutation arguments, we assume that there are two N-element sequences where each sequence is distributed into N instance-witness pairs of the generic CFS. Then, we can run the folding process and prove the permutation at once. A similar way is applied to folding lookup arguments that N elements are a subset of a public sequence of size T.

Constructing RAMenPaSTA and Analysis. Consequently, we construct a ZKAoK for RAM programs, dubbed RAMenPaSTA, that is parallelizable and generic from homomorphic commitment schemes and a compatible ZKAoK. Here, permutation and lookup arguments aim to support the consistency of memory accesses, correct instruction selections, and correct instruction computations by embedding PLONK's arithmetization into rR1CS.

Our RAMenPaSTA supports parallelization for efficiency analysis, and its costs are relatively efficient. First, it does not require a universal circuit. Secondly, when executing the protocol in sequential, it manages to achieve linear size in terms of $W \cdot N + T$ in all costs, which is similar to [GHAK23, FKL+21, DdSGOTV22], where N is the number of execution steps and T is the number of instructions, and W is the witness size for each instruction. However, folding in parallel, our scheme incurs only $\mathcal{O}(W \log N + T)$ prover and verifier time, dominated by the cost of executing the CF schemes in parallel. This is much better than Dora since their prover time cannot be lower than $\Omega(W \cdot N)$ since their protocol needs both prover and verifier to execute the steps sequentially. When compared to distributed ZKP protocols [WZC+18, XZC+22, LXZ+24], our scheme is inferior in terms of performance due to the $\mathcal{O}(\log N)$ factor. However, our scheme is generic and can support any ZKP instantiations, while [GHAK23, FKL+21, DdSGOTV22] have to make a non-black-box use of a specific SNARK instantiation like [Gro16] or PLONK [GWC19] for such efficiency.

Regarding instantiation of RAMenPaSTA, we provide a potential instantiation of RAMenPaSTA presented in Figure 7 from compressed Σ-protocol theory (Section 7.1). Table 1 compares our work and several existing ZKP constructions for the RAM programs.

Table 1. A comparison of our work (RAMenPaSTA) with several existing ZKP for RAM programs. TS means "trusted setup", C is the size of the commitment to the witness, W is the witness size in a single step, N is the number of computation steps, M is the memory size and T is the number of instructions in the RAM program. While Pianist and Mangrove do not focus on proving RAM programs, it can be used to prove any NP relation in parallel, including RAM programs.

Work	Proof size	Prover time	Verifier time	Assumptions
Franzese et al. [FKL+21]	$\mathcal{O}(C(N+T))$	$\mathcal{O}((N+M)W \log W)$	$\mathcal{O}((N+M)W \log W)$	Hom.Com+ZKAoK
Yang et al. [YH24]	$\mathcal{O}(C(N+M))$	$\mathcal{O}(W(N+M))$	$\mathcal{O}(W(N+M))$	Hom.Com+ZKAoK
Dora [GHAK23]	$\mathcal{O}(C(N+T))$	$\mathcal{O}(W(N+T))$	$\mathcal{O}(C(N+T))$	Hom.Com.+ZKAoK
Pianist [LXZ+24]	$\mathcal{O}(N)$	$\tilde{\mathcal{O}}(W)$ per thread	$\mathcal{O}(1)$	DLOG+AGM+TS
MUXProof [DXNT23]	$\mathcal{O}(1)$	$\tilde{\mathcal{O}}((N+T)W)$	$\mathcal{O}(1)$	DLOG+AGM+TS
Dutta et al. [DGP+24]	$\mathcal{O}(1)$	$\tilde{\mathcal{O}}(WN^2)$	$\mathcal{O}(1)$	DLOG+AGM+TS (No ZK)
Mangrove (Parallel) [NDC+24]	$\mathcal{O}(1)$	$\mathcal{O}(W \log N)$	$\mathcal{O}(\log(WN))$	Recursion Heuristic
Nebula [AS24]	$\mathcal{O}(\log^c W)$	$\mathcal{O}(WN)$	$\mathcal{O}(\log^c W)$	Recursion Heuristic (No ZK)
Ishai et al. [IOS23]	$\mathcal{O}(\log^c M)$	$\tilde{\mathcal{O}}(N \log N + M \log M)$	$\mathcal{O}(\log^c M)$	Hash Func. (No ZK)
SPARK [EFKP20]	$\mathcal{O}(\log^c N)$	$\mathcal{O}(WN)$	$\mathcal{O}(\log^c N)$	Hash Func. (No ZK)
Ours	$\mathcal{O}(C(N+T))$	$\mathcal{O}(W(N+T))$	$\mathcal{O}(C(N+T))$	Hom.Com.+ZKAoK
Ours (Parallel)	$\mathcal{O}(C(N+T))$	$\mathcal{O}(W \log N + T)$	$\mathcal{O}(C \log N + T)$	Hom.Com.+ZKAoK

1.2 Related Works

Incrementally Verifiable Computations (IVCs). An IVC allows a prover to prove to a verifier the correct execution of a sequential computation. IVC can be either constructed from recursive composition of SNARK [BCCT13, BSCTV14a], or accumulation/folding [BCMS20, BCL+21, KST22, KS22, KS23b, LGZ+23, ZGGX23, BC24a, KS23a, AS24]. Ben-Sasson et al. [BSCTV14a] showed that it is possible to handle RAM programs using IVC, making it one of the major approaches for proving the correctness of RAM programs. Their work was later improved by [KS22, BC23, BC24b, AS24] to achieve better efficiency. However, accumulation and folding-based IVC constructions require recursion heuristic, i.e., proving computations of random oracle, which is a non-standard way to achieve security as pointed out above, and this was proven to be theoretically impossible [HAN23, BCG24]. This means these IVC-based constructions do not achieve

provable security even in ROM. While [BCCT13] achieves security without recursion heuristic due to using SNARKs not requiring query to RO, it instead relies on the "knowledge-of-exponent" assumption [BP04], which is another strong assumption and might only be instantiated with discrete log-based ZKP protocols using such assumption such as [Gro10, GGPR13, Gro16] (and these protocols also require trusted setup).

"Unroll"-Based Approaches for Proving RAM Programs. Another way is to "unroll" the whole RAM program into a single set of constraints and then prove its correctness using existing ZKAoKs such as [WYKW21, IKOS07, GGPR13, GWC19] (the term "unroll" has been used in [GHAK23,DXNT23]). Many works [BFR+13,BSCTV14b,WSR+15,BSBC+17,MRS17,BCG+18, ZGK+18, EFKP20, FKL+21, DdSGOTV22, IOS23, DXNT23, YH24, CGG+24, JJ24, DGP+24] have followed this approach to handle RAM programs. The advantage is that it avoids the recursion heuristic problem in the IVC-based approach. However, current unroll-based constructions suffer from inefficiency (especially prover time) or require a non-black-box use of cryptographic primitives. For inefficiency, many works [BFR+13, BSCTV14b, WSR+15, DXNT23, CGG+24, JJ24, DGP+24] rely on (i) specific SNARK constructions such as [Gro16, GWC19, CHM+20], (ii) quasilinear PCP [BS08] or (iii) Merkle tree commitment [IOS23], therefore these works incur at least $\Omega(N \log N)$ prover time (for [IOS23], the prover cost is quasilinear in both the memory size M and the number of steps N). For non-genericity, all constructions mentioned above, except [FKL+21, DdSGOTV22, IOS23, YH24], have to rely on *non-standard* assumptions found in specific SNARKs (or other primitives) such as AGM [FKL18], trusted setup or global random oracle [CJS14]. Finally, while [EFKP20,FKL+21,DdSGOTV22,YH24] achieve generic constructions and enjoy linear prover time and proof size, [EFKP20] does not provide zero knowledge for the execution because they need to provide a short digest of the memory in intermediate steps (see Appendix A), and it is unknown whether [FKL+21, DdSGOTV22, YH24] could support proving in parallel and the way they handle (potentially different) instructions during the proving process. Even though [FKL+21] and [DdSGOTV22] claim to support proving instructions, they need to employ universal circuits in each step to prove instruction executions, which is rather expensive.

Parallel/Distributed ZKP. We consider the prover and verifier, each having $\mathcal{O}(N)$ threads, and can execute the folding process in parallel with these threads. This can be seen as a special case of distributed ZKPs, where each thread can be seen as a single prover. Thus, by using a distributed ZKP protocol on input a RAM program, we could get a parallelizable ZKP protocol for RAM program execution. So far, only [WZC+18, XZC+22, LXZ+24] have attempted to propose concrete distributed ZKP protocols by distributing the constraints of SNARKs in [Gro16, ZXZS20, GWC19], respectively, to each prover. Compared to ours, they all incur $\mathcal{O}(N)$ communication cost. For the total proving and verification time, these complexities of their constructions are independent of N. They thus could be more efficient than ours when $W \ll N$, since ours are both $\mathcal{O}(W \log N)$ (see Appendix A for more details). However, to achieve such efficiency, they must make non-black-box use of the underlying SNARKs and their non-standard assumptions.

Concurrent work: Dora. Dora [GHAK23] also leverages the folding scheme and proposes a new primitive called *ZK-bag* to construct a ZKP for RAM program without the need for universal circuits. However, their construction does not follow the IVC approach to achieve succinct proof size but instead designs an interactive proof system that aims to achieve linear communication cost and prover time like [FKL+21, YH24]. It also requires minimal assumption since only a homomorphic commitment scheme is needed. This and the folding scheme make Dora the most similar work to ours. However, whether Dora could support proving in parallel is unknown (see Appendix A for more details), meaning that it might not receive the benefits of reduced proving and verification time from parallelization.

1.3 Paper Organization

The paper is organized in the following structure.

- Section 1 is the introduction of the paper. The related work in this section is then extended in Appendix A
- In Section 2, we describe the technical overview of our result.
- In Section 3, we present the necessary backgrounds, including commitment schemes, the interactive protocol for folding relaxed R1CS instance-witness pairs, and hierarchical structures. Extended preliminaries can be found in Appendix B.

– In Section 4, we formally define CFS's syntax and security properties.
– In Section 5, we propose a generic CFS $\mathcal{CF}_{\mathsf{gnr}}$. Its security proof and detailed efficiency analysis can be found in Appendix C.
– In Section 6, we propose a Π_{RP} employing the CFS described in Section 5 as a building block. Its security proof can be found in Appendix D.
– In Appendix 7, we discuss potential instantiations of Π_{RP} described in Section 6.

2 Technical Overview

Notation for Hidden and Committed Secrets. In this technical overview, whenever writing $[\![A]\!]$ for an object A (either a vector or a tuple of vectors), we understand that A is committed and hidden. Moreover $[\![\cdot]\!]$ is homomorphic, i.e., we can compute $[\![\alpha \cdot A + \beta \cdot B]\!] := \alpha \cdot [\![A]\!] + \beta \cdot [\![B]\!]$ for some scalars α, β.

2.1 Defining CFS

Our definition of CFS follows the paradigm of Folding Scheme in [KST22]. Let $\mathcal{R}$ and $\mathcal{R}_{\mathsf{cond}}$ be NP relations. In Nova, Folding Scheme is used to fold two instance-witness pairs $(I_0, Z_0), (I_1, Z_1)$ into a single pair $(I, Z) \in \mathcal{R}$ iff $(I_0, Z_0), (I_1, Z_1) \in \mathcal{R}$. Our definition is also similar, except with one additional condition that $(I, Z) \in \mathcal{R}$ iff $(I_0, Z_0), (I_1, Z_1) \in \mathcal{R}$ **and there exists a witness** W **satisfying** $(I_0, I_1;\ Z_0, Z_1, W) \in \mathcal{R}_{\mathsf{cond}}$. For example, suppose Z_i contains the input and output of the sequential computation process. In this case, $\mathcal{R}_{\mathsf{cond}}$ simply captures the condition "the output of the current step to be exactly the input of the next step" and W is the intermediate witness for this condition.

Hence, we define CFS as a triple containing an algorithm $\mathsf{CF.Setup}$ and two protocols $\mathsf{CF.Fold}$ and $\mathsf{CF.Prove}$. $\mathsf{CF.Setup}$ returns the public parameter used for folding and proving. $\mathsf{CF.Fold}$ allows interactively folding the two pairs above into a single instance-witness pair (I, Z). Finally, $\mathsf{CF.Prove}$ allows the prover and verifier to check the validity of (I, Z), indicating whether $(I_0, Z_0), (I_1, Z_1) \in \mathcal{R}$ and $(I_0, I_1;\ Z_0, Z_1, W) \in \mathcal{R}_{\mathsf{cond}}$. For security, we need CFS to be knowledge-sound, i.e., there exists an extractor that, given I_0, I_1 and with rewinding capability, can extract the witnesses Z, Z_0, Z_1, W satisfying the conditions above. We also define the honest-verifier zero-knowledge (HVZK) property, modeled by a PPT simulator, to guarantee that the privacy of those witnesses is not compromised.

2.2 Generic Construction of CFS (Generic CFS)

For folding two instance-witness pairs, we denote the witness of the i-th pair as $\mathbb{z}_i$ for $i \in \{0, 1\}$. The witness $\mathbb{z}_i$ is a tuple of some components (to be clarified below). Here, the prover commits to these components into the corresponding commitments. Hence, we write $[\![\mathbb{z}_i]\!]$ to indicate that *the prover keeps witness* $\mathbb{z}_i$ *while both prover and verifier keep commitments to components in* $\mathbb{z}_i$.
Structure of $[\![\mathbb{z}_i]\!]$ for $i \in \{0, 1\}$. For simplicity, consider a program with 2-step computation whose the step i, for $i \in \{0, 1\}$, has an execution trace as a witness $\mathbb{z}_i$. Moreover, there exist some conditions between these two steps. For instance, the output of step 0 equals the input of step 1. We come up with our design of instance-witness pairs for generic CFS. Initially, we propose

$$[\![\mathbb{z}_i]\!] = ([\![\mathbb{x}_i]\!], [\![\mathsf{front}_i]\!], [\![\mathsf{rear}_i]\!], [\![\mathbb{x}_i^\star]\!]) \tag{1}$$

where

– $[\![\mathbb{x}_i]\!]$ is an rR1CS instance-witness pair whose $\mathbb{x}_i$ represents the witness for the computation in step i;
– For $i \in \{0, 1\}$, front_i and rear_i are introduced s.t. rear_0 (in $\mathbb{z}_0$) and front_1 (in $\mathbb{z}_1$) contain those needed for the condition between $\mathbb{z}_0$ and $\mathbb{z}_1$. This condition is captured by a circuit taking as inputs rear_0 and front_1.
– $[\![\mathbb{x}_i^\star]\!]$ is another rR1CS instance-witness pair playing the role of an accumulator to accumulate the condition when folding. We will discuss how this accumulator works and why we need it shortly below.

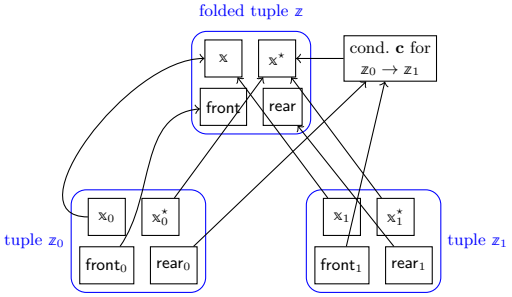

Fig. 1. Intuition for folding $\mathbb{z}_0, \mathbb{z}_1$ having the form of (1) with conditions.

Folding $[\![\mathbb{z}_0]\!]$ and $[\![\mathbb{z}_1]\!]$. Our proposed way to fold $[\![\mathbb{z}_0]\!]$ and $[\![\mathbb{z}_1]\!]$, with condition between rear_0 and front_1, into $[\![\mathbb{z}]\!] = ([\![\mathbb{x}]\!], [\![\mathsf{front}]\!], [\![\mathsf{rear}]\!], [\![\mathbb{x}^\star]\!])$ is as follows.

1. As $[\![\mathbb{x}_0]\!]$ and $[\![\mathbb{x}_1]\!]$ respectively represent single computations for steps 0 and 1, respectively, we simply apply Nova's folding to fold them by using a random challenge $\alpha_1 \in \mathbb{F}$, into $[\![\mathbb{x}]\!]$. Here, $\mathbb{x}$ can be understood as the folded computation trace of both steps 0 and 1.
2. Since we need to capture the condition between rear_0 and front_1, we still keep front_0 and rear_1 in $\mathbb{z}$ by setting $[\![\mathsf{front}]\!] := [\![\mathsf{front}_0]\!]$ and $[\![\mathsf{rear}]\!] := [\![\mathsf{rear}_1]\!]$. That is, front_0 and rear_1 are the input and output, respectively, of the folded computation $\mathbb{x}$ as it is the folded witness for both steps 0 and 1.
3. We now fold the condition. Since $[\![\mathbb{z}_0]\!]$ and $[\![\mathbb{z}_1]\!]$ are two consecutive instance-witness pairs, when folding them, we must have some condition between rear_0 and front_1. For simplicity, we assume that these conditions are captured by public R1CS matrices $\mathbf{A}'$, $\mathbf{B}'$ and $\mathbf{C}'$ such that

$$\mathbf{A}' \cdot \mathbf{c} \circ \mathbf{B}' \cdot \mathbf{c} = \mathbf{C}' \cdot \mathbf{c} \text{ for } \mathbf{c} = (\mathbf{p} \| \mathsf{rear}_0 \| \mathsf{front}_1 \| \mathbf{w}) \tag{2}$$

   where $\mathbf{p}$ is some public vectors supporting the computations, and $\mathbf{w}$ is an auxiliary witness supporting the computation of this R1CS. Regarding $\mathbf{w}$, for example, to compute x^3 where x is the input and x^3 is the output, we may need to compute x^2, included in $\mathbf{w}$. As shown in Nova [KST22], we can transform an R1CS instance-witness pair into an rR1CS one by setting the additional public value to be 1 and the error vector to be zero. Therefore, we can capture the above condition by an rR1CS instance-witness pair $[\![\mathbb{y}]\!]$, i.e., see Section 3.2 for the setting $[\![\mathbb{y}]\!] = (\mathbb{y}.u, \mathbb{y}.\mathsf{pub}, [\![\mathbb{y}.\mathbf{z}_1]\!], \ldots, [\![\mathbb{y}.\mathbf{z}_3]\!], [\![\mathbb{y}.\mathbf{e}]\!])$ where $\mathbb{y}.u = 1$, $\mathbb{y}.\mathbf{e} = \mathbf{0}$, $\mathbb{y}.\mathbf{z}_1 = \mathsf{rear}_0$, $\mathbb{y}.\mathbf{z}_2 = \mathsf{front}_1$ and $\mathbb{y}.\mathbf{z}_2 = \mathbf{w}$. By designing $[\![\mathbb{x}_0^\star]\!]$ and $[\![\mathbb{x}_1^\star]\!]$ to have the same structure with $[\![\mathbb{y}]\!]$, we can accumulate the condition, captured by $[\![\mathbb{y}]\!]$, by running Nova's folding twice to
   (a) fold $[\![\mathbb{x}_0^\star]\!]$ and $[\![\mathbb{x}_1^\star]\!]$, by using random challenge $\alpha_1 \in \mathbb{F}$, into $[\![\mathbb{y}']\!]$, and
   (b) fold $[\![\mathbb{y}']\!]$ and $[\![\mathbb{y}]\!]$, by using a random challenge $\alpha_2 \in \mathbb{F}$, into $[\![\mathbb{x}^\star]\!]$ playing the condition accumulator for $[\![\mathbb{z}]\!]$.

Remark 1. Notice that we fold $[\![\mathbb{x}_0]\!]$ and $[\![\mathbb{x}_1]\!]$ into $[\![\mathbb{x}]\!]$ and fold $[\![\mathbb{x}_0^\star]\!]$ and $[\![\mathbb{x}_1^\star]\!]$ into $[\![\mathbb{x}^\star]\!]$ by using the same α_1. These two foldings are arguably independent (i.e., no common components). Using the same challenge α_1 for both foldings does not violate the security and helps save the communication cost.

For better visualization of our folding process, we refer the readers to Figure 1. Notice that this process of folding $[\![\mathbb{z}_0]\!]$ and $[\![\mathbb{z}_1]\!]$ only leverages the homomorphism property of the commitment scheme according to Nova's folding. Moreover, regarding knowledge soundness, it is possible to extract the original witnesses by rewinding. HVZK is guaranteed due to the hiding property of the commitment scheme and HVZK of $\mathsf{CF.Prove}$ as a ZKAoK.

2.3 Extension to Folding N Instance-Witness Pairs

Let $N \in \mathbb{Z}_+$ where $N > 2$. We can extend the above folding process of two instance-witness pairs into a folding one for N instance-witness pairs, in which any two consecutive pairs have some condition between them. To this end, we first define the following notion of binary-tree-like hierarchical structures.

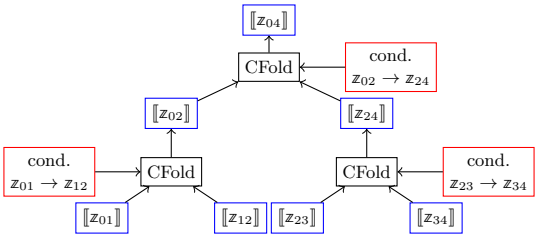

Fig. 2. An example of folding in parallel for a computation with 4 steps by following $\mathsf{HS} = \{(0,1),(1,2),(2,3),(3,4),(0,2),(2,4),(0,4)\}$. Each box named "CFold" represents the folding process in Figure 1. The prover with two parallel threads can fold two pairs (z_{01}, z_{12}) and (z_{23}, z_{34}) at depth 2 simultaneously as they can be folded independently. In this way, the prover obtains z_{02} and z_{24}, which are all the witnesses of depth 1. The prover then folds z_{02} and z_{24} into z_{04}, which is the final witness.

Definition 1 (Hierarchical Structures). *Let $N \in \mathbb{Z}_+$. A hierarchical structure $\mathsf{HS} = \{(l_i, r_i)\}_{i\in[2N-1]}$ can be arranged as a binary tree as follows.*

- *The leaf nodes are indexed by $(0,1),\ldots,(N-1,N)$ from left to right.*
- *If $(l,r) \in \mathsf{HS}$ and $l+1<r$, then there exists a unique j s.t. $l<j$, $j<r$ and $(l,j),(j,r) \in \mathsf{HS}$. We see that (l,j) and (j,r) are direct child nodes of (l,r). In this case, we define the set $\mathsf{HS}_{\mathsf{fold}}$ to contain all triples (l,j,r) s.t. $l+1<r$ and $(l,r),(l,j),(j,r) \in \mathsf{HS}$.*

Remark 2. Also, we can see from Definition 1 that the root node is $(0,N) \in \mathsf{HS}$. Moreover, (l,r) is a leaf node iff $r-l=1$. All nodes, except leaf nodes, are called non-leaf nodes.

Intuitively, we would like each non-leaf node $(l,r) \in \mathsf{HS}$ to contain a folded instance-witness pair $[\![z_{lr}]\!]$ of the $(l+1)$-th to r-th pairs (among the N pairs introduced above). The subscript "lr" is to indicate the node $(l,r) \in \mathsf{HS}$. To this end, initially, in each leaf node $(i-1,i) \in \mathsf{HS}$, we assign to it the i-th instance-witness pair, among the N pairs introduced above, and denote by $[\![z_{(i-1)i}]\!]$.

For each $(i,j,r) \in \mathsf{HS}_{\mathsf{fold}}$, suppose we already had $[\![z_{lj}]\!]$ and $[\![z_{jr}]\!]$. By applying the CFS in Section 2.2 to fold $[\![z_{lj}]\!]$ and $[\![z_{jr}]\!]$, we obtain $[\![z_{lr}]\!]$. In the end, we obtain the final folded instance-witness pair $[\![z_{0N}]\!]$ at the root node $(0,N) \in \mathsf{HS}$. For an example folding process, we refer the reader to Figure 2. From Figure 2, it can be seen that we can proceed with the two foldings independently in parallel. Hence, folding by following a binary-tree-like structure will help reduce the computation cost significantly as follows.

A Strategy for Folding in Parallel. As HS is a binary-tree-like structure, we can fold in parallel to make the protocol highly efficient. Folding in parallel can be done in many ways, depending on the strategy. Here, we describe a simple strategy with N threads. Assume that the height of HS is $H = \mathcal{O}(\log N)$. We divide the nodes of HS into different depths from 0 to H as follows. A node is of depth d if it is d-node away from the root node $(0,N)$. Note that all foldings, at any depth d, are all independent, i.e., no two foldings involve the same instance-witness pair. Hence, we *can use at most N threads* to fold all the instance-witness pairs of depth d *simultaneously* to receive the instance-witness pairs of depth $d-1$. In this way, after H steps, we can reach $[\![z_{0N}]\!]$. Again, we refer the reader to Figure 2 for an example. Following HS, the folding time can be at most $H \cdot t_{\mathsf{fold}} = \mathcal{O}(t_{\mathsf{fold}} \cdot \log N)$ where t_{fold} is the time for a single folding.

Remark 3. Note that, in a sequential computation process with N steps, we simply define $[\![z_{i(i-1)}]\!]$ to be the instance-witness pair of its i-th step. By using CFS in Section 2.1 (which includes folding conditions), we directly fold everything into $[\![\mathbf{z}_{0N}]\!]$ such that all the conditions between $[\![z_{i(i-1)}]\!]$ and $[\![z_{i(i+1)}]\!]$ are captured. In Nova [KST22], $[\![z_{i(i+1)}]\!]$ additionally **must contains the witness of correctly folding the previous steps** to capture the constraints between $[\![z_{i(i-1)}]\!]$ and $[\![z_{i(i+1)}]\!]$. Since the folding step requires querying RO, thus proving $[\![z_{i(i+1)}]\!]$ in Nova requires modeling RO as a circuit, which does not happen in our case.

2.4 Supporting Tuple Permutation and Tuple Lookup Arguments

We enhance the generic construction described in Section 2.2 to support proving tuple lookup and permutation arguments in the following sense. Let $s_1, s_2 \in \mathbb{Z}_+$. Assume that each instance-witness

pair $[\![\mathbb{z}_{(i-1)i}]\!]$ at the leaf node $(i-1,i)$ of HS contains $\mathbf{b}^{(0)}_{(i-1)i}, \mathbf{b}^{(1)}_{(i-1)i} \in \mathbb{F}^{s_1}$ and $\mathbf{b}^{(2)}_{(i-1)i} \in \mathbb{F}^{s_2}$. We consider the following conditions on $\mathbf{b}^{(0)}_{(i-1)i}, \mathbf{b}^{(1)}_{(i-1)i}$ and $\mathbf{b}^{(2)}_{(i-1)i}$.

- *Tuple Permutation Argument.* Show that $(\mathbf{b}^{(0)}_{(i-1)i})_{i\in[N]}$ is a permutation of $(\mathbf{b}^{(1)}_{(i-1)i})_{i\in[N]}$, i.e., there exists a permutation $\sigma : [N] \to [N]$ satisfying $(\mathbf{b}^{(0)}_{(i-1)i})_{i\in[N]} = (\mathbf{b}^{(1)}_{(\sigma(i)-1)\sigma(i)})_{i\in[N]}$.
- *Tuple Lookup Argument.* Let $T \in \mathbb{Z}_+$ and $(\mathbf{p}_j)_{j\in[T]}$ where $\mathbf{p}_j \in \mathbb{F}^{s_2}$ for $j \in [T]$. Show that $\{\mathbf{b}^{(2)}_{(i-1)i}\}_{i\in[N]} \subseteq \{\mathbf{p}_j\}_{j\in[T]}$.

The above tuple permutation and tuple lookup arguments are necessary for our construction of RAMenPaSTA, to be discussed in Sections 2.6 and 6, containing not only proving an N-step computation but also proving memory consistency (using technique from [BCG+18,ZGK+18,FKL+21]) and instruction lookup.

Tuple Permutation Argument Adapted from [Hab22]. We can adapt Haböck's technique [Hab22] to handle tuple permutation arguments by proving that

$$\sum_{i\in[N]} \mathsf{binv}^{(0)}_{(i-1)i} = \sum_{i\in[N]} \mathsf{binv}^{(1)}_{(i-1)i} \tag{3}$$

where $\mathsf{binv}^{(j)}_{(i-1)i} = (\tau + \langle \mathbf{b}^{(j)}_{(i-1)i}, (\omega^k)_{k\in[0,s_1-1]}\rangle)^{-1}$ $\forall j \in \{0,1\}, \forall i \in [N]$ for $\tau, \omega \xleftarrow{\$} \mathbb{F}$. Intuitively, compress each $\mathbf{b}^{(j)}_{(i-1)i}$ into a single value and prove (3).

Tuple Lookup Argument Adapted from [Hab22]. From Haböck's technique [Hab22], we can handle tuple lookup by proving the existence of $(\mathsf{mul}_j)_{j\in[T]} \subseteq \mathbb{F}$ s.t.

$$\sum_{i\in[N]} \mathsf{binv}^{(2)}_{(i-1)i} = \sum_{j\in[T]} \mathsf{pinv}_j \tag{4}$$

where, for $\chi, \psi \xleftarrow{\$} \mathbb{F}$, $\mathsf{binv}^{(2)}_{(i-1)i} = (\chi + \langle \mathbf{b}^{(2)}_{(i-1)i}, (\psi^k)_{k\in[0,s_2-1]}\rangle)^{-1}$ for $i \in [N]$ and $\mathsf{pinv}_j = \mathsf{mul}_j \cdot (\chi + \langle \mathbf{p}_j, (\psi^k)_{k\in[0,s_2-1]}\rangle)^{-1}$ for $j \in [T]$.

Augmenting $[\![\mathbb{z}_{lr}]\!]$ to Support Tuple Permutation and Tuple Lookup. Our observation is that LHS and RHS of (3) and LHS of (4) are computed by taking the sum of components with indices corresponding to the leaf nodes $\{(i-1,i)\}_{i\in[N]}$ of HS. However, the design of $[\![\mathbb{z}_{lr}]\!]$ in (1) is not sufficient for us to compute this sum. Therefore, our idea is to augment to each leaf node $(i-1,i)$ an additional commitment-opening pair $[\![\mathbf{s}_{(i-1)i}]\!]$ s.t., after folding following HS to compute $[\![\mathbf{s}_{lr}]\!] := [\![\mathbf{s}_{lj}]\!] + [\![\mathbf{s}_{jr}]\!]$ for $(l,j,r) \in \mathsf{HS}_{\mathsf{fold}}$, the final instance-witness pair $[\![\mathbb{z}_{0N}]\!]$ has $[\![\mathbf{s}_{0N}]\!] = \sum_{i\in[N]} [\![\mathbf{s}_{(i-1)i}]\!]$ where the sum is computed by leveraging the homomorphism of the underlying commitment scheme. For $i \in [N]$, if $\mathbf{s}_{(i-1)i}$ contains $\mathsf{binv}^{(j)}_{(i-1)i}$ $\forall j \in [3]$, then $\mathbf{s}_{0N}$ contains $\sum_{i\in[N]} \mathsf{binv}^{(j)}_{(i-1)i}$ for $j \in [3]$.

Unfortunately, regarding knowledge soundness, when having opening of $[\![\mathbf{s}_{lr}]\!]$, we cannot extract back those for $[\![\mathbf{s}_{lj}]\!]$ and $[\![\mathbf{s}_{jr}]\!]$. To guarantee extractability, we additionally introduce $[\![\mathbf{a}_{lj}]\!]$ and $[\![\mathbf{a}_{jr}]\!]$ respectively to $[\![\mathbb{z}_{lj}]\!]$ and $[\![\mathbb{z}_{jr}]\!]$. Then, when folding, we can compute $[\![\mathbf{a}_{lr}]\!] := [\![\mathbf{a}_{lj}]\!] + \alpha_1 \cdot [\![\mathbf{a}_{jr}]\!] + \alpha_1^2 \cdot ([\![\mathbf{s}_{lj}]\!] - [\![\mathbf{s}_{jr}]\!])$ for some random $\alpha_1 \xleftarrow{\$} \mathbb{F}$ which is quadratic in α_1. Observe that, for the same $[\![\mathbf{a}_{lj}]\!]$, $[\![\mathbf{a}_{jr}]\!]$, $[\![\mathbf{s}_{lj}]\!]$ and $[\![\mathbf{s}_{jr}]\!]$, if we rewind three times for with α_1 i.i.d., then we extract back $\mathbf{a}_{lj}$, $\mathbf{a}_{jr}$, $\mathbf{s}_{lj}$ and $\mathbf{s}_{jr}$. Specifically, if we have distinct $\{a^{(j)}_{lr}\}_{j\in[3]}$, $[\![\mathbf{s}_{lr}]\!] = [\![\mathbf{s}_{lj}]\!] + [\![\mathbf{s}_{jr}]\!]$, $[\![\mathbf{a}^{(j)}_{lr}]\!] := [\![\mathbf{a}_{lj}]\!] + \alpha_1^{(j)} \cdot [\![\mathbf{a}_{jr}]\!] + (\alpha_1^{(j)})^2 \cdot ([\![\mathbf{s}_{lj}]\!] - [\![\mathbf{s}_{jr}]\!])$ $\forall j \in [3]$ and the openings $\mathbf{s}_{lr}$, $\mathbf{a}^{(j)}_{lr}$ for $j \in [3]$, then we can extract the openings $\mathbf{a}_{lj}$, $\mathbf{a}_{jr}$ and $\mathbf{s}_{lj} - \mathbf{s}_{jr}$ respectively of $[\![\mathbf{a}_{lj}]\!]$, $[\![\mathbf{a}_{jr}]\!]$ and $[\![\mathbf{s}_{lj}]\!] - [\![\mathbf{s}_{jr}]\!]$. Since we have the opening $\mathbf{s}_{lr}$ of $[\![\mathbf{s}_{lr}]\!] = [\![\mathbf{s}_{lj}]\!] + [\![\mathbf{s}_{jr}]\!]$, by the binding of commitment scheme, we know that $\mathbf{s}_{lr} = \mathbf{s}_{lj} + \mathbf{s}_{jr}$. Hence, knowing both $\mathbf{s}_{lj} - \mathbf{s}_{jr}$ and $\mathbf{s}_{lj} + \mathbf{s}_{jr}$ helps recover $\mathbf{s}_{lj}$ and $\mathbf{s}_{jr}$ by using simple algebra.

Eventually, our *final design* of $[\![\mathbb{z}_{lr}]\!]$ is the following form

$$[\![\mathbb{z}_{lr}]\!] = ([\![\mathbb{x}_{lr}]\!], [\![\mathsf{front}_{lr}]\!], [\![\mathsf{rear}_{lr}]\!], [\![\mathbb{x}^{\star}_{lr}]\!], [\![\mathbf{s}_{lr}]\!], [\![\mathbf{a}_{lr}]\!]). \tag{5}$$

2.5 ZKP for RAM Program Execution: A High Level Overview

As we have described our building blocks, we now proceed to our final goal: Constructing a ZKP protocol to prove the correctness of a RAM program. We follow the notation of [FKL+21] with some modifications including the use of program counters $\{\mathsf{pc}_i\}_{i\in[0,N]}$ to select instructions. We now briefly describe the execution of an N-step RAM program. Let $M \in \mathbb{Z}_+$ be the memory size, $\mathsf{mem} = (\mathsf{mem}_i)_{i\in[M]}$ be the memory and $\mathcal{F}' = \{F'_j\}_{j\in[T]}$ be a public instruction set of cardinality T. Initially, we assume that program counter $\mathsf{pc}_0 = 1$ and val_0 is some initial input value. For $i \in [N]$, the program counter pc_{i-1} determines instruction $F_i := F'_{\mathsf{pc}_{i-1}}$ from $\mathcal{F}'$, which executes on input val_{i-1} and returns output $(\mathsf{pc}_i, \ell_i, \mathsf{val}_i, \mathsf{mop}_i)$ where $\ell_i \in [M]$ is an address in memory and $\mathsf{mop}_i \in \{0, 1\}$ is a memory access operation. If $\mathsf{mop}_i = 0$, it is identified as a READ operation. Otherwise, if $\mathsf{mop}_i = 1$, it is a WRITE operation. pc_i and val_i are used for the next step while ℓ_i, val_i, and mop_i determine the action for modifying the memory. Specifically, if $\mathsf{mop}_i = 1$ (i.e., WRITE), set $\mathsf{mem}_{\ell_i} := \mathsf{val}_i$. Otherwise, set $\mathsf{val}_i := \mathsf{mem}_{\ell_i}$ when $\mathsf{mop}_i = 0$ (i.e., READ). The output of the computation is val_N. See Appendix B.1 for a detailed description of RAM programs. Intuitively, with this description, the instance, witness, and major constraints for proving the correctness of a RAM program are as follows.

- *The public instance* of the RAM program is $(\mathsf{pc}_0, \mathsf{val}_{\mathsf{out}} = \mathsf{val}_N)$ and *the private witness* are the value $\mathsf{val}_{\mathsf{in}} = \mathsf{val}_0$ and the tuples $(\mathsf{pc}_i, \ell_i, \mathsf{val}_i, \mathsf{mop}_i)_{i\in[N-1]}$.
- 1st major constraint. $F'_{\mathsf{pc}_{i-1}}(\mathsf{val}_{i-1}) = (\mathsf{pc}_i, \ell_i, \mathsf{val}_i, \mathsf{mop}_i)$ $\forall i \in [N]$, also known as *correct execution of instruction*.
- 2nd major constraint. $F'_{\mathsf{pc}_{i-1}} \in \mathcal{F}'$ $\forall i \in [N]$, also known as *correct selection of instructions*.
- 3rd major constraint. If $\mathsf{mop}_i = 0$ and let j be the biggest integer less than i such that $\ell_j = \ell_i$ and $\mathsf{mop}_j = 1$, then it must hold that $\mathsf{val}_j = \mathsf{val}_i$. Also known as *memory consistency check*.

The 1st and 2nd major constraints are straightforward. The 1st constraint requires that the result of *each instruction is computed correctly*, while the 2nd constraint requires that the *next instruction must be valid*, i.e., it must be one of the specified instructions in $\mathcal{F}'$. The 3rd constraint captures the *correctness of reading/writing the values from/to the memory*. More specifically, let $\ell = \ell_i = \ell_j$ and suppose we read the value val from mem_ℓ, then this value must be equal to the last time mem_ℓ was written, which is at time time_j. To prove memory consistency, we use the technique from [BCG+18, ZGK+18, FKL+21] by forming two sequences $(\mathsf{macs}_i)_{i\in[N]} = (\ell_i, \mathsf{time}_i, \mathsf{val}_i, \mathsf{mop}_i)_{i\in[N]}$ and $(\mathsf{macs}'_i)_{i\in[N]}$ where $(\mathsf{macs}'_i)_{i\in[N]}$ is sorted via address then time log. Then it is well-known [BCG+18, FKL+21] that memory consistency holds iff the following system (6) holds.

$$\begin{cases} 1 \le \ell_i \le M \wedge \mathsf{mop}_i \in \{0,1\} & \forall i \in [N], \\ \mathsf{time}_{i-1} < \mathsf{time}_i & \forall i \in [2,N], \\ \big(\ell'_{i-1} < \ell'_i\big) \vee \big((\ell'_{i-1} = \ell'_i) \wedge (\mathsf{time}'_{i-1} < \mathsf{time}'_i)\big) & \forall i \in [2,N], \\ (\mathsf{macs}_{i-1} = 1) \vee (i - 1 > 1) & \forall i \in [2,N], \\ (\ell'_{i-1} = \ell'_i) \vee (\mathsf{mop}'_i = 1) & \forall i \in [2,N], \\ (\ell'_{i-1} \neq \ell'_i) \vee (\mathsf{val}'_{i-1} = \mathsf{val}'_i) \vee (\mathsf{mop}'_i = 1) & \forall i \in [2,N]. \end{cases} \tag{6}$$

See Appendix B.2 for a detailed description of (6). Note that the constraints in (6) can be "represented" with a fixed arithmetic circuit $\mathsf{C}_{\mathsf{mem}}$ such that $\mathsf{C}_{\mathsf{mem}}(i, \mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1}, \mathsf{macs}_i, \mathsf{macs}'_i) = 1$ iff (6) holds. Finally, since we sorted $(\mathsf{macs}_i)_{i\in[n]}$ into $(\mathsf{macs}'_i)_{i\in[n]}$, we must ensure that they are permutations of each other. Thus, the 3rd constraint can be split into two sub-constraints as follows.

- Sub-constraint 1: $\mathsf{C}_{\mathsf{mem}}(i, \mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1}, \mathsf{macs}_i, \mathsf{macs}'_i) = 1$ $\forall i \in [2, N]$.
- Sub-constraint 2: $(\mathsf{macs}'_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}_i)_{i\in[N]}$.

In the next section, we show how to employ the conditional folding scheme to handle all the major constraints above, hence achieving a generic, parallelizable and transparent argument for RAM programs, namely, RAMenPaSTA, which requires only a homomorphic commitment, a ZKAoK, and without recursion heuristic or any other non-black-box assumptions.

2.6 RAMenPaSTA

Finally, we show how to employ CFS to handle **both** memory and instructions in RAM programs. The result is a generic, parallelizable and transparent argument for RAM programs, which we name RAMenPaSTA.

(i) Handling Memory Consistency. Recall in Section 2.5 that, to prove memory consistency, we need to form two sequences of memory accesses $(\mathsf{macs}_i)_{i\in[N]} = (\ell_i, \mathsf{time}_i, \mathsf{val}_i, \mathsf{mop}_i)_{i\in[N]}$ and $(\mathsf{macs}'_i)_{i\in[N]} = (\ell'_i, \mathsf{time}'_i, \mathsf{val}'_i, \mathsf{mop}'_i)_{i\in[N]}$ which is a sorted version of $(\mathsf{macs}_i)_{i\in[N]}$. Then, there exists circuit $\mathsf{C}_{\mathsf{mem}}$ such that memory accesses $(\mathsf{macs}_i)_{i\in[N]}$ is consistent iff

(a) $\mathsf{C}_{\mathsf{mem}}(i, \mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1}, \mathsf{macs}_i, \mathsf{macs}'_i) = 1\ \forall i \in [2, N]$; and
(b) $(\mathsf{macs}'_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}_i)_{i\in[N]}$.

Notice that (a) is the condition between any two consecutive computation steps while (b) is a tuple permutation which is suitable with our design in Section 2.4.

(ii) Universally Realizing Instructions by PLONK Structures. We first need to have a description of each instruction. Our approach is to use PLONK's arithmetization [GWC19] (recalled in Appendix B.3) to encode each instruction $F'_j \in \mathcal{F}'$ as a vector $\mathsf{plkst}'_j \in \mathbb{F}^{n_{\mathsf{plk}}}$ where $n_{\mathsf{plk}} \in \mathbb{Z}_+$. Then, there is a circuit $\mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}$, parameterized by $\gamma, \delta \xleftarrow{\$} \mathbb{F}$, s.t., on inputs a PLONK structure plkst (corresponding to an instruction F) and $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{w}$ where $\mathbf{w}$ is some supporting witness proving $F(\mathbf{x}) = \mathbf{y}$ with PLONK's arithmetization, the following conditions must hold.

- If $F(\mathbf{x}) = \mathbf{y}$ and $\mathbf{w}$ is valid, then $\mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}(\mathsf{plkst}, \mathbf{x}, \mathbf{y}, \mathbf{w})$ always returns 1.
- Otherwise, $\mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}(\mathsf{plkst}, \mathbf{x}, \mathbf{y}, \mathbf{w})$ returns 1 with probability at most $\mathcal{O}(n_{\mathsf{plk}}/\,|\mathbb{F}|)$.

We choose PLONK's arithmetization and realize instructions as PLONK structures for the following purposes. First, PLONK structure of size $n_{\mathsf{plk}} = |\mathsf{plkst}|$ is sufficient to represent any arithmetic circuit of n_{gate} gates (including multiplications and additions) s.t. $n_{\mathsf{plk}} = \mathcal{O}(n_{\mathsf{gate}})$. Second, $\mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}$ also has the number of gates bounded by $\mathcal{O}(n_{\mathsf{plk}})$, making verification efficient compared to those for universal circuits [Val76]. Third, since plkst is viewed as input to $\mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}$, when applying ZKPs for verifying execution of $\mathsf{C}_{\mathsf{plk}}$, the privacy of plkst is guaranteed.

Therefore, for an N-step computation with instructions $(F_i)_{i\in[N]}$ selected from $\mathcal{F}'$, we have the corresponding $(\mathsf{plkst}_i)_{i\in[N]}$ s.t., for $\gamma, \delta \xleftarrow{\$} \mathbb{F}$, the correctness of the i-th step is reduced to $\mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}(\mathsf{plkst}_i, \mathsf{val}_{i-1}, (\mathsf{pc}_i \| \mathsf{macs}_i), \mathsf{auxplk}_i) = 1$ for $i \in [N]$ where $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{w}_i$ respectively are replaced by val_{i-1}, $(\mathsf{pc}_i \| \mathsf{macs}_i)$ and auxplk_i. This probabilistic test, for all $i \in [N]$, incurs a total soundness error $\mathcal{O}(N \cdot n_{\mathsf{plk}}/\,|\mathbb{F}|)$ by using union bound.

(iii) Correct Selections of Instructions. Finally, we need to ensure that $\{F_i\}_{i\in[N]} \subseteq \{F'_j\}_{j\in[T]}$. Recall that, for $i \in [N]$, $F_i = F'_{\mathsf{pc}_{i-1}}$. Moreover, the instruction set $\mathcal{F}' = \{F'_j\}_{j\in[T]}$ is described alternatively by the set $\{\mathsf{plkst}'_j\}_{j\in[T]}$. Therefore, we understand that the PLONK structure of the i-th instruction F_i is $\mathsf{plkst}_i := \mathsf{plkst}'_{\mathsf{pc}_{i-1}}$. Hence, to prove correct selections of instructions, we alternatively prove the tuple lookup $\{(\mathsf{pc}_{i-1} \| \mathsf{plkst}_i)\}_{i\in[N]} \subseteq \{(j \| \mathsf{plkst}'_j)\}_{j\in[T]}$ which is already discussed in Section 2.4.

Putting All Together. With the above discussion, the components for proving a RAM program contains, for each step $i \in [N]$,

$$\overline{\mathsf{pc}}_i,\ \overline{\mathsf{val}}_i,\ \mathsf{plkst}_i,\ \mathsf{auxplk}_i,\ \mathsf{pc}_i,\ \mathsf{macs}_i,\ \mathsf{macs}'_i \tag{7}$$

where $\overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}$, $\overline{\mathsf{val}}_i = \mathsf{val}_{i-1}$ belongs to macs_{i-1}, $\mathsf{macs}_i = (\ell_i, \mathsf{time}_i, \mathsf{val}_i, \mathsf{mop}_i)$ and $\mathsf{macs}'_i = (\ell'_i, \mathsf{time}'_i, \mathsf{val}'_i, \mathsf{mop}'_i)$. Arguably, we need

- $\overline{\mathsf{pc}}_i$ to determine plkst_i by selecting $\mathsf{plkst}'_{\mathsf{pc}_{i-1}}$ form $\mathcal{F}' = \{\mathsf{plkst}'_j\}_{j\in[T]}$;
- $\overline{\mathsf{val}}_i$ as input to F_i (corresponding to plkst_i) to compute pc_i and macs_i (with verification, supported by plkst_i, by circuit $\mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}$); and
- macs'_i for proving memory consistency of sequence $(\mathsf{macs}_i)_{i\in[N]}$ via $\mathsf{C}_{\mathsf{mem}}$.

These constraints will be summarized in (22). Thus, we can distribute them to the N instance-witness pairs $(\llbracket \mathbb{z}_{(i-1)i} \rrbracket)_{i\in[N]}$ of the form (5) that supports tuple permutation, tuple lookup arguments and also folding these pairs in parallel following Section 2.3. Eventually, we can construct a parallelizable and generic CFS that can be transformed into a ZKAoK for RAM programs, dubbed RAMenPaSTA. See Section 6 for a detailed discussion.

3 Preliminaries

Notations. We denote by $\mathbb{Z}$ and $\mathbb{Z}_+$ as the sets of integers and positive integers, respectively. For $a, b \in \mathbb{Z}$ s.t. $a \leq b$, we write $[a, b]$ and $[a]$ to indicate $\{a, a+1, \ldots, b\}$ and $\{1, \ldots, a\}$ (applicable when $a \geq 1$), respectively. Let $\mathbb{F}$ be a finite field. All vectors in this paper are column vectors. For two vectors $\mathbf{a}, \mathbf{b}$, we denote $(\mathbf{a}\|\mathbf{b})$ to be the vector $[\mathbf{a}^\top|\mathbf{b}^\top]^\top$. We use $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-n})$ for all $n \in \mathbb{N}$. We say an algorithm is probabilistic polynomial time (PPT) if this algorithm runs within polynomial time in the size of its inputs. For any relation $\mathcal{R}$ in this paper, if there exist two parties, namely, prover and verifier, jointly execute an interactive proof/argument for prover's knowledge of some witness wit for statement st and relation $\mathcal{R}$, then we may write $(\mathsf{st};\ \mathsf{wit}) \in \mathcal{R}$ where ";" is to separate the common input st and prover's input wit.

3.1 Commitment Scheme

A commitment scheme $\mathcal{C}$ allows one to commit to a secret vector (e.g., $\mathbf{c}$) into a commitment (e.g., $\tilde{\mathsf{c}}$) by a key ck and some additional randomness $\hat{\mathsf{c}}$. We define the relation $\mathcal{R}_{\mathsf{com}}$ s.t. $(\mathsf{ck}, \tilde{\mathsf{c}};\ \mathbf{c}, \hat{\mathsf{c}}) \in \mathcal{R}_{\mathsf{com}}$ iff $\tilde{\mathsf{c}}$ is computed from $\mathbf{c}$ with commitment key ck and randomness $\hat{\mathsf{c}}$. Regarding security, $\mathcal{C}$ should satisfy *binding*, i.e., when obtaining $\tilde{\mathsf{c}}$ from committing to $\mathbf{c}$ with randomness $\hat{\mathsf{c}}$, it is hard to find a distinct $\mathbf{c}' \neq \mathbf{c}$ and $\hat{\mathsf{c}}$ s.t. $(\mathsf{ck}, \tilde{\mathsf{c}};\ \mathbf{c}', \hat{\mathsf{c}}') \in \mathcal{R}_{\mathsf{com}}$, and *hiding*, i.e., $\tilde{\mathsf{c}}$ reveals nothing about $\mathbf{c}$. A formal definition of commitment scheme is deferred to Appendix B.6.

Remark 4. For ease of writing, we define the protocol Π_{com} in (8) for the prover to commit to $\mathbf{c}$, with randomness $\hat{\mathsf{c}}$ is sampled implicitly, and send commitment $\tilde{\mathsf{c}}$ to verifier so that both parties achieve $[\![\mathbf{c}]\!]_{\mathsf{ck}}$. Hence, when denoting $[\![\mathbf{c}]\!]_{\mathsf{ck}}$, we understand that prover keeps ck, $\mathbf{c}$, $\hat{\mathsf{c}}$ and $\tilde{\mathsf{c}}$ while verifier keeps ck and $\tilde{\mathsf{c}}$.

$$\Pi_{\mathsf{com}}(\mathsf{ck};\ \mathbf{c}) \to [\![\mathbf{c}]\!]_{\mathsf{ck}} \tag{8}$$

Remark 5. Following Remark 4, we may write a tuple mixing commitments and public values, e.g., $(a, [\![\mathbf{b}]\!]_{\mathsf{ck}_1}, c, [\![\mathbf{d}]\!]_{\mathsf{ck}_2})$ for some commitment keys $\mathsf{ck}_1, \mathsf{ck}_2$. Here, we understand that the verifier knows a, c and commitments to $\mathbf{b}, \mathbf{d}$ while the prover knows everything, including $\mathbf{b}, \mathbf{d}$ and commitment randomness. In some case, we may write $(a, [\![\mathbf{b}]\!]_{\mathsf{ck}_1}, c, [\![\mathbf{d}]\!]_{\mathsf{ck}_2};\ \mathbf{e})$ to imply that prover additionally has a secret $\mathbf{e}$ since $\mathbf{e}$ is after ";" according to notations in the beginning of Section 3.

Remark 6 (Homomorphism). In this result, the commitment scheme is assumed to be homomorphic. Specifically, we can compute $[\![\alpha_0 \cdot \mathbf{c}_0 + \alpha_1 \cdot \mathbf{c}_1]\!]_{\mathsf{ck}}$ from $\alpha_0, \alpha_1 \in \mathbb{F}$ and $[\![\mathbf{c}_0]\!]_{\mathsf{ck}}, [\![\mathbf{c}_1]\!]_{\mathsf{ck}}$ by computing $\alpha_0 \cdot [\![\mathbf{c}_0]\!] + \alpha_1 \cdot [\![\mathbf{c}_1]\!]$ for $\alpha_0, \alpha \in \mathbb{F}$.

Remark 7. We always commit to public vectors with default randomness 0, e.g., $[\![\mathbf{0}]\!]_{\mathsf{ck}} = (\mathsf{ck}, \mathsf{C.Commit}(\mathsf{ck}, \mathbf{0}, 0);\ \mathbf{0}, 0)$ for any ck where $\mathbf{0}$ is a public zero vector.

3.2 Interactive Folding Protocol for Folding Relaxed R1CS Instance-Witness Pairs

We recall the notion of relaxed R1CS (rR1CS) and its corresponding folding scheme (recalled in Appendix B.8), as an interactive protocol, from [KST22].

rR1CS. Let $d \in \mathbb{Z}_+$ and $m_{\mathsf{pub}}, m_1, \ldots, m_d, n \in \mathbb{Z}_+$, $m = m_{\mathsf{pub}} + \sum_{i \in [d]} m_i$. Let

$$\mathbb{x} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, \mathbb{x}.\mathbf{z}_1, \ldots, \mathbb{x}.\mathbf{z}_d, \mathbb{x}.\mathbf{e}) \in \mathbb{F} \times \mathbb{F}^{m_{\mathsf{pub}}} \times \mathbb{F}^{m_1} \times \cdots \times \mathbb{F}^{m_d} \times \mathbb{F}^n.$$

Let $\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_d, \mathsf{cke})$ be a tuple of commitment keys. Then, we write

$$[\![\mathbb{x}]\!]_{\mathsf{tck}} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, [\![\mathbb{x}.\mathbf{z}_1]\!]_{\mathsf{ck}_1}, \ldots, [\![\mathbb{x}.\mathbf{z}_d]\!]_{\mathsf{ck}_d}, [\![\mathbb{x}.\mathbf{e}]\!]_{\mathsf{cke}}) \tag{9}$$

to indicate instance-witness pair after committing to $\mathbb{x}.\mathbf{z}_1, \ldots, \mathbb{x}.\mathbf{z}_d$ and $\mathbb{x}.\mathbf{e}$ by $\mathsf{ck}_1, \ldots, \mathsf{ck}_d$ and cke, respectively. See Remark 5 for the use of this notation.

Let $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C}) \in (\mathbb{F}^{n \times m})^3$ be a tuple of three matrices, together called rR1CS structure. Then, $[\![\mathbb{x}]\!]_{\mathsf{tck}}$ is a valid rR1CS pair if it satisfies the relation

$$\mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}} = \left\{ [\![\mathbb{x}]\!]_{\mathsf{tck}} \;\middle|\; \begin{array}{l} \mathbb{x}.u \in \mathbb{F} \wedge \mathbb{x}.\mathsf{pub} \in \mathbb{F}^{m_{\mathsf{pub}}} \wedge (\mathbb{x}.\mathbf{z}_i \in \mathbb{F}^{m_i}\ \forall i \in [d]) \wedge \mathbb{x}.\mathbf{e} \in \mathbb{F}^n \\ \wedge [\![\mathbb{x}.\mathbf{z}_i]\!]_{\mathsf{ck}_i} \in \mathcal{R}_{\mathsf{com}}\ \forall i \in [d] \wedge [\![\mathbb{x}.\mathbf{e}]\!]_{\mathsf{cke}} \in \mathcal{R}_{\mathsf{com}} \\ \wedge \mathbf{A} \cdot \mathbf{z}' \circ \mathbf{B} \cdot \mathbf{z}' = \mathbb{x}.u \cdot \mathbf{C} \cdot \mathbf{z}' + \mathbb{x}.\mathbf{e} \end{array} \right\} \tag{10}$$

$\underline{\Pi_{\mathsf{rr1cs}}(\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_d, \mathsf{cke}), \mathfrak{S}, [\![\mathbb{x}_0]\!]_{\mathsf{tck}}, [\![\mathbb{x}_1]\!]_{\mathsf{tck}}) \to [\![\mathbb{x}]\!]_{\mathsf{tck}}}$

$\mathsf{ck}_1, \ldots, \mathsf{ck}_d$ and cke are commitment keys of a commitment scheme $\mathcal{C}$. Parse

$$[\![\mathbb{x}_i]\!]_{\mathsf{tck}} = (\mathbb{x}_i.u, \mathbb{x}_i.\mathsf{pub}, [\![\mathbb{x}_i.\mathbf{z}_1]\!]_{\mathsf{ck}_1}, \ldots, [\![\mathbb{x}_i.\mathbf{z}_d]\!]_{\mathsf{ck}_d}, [\![\mathbb{x}_i.\mathbf{e}]\!]_{\mathsf{cke}}) \ \forall i \in \{0,1\}$$

following the form (9). Protocol Π_{rr1cs} works as follows.

1. Prover: $\mathbf{g} \leftarrow \mathsf{garb}(\mathfrak{S}, [\![\mathbb{x}_0]\!]_{\mathsf{tck}}, [\![\mathbb{x}_1]\!]_{\mathsf{tck}})$.
2. Both parties run $[\![\mathbf{g}]\!]_{\mathsf{cke}} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke};\ \mathbf{g})$ where Π_{com} is defined in (8).
3. Verifier: $\alpha \xleftarrow{\$} \mathbb{F}$ and send α to prover.
4. This step folds $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$ and $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$ with α. Specifically, both parties compute
$$\mathbb{x}.u := \mathbb{x}_0.u + \alpha \cdot \mathbb{x}_1.u, \qquad [\![\mathbb{x}.\mathbf{z}_i]\!]_{\mathsf{ck}_i} := [\![\mathbb{x}_0.\mathbf{z}_i]\!]_{\mathsf{ck}_i} + \alpha \cdot [\![\mathbb{x}_1.\mathbf{z}_i]\!]_{\mathsf{ck}_i} \ \forall i \in [d],$$
$$\mathbb{x}.\mathsf{pub} := \mathbb{x}_0.\mathsf{pub} + \alpha \cdot \mathbb{x}_1.\mathsf{pub}, \quad [\![\mathbb{x}.\mathbf{e}]\!]_{\mathsf{cke}} := [\![\mathbb{x}_0.\mathbf{e}]\!]_{\mathsf{cke}} + \alpha \cdot [\![\mathbf{g}]\!]_{\mathsf{cke}} + \alpha^2 \cdot [\![\mathbb{x}_1.\mathbf{e}]\!]_{\mathsf{cke}}.$$

The output of this protocol is $[\![\mathbb{x}]\!]_{\mathsf{tck}} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, [\![\mathbb{x}.\mathbf{z}_1]\!]_{\mathsf{ck}_1}, \ldots, [\![\mathbb{x}.\mathbf{z}_d]\!]_{\mathsf{ck}_d}, [\![\mathbb{x}.\mathbf{e}]\!]_{\mathsf{cke}})$.

Fig. 3. Protocol Π_{rr1cs}.

where $\mathbf{z}' = (\mathbb{x}.\mathsf{pub} \| \mathbb{x}.\mathbf{z}_1 \| \ldots \| \mathbb{x}.\mathbf{z}_d)$ and $\circ$ is the entry-wise multiplication.

Folding rR1CS Instance-Witness Pairs. We recall from [KST22] for folding two instance-witness pairs into a single satisfying pair w.r.t. $\mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ in (10) (see Appendix B.8 for folding scheme's definition). Let $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}, [\![\mathbb{x}_1]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$. We show how to fold $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$ and $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$ to obtain the new pair $[\![\mathbb{x}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$. With random α, the idea is to compute $\mathbb{x}.u := \mathbb{x}_0.u + \alpha \cdot \mathbb{x}_1.u$, $\mathbb{x}.\mathsf{pub} := \mathbb{x}_0.\mathsf{pub} + \alpha \cdot \mathbb{x}_1.\mathsf{pub}$, $\mathbb{x}.\mathbf{z}_i := \mathbb{x}_0.\mathbf{z}_i + \alpha \cdot \mathbb{x}_1.\mathbf{z}_i \ \forall i \in [d]$. Let $\mathbf{z}'_i := (\mathbb{x}_i.\mathsf{pub} \| \mathbb{x}_i.\mathbf{z}_1 \| \ldots \| \mathbb{x}_i.\mathbf{z}_d)$, $\forall i \in \{0,1\}$, and $\mathbf{z}' := (\mathbb{x}.\mathsf{pub} \| \mathbb{x}.\mathbf{z}_1 \| \ldots \| \mathbb{x}.\mathbf{z}_d)$. Define

$$\mathsf{garb}(\mathfrak{S}, [\![\mathbb{x}_0]\!]_{\mathsf{tck}}, [\![\mathbb{x}_1]\!]_{\mathsf{tck}}) = \mathbf{A} \cdot \mathbf{z}'_0 \circ \mathbf{B} \cdot \mathbf{z}'_1 + \mathbf{A} \cdot \mathbf{z}'_1 \circ \mathbf{B} \cdot \mathbf{z}'_0 - \mathbf{C} \cdot (\mathbb{x}_1.u \cdot \mathbf{z}'_0 + \mathbb{x}_0.u \cdot \mathbf{z}'_1).$$

Then, one can check that

$$\mathbf{A} \cdot \mathbf{z}' \circ \mathbf{B} \cdot \mathbf{z}' = \mathbb{x}.u \cdot \mathbf{C} \cdot \mathbf{z}' + \mathbb{x}.\mathbf{e} \tag{11}$$

where $\mathbb{x}.\mathbf{e} := \mathbb{x}_0.\mathbf{e} + \alpha \cdot \mathsf{garb}(\mathfrak{S}, [\![\mathbb{x}_0]\!]_{\mathsf{tck}}, [\![\mathbb{x}_1]\!]_{\mathsf{tck}}) + \alpha^2 \cdot \mathbb{x}_1.\mathbf{e}$. From (11), we obtain a new pair $[\![\mathbb{x}]\!]_{\mathsf{tck}}$ satisfying $[\![\mathbb{x}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$. Hence, we present protocol Π_{rr1cs} (in Figure 3) for folding two pairs with extractability discussed in Lemma 1.

Lemma 1 ($(3; |\mathbb{F}|)$-Special Soundness of Π_{rr1cs}). *Let $\mathcal{C}$ be a homomorphic commitment scheme. Assume that Π_{rr1cs} in Figure 3 are rewinded thrice (from step 4), with the same $[\![\mathbf{g}]\!]_{\mathsf{cke}}$ and distinct $\{\alpha^{(i)}\}_{i\in[3]}$, to produce $\{[\![\mathbb{x}^{(i)}]\!]\}_{i\in[3]}$, respectively. If we have the valid witnesses s.t. $[\![\mathbb{x}^{(i)}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ $\forall i \in [3]$, then we can extract witnesses to construct $([\![\mathbb{x}_i]\!]_{\mathsf{tck}})_{i\in\{0,1\}}$ s.t. $[\![\mathbb{x}_i]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ $\forall i \in \{0,1\}$.*

The proof of Lemma 1 will be presented in Appendix B.9.

Efficiency. We first define the following notations in Definition 2.

Definition 2 (Notations for Efficiency). *We define the following notations.*

$\mathsf{c}(k)$**:** *the size of the commitment to a message of length k.*

$\mathsf{tp}^{\mathsf{c}}(k), \mathsf{tp}^{\mathsf{h}}(k)$**:** *the running times for the prover respectively to commit and to process homomorphism of the commitments to messages of length k.*

$\mathsf{tv}^{\mathsf{c}}(k), \mathsf{tv}^{\mathsf{h}}(k)$**:** *the running times for the verifier respectively to commit and to process homomorphism of the commitments to messages of length k.*

Assuming that $\mathbf{A}, \mathbf{B}, \mathbf{C}$ have $\mathcal{O}(n)$ non-zero entries, the efficiency related to protocol Π_{rr1cs} in Figure 3 as below. See Appendix B.9 for more details.

- *Size of* $[\![\mathbb{x}]\!]_{\mathsf{tck}}$. $\mathcal{O}(m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{c}(m_i) + \mathsf{c}(n))$.
- *Communication Cost of* Π_{rr1cs}. $\mathsf{c}(n)$.
- *Prover Time of* Π_{rr1cs}. $\mathcal{O}(n + \mathsf{tp}^{\mathsf{c}}(n) + m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tp}^{\mathsf{h}}(m_i) + \mathsf{tp}^{\mathsf{h}}(n))$.
- *Verifier Time of* Π_{rr1cs}. $\mathcal{O}(\mathsf{tv}^{\mathsf{c}}(n) + m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tv}^{\mathsf{h}}(m_i) + \mathsf{tv}^{\mathsf{h}}(n))$.

4 Conditional Folding Scheme

We formally present the definition of Conditional Folding Scheme (CFS), whose idea was discussed in Section 2.1. For a relation $\mathcal{R}$, CFS allows to fold two instance-witness pairs into a new pair s.t. the folded pair satisfying $\mathcal{R}$ implies that (i) the two original pairs also satisfy $\mathcal{R}$, and (ii) together satisfy another condition relation, denoted by $\mathcal{R}_{\mathsf{cond}}$. Section 4.1 is the syntax of a conditional folding scheme. Section 4.2 presents its correctness and security properties.

4.1 Syntax

Definition 3 (Syntax of CFS). *Let $\mathcal{PP}$, $\mathcal{I}$, and $\mathcal{Z}$ respectively be the sets of public parameters, instances, and witnesses. Let $\mathcal{P}_{\mathsf{aux}}$ and $\mathcal{W}_{\mathsf{aux}}$ be the set of auxiliary public inputs and witnesses supporting conditions for folding. Let $\mathcal{R} \subseteq \mathcal{PP} \times \mathcal{I} \times \mathcal{Z}$ and $\mathcal{R}_{\mathsf{cond}} \subseteq \mathcal{PP} \times \mathcal{I} \times \mathcal{I} \times \mathcal{P}_{\mathsf{aux}} \times \mathcal{Z} \times \mathcal{Z} \times \mathcal{W}_{\mathsf{aux}}$ be relations. A CFS $\mathcal{CF}$ for relation $\mathcal{R}$ and associated condition relation $\mathcal{R}_{\mathsf{cond}}$, is a tuple $\mathcal{CF}[\mathcal{R}, \mathcal{R}_{\mathsf{cond}}] = (\mathsf{CF.Setup}, \mathsf{CF.Fold}, \mathsf{CF.Prove})$ described as follows.*

$\mathsf{CF.Setup}(1^\lambda) \to \mathsf{pp}$**:** *This PPT algorithm returns a public parameter* pp.

$\mathsf{CF.Fold}(\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W) \to (I;\ Z)$**:** *This is an interactive protocol between prover, holding $(I_0, Z_0), (I_1, Z_1) \in \mathcal{I} \times \mathcal{Z}$ and auxiliary public input $P \in \mathcal{P}_{\mathsf{aux}}$ and witness $W \in \mathcal{W}_{\mathsf{aux}}$ supporting the condition for folding, and verifier, holding $I_0, I_1 \in \mathcal{I}$ and $P \in \mathcal{P}_{\mathsf{aux}}$. In the end, the prover and verifier receive the folded pair (I, Z) and folded instance $I \in \mathcal{I}$, respectively.*

$\mathsf{CF.Prove}(\mathsf{pp}, I;\ Z) \to \{0,1\}$**:** *This is an interactive protocol between prover, holding $(I, Z) \in \mathcal{I} \times \mathcal{Z}$, and verifier, holding $I \in \mathcal{I}$, s.t. prover tries to convince verifier that he knows $Z \in \mathcal{Z}$ satisfying $(\mathsf{pp}, I;\ Z) \in \mathcal{R}$. Eventually, verifier returns $b \in \{0,1\}$ for deciding whether to accept ($b = 1$) or reject ($b = 0$).*

4.2 Security Requirements

We now define the security properties including correctness, knowledge soundness and honest-verifier zero knowledge (HVZK) in Definitions 4, 5, and 7, respectively, for a CFS $\mathcal{CF}[\mathcal{R}, \mathcal{R}_{\mathsf{cond}}]$, whose syntax is defined in Definition 3.

Correctness. Correctness of a CFS is formally defined in Definition 4.

Definition 4 (Correctness). *For $\mathsf{pp} \leftarrow \mathsf{CF.Setup}(1^\lambda)$, any $\{(\mathsf{pp}, I_i;\ Z_i)\}_{i \in \{0,1\}} \subseteq \mathcal{R}$, $P \in \mathcal{P}_{\mathsf{aux}}$ and $W \in \mathcal{W}_{\mathsf{aux}}$ s.t. $(\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W) \in \mathcal{R}_{\mathsf{cond}}$, it holds that*

$$\Pr\left[(\mathsf{pp}, I;\ Z) \in \mathcal{R} \middle| (I;\ Z) \leftarrow \mathsf{CF.Fold}(\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W)\right] \geq 1 - \epsilon_1.$$

And, for any $(\mathsf{pp}, I;\ Z) \in \mathcal{R}$, $\Pr\left[b = 1 \middle| b \leftarrow \mathsf{CF.Prove}(\mathsf{pp}, I;\ Z)\right] \geq 1 - \epsilon_2$ where ϵ_1 and ϵ_2 are respectively folding and completeness errors (of $\mathsf{CF.Prove}$). The sum $\epsilon_1 + \epsilon_2$ is the correctness error of $\mathcal{CF}$.

Knowledge Soundness. We now briefly discuss the intuition for modeling knowledge soundness property. To model, we use a PPT extractor $\mathcal{E}$, having oracle access to the potential adversary playing the role of a prover, to extract the witnesses given the instances I_0 and I_1. Here, we allow $\mathcal{E}$ to rewind $\mathcal{A}$ to any previous state of $\mathcal{A}$, several times polynomial in λ. Therefore, we write $\mathcal{E}^{\mathcal{A}(\mathsf{sec})}(\mathsf{pp}, I_0, I_1, P)$ to indicate the execution of $\mathcal{E}$ (playing the role of verifier) with oracle access to $\mathcal{A}$ (playing the role of prover) to extract the witness Z_i corresponding to I_i, w.r.t. $\mathcal{R}$ for $i \in \{0,1\}$, and witness W for guaranteeing the condition for folding I_0 and I_1. This notation implies the executions of $\mathsf{CF.Fold}$ and $\mathsf{CF.Prove}$ inside. That is, $\mathcal{E}$ can invoke $\mathsf{CF.Fold}$ and $\mathsf{CF.Prove}$ appropriately such that $\mathcal{A}$ cannot distinguish $\mathcal{E}$ from a verifier.

If there is any invocation to $\mathsf{CF.Prove}$ that returns 0, $\mathcal{E}$ simply aborts and return $\perp$. Since $\mathcal{A}$ is trying to break knowledge soundness, we consider that $\mathsf{CF.Prove}$ always returns 1 because returning 0 indicates that $\mathcal{A}$ does not know the corresponding witnesses for I_i to satisfy $\mathcal{R}$, $\forall i \in \{0,1\}$, with overwhelming probability, by the correctness in Definition 4.

Eventually, if $\mathcal{E}$ returns (Z_0, Z_1, W) as the extracted witnesses, the knowledge soundness of $\mathcal{CF}$ is guaranteed if, with overwhelming probability, (i) Z_0 and Z_1 are valid witnesses for I_0 and I_1, respectively, w.r.t. $\mathcal{R}$, and, (ii) with witness W, the condition must hold, i.e., $(\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W) \in \mathcal{R}_{\mathsf{cond}}$.

The above discussion is formalized in $\mathsf{E}_{\mathcal{A}}^{\mathsf{cf\text{-}sound}}(\lambda)$ (c.f. Figure 4). Knowledge soundness is formally defined in Definition 5.

$\mathsf{pp} \leftarrow \mathsf{CF.Setup}(1^\lambda)$; $(I_0, I_1, P, \mathsf{sec}) \leftarrow \mathcal{A}(\mathsf{pp})$; $(Z_0, Z_1, W) \leftarrow \mathcal{E}^{\mathcal{A}(\mathsf{sec})}(\mathsf{pp}, I_0, I_1, P)$.
$b := ((\mathsf{pp}, I_0;\ Z_0) \in \mathcal{R}) \wedge ((\mathsf{pp}, I_1;\ Z_1) \in \mathcal{R}) \wedge ((\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W) \in \mathcal{R}_{\mathsf{cond}})$.
Return $\neg b$.

Fig. 4. Experiment $\mathsf{E}_{\mathcal{A}}^{\mathsf{cf\text{-}sound}}(\lambda)$.

Definition 5 (Knowledge Soundness). *$\mathcal{CF}$ is said to satisfy knowledge soundness if $N \in \mathbb{Z}_+$, any (PPT) $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$, with rewindable oracle access to $\mathcal{A}$, it holds that $\Pr[\mathsf{E}_{\mathcal{A}}^{\mathit{cf\text{-}sound}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ where $\mathsf{E}_{\mathcal{A}}^{\mathit{cf\text{-}sound}}(\lambda)$.*

HVZK. Before formally defining HVZK of $\mathcal{CF}$, we define the transcript the Definition 6. Then, we define HVZK of $\mathcal{CF}$ in Definition 7.

Definition 6. *We denote by $\Pi_{\mathcal{CF}}(\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W)$ to be the combined protocol that sequentially runs $(I;\ Z) \leftarrow \mathsf{CF.Fold}(\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W)$ and then $b \leftarrow \mathsf{CF.Prove}(\mathsf{pp}, I;\ Z)$. We define the transcript*

$$\mathsf{tr} \leftarrow \mathsf{View}(\Pi_{\mathcal{CF}}(\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W)) \tag{12}$$

of $\Pi_{\mathcal{CF}}$ between prover and verifier to contain all public inputs and exchanged messages during executing protocols $\mathsf{CF.Fold}$ and $\mathsf{CF.Prove}$ in $\Pi_{\mathcal{CF}}$.

Definition 7 (HVZK). *$\mathcal{CF}$ is HVZK if, for $\mathsf{pp} \leftarrow \mathsf{CF.Setup}(1^\lambda)$, there exists a PPT simulator $\mathcal{S}$ s.t., for any distinguisher $\mathcal{A}$, any $(I_0, I_1, P;\ Z_0, Z_1, W)$ satisfying $(\mathsf{pp}, I_i; Z_i) \in \mathcal{R}\ \forall i \in \{0, 1\}$ and $((\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W) \in \mathcal{R}_{\mathsf{cond}})$,*

$$\begin{aligned}\Big|\Pr\left[\mathcal{A}(\mathsf{tr}) = 1 \middle| \mathsf{tr} \leftarrow \mathsf{View}(\Pi_{\mathcal{CF}}(\mathsf{pp}, I_0, I_1, P;\ Z_0, Z_1, W))\right] \\ - \Pr\left[\mathcal{A}(\mathsf{tr}) = 1 \middle| \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{pp}, I_0, I_1, P)\right]\Big| \leq \mathsf{negl}(\lambda).\end{aligned}$$

5 A Generic CFS

We propose a generic CFS $\mathcal{CF}_{\mathsf{gnr}}$, following the discussion in Sections 2.2 and 2.4. Section 5.1 presents the design of the instance-witness pairs for CFS following the form discussed in (5). In Section 5.2, we define the relations for the pairs defined in Section 5.1. In Section 5.3, we construct protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ for folding two instance-witness pairs specified in Section 5.1 w.r.t. relations in Section 5.2.

5.1 Form of Instance-Witness Pairs for Generic CFS

Following Section 2.4, we come up an instance-witness pair of the form $[\![\mathbb{z}]\!] = ([\![\mathbb{x}]\!], [\![\mathsf{front}]\!], [\![\mathsf{rear}]\!], [\![\mathbb{x}^\star]\!], [\![\mathbf{s}]\!], [\![\mathbf{a}]\!])$ in (5) without mentioning the commitment keys (for committing witnesses) and indices (for folding following HS). Recall that $[\![\mathbb{x}]\!]$ and $[\![\mathbb{x}^\star]\!]$ are rR1CS instance-witness pairs w.r.t. rR1CS structures $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C})$ and $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$, respectively. In our design, the witness of $[\![\mathbb{x}]\!]$ contains $d \in \mathbb{Z}_+$ vectors $\mathbb{x}.\mathbf{z}_1, \ldots, \mathbb{x}.\mathbf{z}_d$ respectively of lengths $m_1, \ldots, m_d$ and an error vector $\mathbb{x}.\mathbf{e}$ of length n. The witness of $[\![\mathbb{x}^\star]\!]$ contains 3 vectors $\mathbb{x}^\star.\mathbf{z}_1, \ldots, \mathbb{x}^\star.\mathbf{z}_3$ respectively of length m'_1, m'_2, m'_3, and an error vector $\mathbb{x}^\star.\mathbf{e}'$ of length n'. $\mathbb{x}^\star$ is of length 3 to capture the condition between $\mathbb{x}^\star.\mathbf{z}_1$ and $\mathbb{x}^\star.\mathbf{z}_2$ with supporting witness $\mathbb{x}^\star.\mathbf{z}_3$. Finally, $\mathbf{s}$ and $\mathbf{a}$ are of length $s \in \mathbb{Z}_+$.

Let $\mathcal{C}$ be a homomorphic commitment scheme defined in Section 3.1. Let $d \in \mathbb{Z}_+$. Let $\mathsf{ck}_1, \ldots, \mathsf{ck}_d$, cke, ck'_1, ck'_2, ck'_3, cke' and cks be commitment keys for committing messages over $\mathbb{F}$ of lengths $m_1, \ldots, m_d$, n, m'_1, m'_2, m'_3, n' and s, respectively. We define instance-witness pair $[\![\mathbb{z}]\!]_{\mathsf{pp}}$ with public parameter pp containing commitment keys generated from $\mathcal{C}$. The formal design is as follows.

$$\mathsf{pp} = (\mathsf{tck}, \quad \mathsf{tck}', \quad \mathsf{cks}), \tag{13}$$

$$\mathbb{z} = (\mathbb{x}, \quad \mathsf{front}, \quad \mathsf{rear}, \quad \mathbb{x}^\star, \quad \mathbf{s}, \quad \mathbf{a}), \tag{14}$$

$$[\![\mathbb{z}]\!]_{\mathsf{pp}} = ([\![\mathbb{x}]\!]_{\mathsf{tck}}, \quad [\![\mathsf{front}]\!]_{\mathsf{ck}'_2}, \quad [\![\mathsf{rear}]\!]_{\mathsf{ck}'_1}, \quad [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}, \quad [\![\mathbf{s}]\!]_{\mathsf{cks}}, \quad [\![\mathbf{a}]\!]_{\mathsf{cks}}) \tag{15}$$

where $\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_d, \mathsf{cke})$ and $\mathsf{tck}' = (\mathsf{ck}'_1, \ldots, \mathsf{ck}'_3, \mathsf{cke}')$ are tuple of keys for rR1CS instance-witness pairs (see Section 3.2); front, rear, $\mathbf{s}$, $\mathbf{a}$ are vectors over $\mathbb{F}$; and $\mathbb{x}, \mathbb{x}^\star, [\![\mathbb{x}]\!]_{\mathsf{tck}}, [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}$ have the following forms.

$$\begin{aligned}
\mathbb{x} &= (\mathbb{x}.u, \quad \mathbb{x}.\mathsf{pub}, \quad \mathbb{x}.\mathbf{z}_1, \quad \ldots, \quad \mathbb{x}.\mathbf{z}_d, \quad \mathbb{x}.\mathbf{e} \quad), \\
[\![\mathbb{x}]\!]_{\mathsf{tck}} &= (\mathbb{x}.u, \quad \mathbb{x}.\mathsf{pub}, \quad [\![\mathbb{x}.\mathbf{z}_1]\!]_{\mathsf{ck}_1}, \quad \ldots, \quad [\![\mathbb{x}.\mathbf{z}_d]\!]_{\mathsf{ck}_d}, \quad [\![\mathbb{x}.\mathbf{e}]\!]_{\mathsf{cke}} \quad), \qquad (16) \\
\mathbb{x}^\star &= (\mathbb{x}^\star.u, \quad \mathbb{x}^\star.\mathsf{pub}, \quad \mathbb{x}^\star.\mathbf{z}_1, \quad \ldots, \quad \mathbb{x}^\star.\mathbf{z}_3, \quad \mathbb{x}^\star.\mathbf{e} \quad), \\
[\![\mathbb{x}^\star]\!]_{\mathsf{tck}'} &= (\mathbb{x}^\star.u, \quad \mathbb{x}^\star.\mathsf{pub}, \quad [\![\mathbb{x}^\star.\mathbf{z}_1]\!]_{\mathsf{ck}'_1}, \quad \ldots, \quad [\![\mathbb{x}^\star.\mathbf{z}_3]\!]_{\mathsf{ck}'_3}, \quad [\![\mathbb{x}^\star.\mathbf{e}]\!]_{\mathsf{cke}'} \quad). \qquad (17)
\end{aligned}$$

It can be seen that $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{n \times m}$ and $\mathbf{A}', \mathbf{B}', \mathbf{C}' \in \mathbb{F}^{n' \times m'}$ where $m = m_{\mathsf{pub}} + \sum_{i \in [d]} m_i$ and $m' = m'_{\mathsf{pub}} + \sum_{i \in [3]} m'_i$ where $m_{\mathsf{pub}} = |\mathbb{x}.\mathsf{pub}|$ and $m'_{\mathsf{pub}} = |\mathbb{x}^\star.\mathsf{pub}|$.

5.2 Defining Relations

We define the relations for the instance-witness pairs in (15) to support our construction of CFS $\mathcal{CF}_{\mathsf{gnr}}$. Let pp be of the form (13). Let $\mathfrak{S}, \mathfrak{S}'$ be fixed in advance. We define the relations $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$, in (18), and $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$, in (19), respectively, s.t.

- $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ captures the constraints inside each $[\![\mathbb{z}]\!]_{\mathsf{pp}}$ of the form (15).
- $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ captures the condition when folding $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$ into $[\![\mathbb{z}]\!]_{\mathsf{pp}}$ where $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$, $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}]\!]_{\mathsf{pp}}$ are of the form (15).

$$\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} = \left\{ [\![\mathbb{z}]\!]_{\mathsf{pp}} \;\middle|\; \begin{array}{l} [\![\mathsf{front}]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}]\!]_{\mathsf{ck}'_1}, [\![\mathbf{s}]\!]_{\mathsf{cks}}, [\![\mathbf{a}]\!]_{\mathsf{cks}} \in \mathcal{R}_{\mathsf{com}} \\ \wedge [\![\mathbb{x}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}} \wedge [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}} \end{array} \right\}. \qquad (18)$$

We now define the condition relation $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ with associated set $\mathcal{W}_{\mathsf{gnr\text{-}aux}}$ defined to be the set containing all auxiliary witnesses in the forms of fixed-length vectors over $\mathbb{F}$ s.t. we can fold $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$ with auxiliary witness $\mathbf{w} \in \mathcal{W}_{\mathsf{gnr\text{-}aux}}$. Parse $[\![\mathbb{z}_i]\!]_{\mathsf{pp}} = ([\![\mathbb{x}_i]\!]_{\mathsf{tck}}, [\![\mathsf{front}_i]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}_i]\!]_{\mathsf{ck}'_1}, [\![\mathbb{x}_i^\star]\!]_{\mathsf{tck}'}, [\![\mathbf{s}_i]\!]_{\mathsf{cks}}, [\![\mathbf{a}_i]\!]_{\mathsf{cks}})$ as in (15) for $i \in \{0, 1\}$. Relation $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ is formally defined to be

$$\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}} = \left\{ (\mathbf{p}, [\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}};\ \mathbf{w}) \,\middle|\, \mathbf{w} \in \mathcal{W}_{\mathsf{gnr\text{-}aux}} \wedge \mathbf{A}' \cdot \mathbf{c} \circ \mathbf{B}' \cdot \mathbf{c} = \mathbf{C}' \cdot \mathbf{c} \right\} \qquad (19)$$

where $\mathbf{p}$ is an additional public input supporting the condition and vector $\mathbf{c} = (\mathbf{p} \| \mathsf{rear}_0 \| \mathsf{front}_1 \| \mathbf{w})$. Relation $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ specifies the condition in the form of a computation of R1CS equation w.r.t. matrices $\mathbf{A}', \mathbf{B}'$ and $\mathbf{C}'$ in $\mathfrak{S}'$.

5.3 Protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ and Construction of CFS $\mathcal{CF}_{\mathsf{gnr}}$

We formally describe the generic CFS $\mathcal{CF}_{\mathsf{gnr}} = (\mathsf{CF.Setup}, \mathsf{CF.Fold}, \mathsf{CF.Prove})$. To this end, first describe protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ in Figure 5 for folding $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$ into $[\![\mathbb{z}]\!]_{\mathsf{pp}}$ and then use $\Pi_{\mathsf{fold\text{-}gnr}}$ to construct $\mathsf{CF.Fold}$ in Figure 6.

Overview of $\Pi_{\mathsf{fold\text{-}gnr}}$. We describe in high level the protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ for folding $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$ into $[\![\mathbb{z}]\!]_{\mathsf{pp}}$, where $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$, $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}]\!]_{\mathsf{pp}}$ are of the form (15). This high-level description follows the discussion for folding two instance-witness pairs in Sections 2.2 and 2.4. The prover and verifier proceed as follows.

- Obtain folded rR1CS instance-witness pair $[\![\mathbb{x}]\!]$ by folding $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$ and $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$ using Nova's folding with a challenge $\alpha_1 \in \mathbb{F}$. This can be done by running protocol Π_{rr1cs} in Figure 3 w.r.t. tck and $\mathfrak{S}$.
- Set $[\![\mathsf{front}]\!]_{\mathsf{ck}'_2} := [\![\mathsf{front}_0]\!]_{\mathsf{ck}'_2}$ and $[\![\mathsf{rear}]\!]_{\mathsf{ck}'_1} := [\![\mathsf{rear}_1]\!]_{\mathsf{ck}'_1}$.
- Obtain the accumulated relaxed R1CS instance-witness pair $[\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}$ for the condition by first forming the rR1CS instance-witness pair $[\![\mathbb{y}]\!]_{\mathsf{tck}'}$ where $\mathbb{y}$ contains $[\![\mathsf{rear}_0]\!]_{\mathsf{ck}'_1}, [\![\mathsf{front}_1]\!]_{\mathsf{ck}'_2}$ and some auxiliary witness $[\![\mathbf{w}]\!]_{\mathsf{ck}'_3}$ s.t. $[\![\mathbb{y}]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}$. Then, fold the three pairs $[\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}$ $[\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'}$ and $[\![\mathbb{y}]\!]_{\mathsf{tck}'}$, into $[\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}$ by first running protocol Π_{rr1cs}, described in Figure 3, with challenge $\alpha_1 \in \mathbb{F}$ to fold $[\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}$ and $[\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'}$ into $[\![\mathbb{y}']\!]_{\mathsf{tck}'}$. Then, fold $[\![\mathbb{y}']\!]_{\mathsf{tck}'}$ and $[\![\mathbb{y}]\!]_{\mathsf{tck}'}$ into $[\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}$ by again running protocol Π_{rr1cs} with challenge $\alpha_2 \in \mathbb{F}$.

$\underline{\Pi_{\mathsf{fold\text{-}gnr}}(\mathsf{pp}, \mathfrak{S}, \mathfrak{S}', [\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}}, \mathbf{p};\ \mathbf{w}) \to [\![\mathbb{z}]\!]_{\mathsf{pp}}}$

Parse $[\![\mathbb{z}_i]\!]_{\mathsf{pp}} = ([\![\mathbb{x}_i]\!]_{\mathsf{tck}}, [\![\mathsf{front}_i]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}_i]\!]_{\mathsf{ck}'_1}, [\![\mathbb{x}_i^\star]\!]_{\mathsf{tck}'}, [\![\mathbf{s}_i]\!]_{\mathsf{cks}}, [\![\mathbf{a}_i]\!]_{\mathsf{cks}})\ \forall i \in \{0,1\}$ where public parameter pp is of the form (13). We describe protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ as follows.

1. Prover: Compute $\mathbf{g} \leftarrow \mathsf{garb}(\mathfrak{S}, [\![\mathbb{x}_0]\!]_{\mathsf{tck}}, [\![\mathbb{x}_1]\!]_{\mathsf{tck}})$ and $\mathbf{g}_1 \leftarrow \mathsf{garb}(\mathfrak{S}', [\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}, [\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'})$ where garb is in Section 3.2.
2. Both parties run $[\![\mathbf{g}]\!]_{\mathsf{cke}} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke};\ \mathbf{g})$, $[\![\mathbf{w}]\!]_{\mathsf{ck}'_3} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ck}'_3;\ \mathbf{w})$ and $[\![\mathbf{g}_1]\!]_{\mathsf{cke}'} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke}';\ \mathbf{g}_1)$ where Π_{com} is defined in (8). Form the pair (for the condition)
$$[\![\mathbb{y}]\!]_{\mathsf{tck}'} = (\mathbb{y}.u,\ \mathbb{y}.\mathsf{pub},\ [\![\mathbb{y}.\mathbf{z}_1]\!]_{\mathsf{ck}'_1},\ [\![\mathbb{y}.\mathbf{z}_2]\!]_{\mathsf{ck}'_2},\ [\![\mathbb{y}.\mathbf{z}_3]\!]_{\mathsf{ck}'_3},\ [\![\mathbb{y}.\mathbf{e}]\!]_{\mathsf{cke}'}\ )$$
$$:= (1,\ \mathbf{p},\ [\![\mathsf{rear}_0]\!]_{\mathsf{ck}'_1},\ [\![\mathsf{front}_1]\!]_{\mathsf{ck}'_2},\ [\![\mathbf{w}]\!]_{\mathsf{ck}'_3},\ [\![\mathbf{0}^{n'}]\!]_{\mathsf{cke}'}\ ).$$
Notice that $\mathsf{ck}'_1 = \mathsf{ck}_2$ and $\mathsf{ck}'_2 = \mathsf{ck}_1$ as designed in Section 5.1 and the commitment $[\![\mathbf{0}^{n'}]\!]_{\mathsf{cke}'}$ can be determined by verifier alone, as in Remark 7.
3. Verifier: Sample $\alpha_1 \xleftarrow{\$} \mathbb{F}$ and send α_1 to prover.
4. Both parties run step 4 in Π_{rr1cs} (see Figure 3) to
   - fold $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$, $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$ into $[\![\mathbb{x}]\!]_{\mathsf{tck}}$ with α_1 and $[\![\mathbf{g}]\!]_{\mathsf{cke}}$, and
   - fold $[\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}$, $[\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'}$ into $[\![\mathbb{y}']\!]_{\mathsf{tck}'}$ with α_1 and $[\![\mathbf{g}_1]\!]_{\mathsf{cke}'}$.
5. Prover: Compute $\mathbf{g}_2 \leftarrow \mathsf{garb}(\mathfrak{S}', [\![\mathbb{y}']\!]_{\mathsf{tck}'}, [\![\mathbb{y}]\!]_{\mathsf{tck}'})$.
6. Both parties run $[\![\mathbf{g}_2]\!]_{\mathsf{cke}'} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke}';\ \mathbf{g}_2)$.
7. Verifier: Sample $\alpha_2 \xleftarrow{\$} \mathbb{F}$ and send α_2 to prover.
8. Both parties run step 4 in Π_{rr1cs} to fold $[\![\mathbb{y}']\!]_{\mathsf{tck}'}$, $[\![\mathbb{y}]\!]_{\mathsf{tck}'}$ into $[\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}$ with $\alpha_2, [\![\mathbf{g}_2]\!]_{\mathsf{cke}'}$.
9. Finally, both parties compute the followings.
$$[\![\mathsf{front}]\!]_{\mathsf{ck}'_2} := [\![\mathsf{front}_0]\!]_{\mathsf{ck}'_2}, \quad [\![\mathbf{s}]\!]_{\mathsf{cks}} := [\![\mathbf{s}_0]\!]_{\mathsf{cks}} + [\![\mathbf{s}_1]\!]_{\mathsf{cks}},$$
$$[\![\mathsf{rear}]\!]_{\mathsf{ck}'_1} := [\![\mathsf{rear}_1]\!]_{\mathsf{ck}_2}, \quad [\![\mathbf{a}]\!]_{\mathsf{cks}} := [\![\mathbf{a}_0]\!]_{\mathsf{cks}} + \alpha_1 \cdot [\![\mathbf{a}_1]\!]_{\mathsf{cks}} + \alpha_1^2 \cdot ([\![\mathbf{s}_0]\!]_{\mathsf{cks}} - [\![\mathbf{s}_1]\!]_{\mathsf{cks}}).$$

The output of this protocol is $[\![\mathbb{z}]\!]_{\mathsf{pp}} = ([\![\mathbb{x}]\!]_{\mathsf{tck}}, [\![\mathsf{front}]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}]\!]_{\mathsf{ck}'_1}, [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}, [\![\mathbf{s}]\!]_{\mathsf{cks}}, [\![\mathbf{a}]\!]_{\mathsf{cks}})$.

Fig. 5. Protocol $\Pi_{\mathsf{fold\text{-}gnr}}$.

Generic CFS $\mathcal{CF}_{\mathsf{gnr}}$

$\mathsf{CF.Setup}(1^\lambda) \to \mathsf{pp}$**:** This algorithm works as follows.
1. Sample $\mathsf{ck}_1, \ldots, \mathsf{ck}_d, \mathsf{cke}, \mathsf{ck}'_1, \mathsf{ck}'_2, \mathsf{ck}'_3, \mathsf{cke}', \mathsf{cks}$ from $\mathsf{C.Setup}$.
2. Form tuples $\mathsf{tck} := (\mathsf{ck}_1, \ldots, \mathsf{ck}_d, \mathsf{cke})$ and $\mathsf{tck}' := (\mathsf{ck}'_1, \mathsf{ck}'_2, \mathsf{ck}'_3, \mathsf{cke}')$.
3. Return $\mathsf{pp} := (\mathsf{tck}, \mathsf{tck}', \mathsf{cks})$.

$\mathsf{CF.Fold}(\mathsf{pp}, [\![\mathbb{z}_0]\!]_{\mathsf{pp}} [\![\mathbb{z}_1]\!]_{\mathsf{pp}}, \mathbf{p};\ \mathbf{w}) \to [\![\mathbb{z}]\!]_{\mathsf{pp}}$**:** Both parties run $\Pi_{\mathsf{fold\text{-}gnr}}$ (see Figure 5) by $[\![\mathbb{z}]\!]_{\mathsf{pp}} \leftarrow \Pi_{\mathsf{fold\text{-}gnr}}(\mathsf{pp}, \mathfrak{S}, \mathfrak{S}', [\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}}, \mathbf{p};\ \mathbf{w})$.

$\mathsf{CF.Prove}(\mathsf{pp}, [\![\mathbb{z}]\!]_{\mathsf{pp}}) \to \{0,1\}$**:** This is a proof/argument for $[\![\mathbb{z}]\!]_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$.

Fig. 6. Generic CFS $\mathcal{CF}_{\mathsf{gnr}}$.

- Finally, compute $[\![\mathbf{s}]\!]_{\mathsf{cks}} := [\![\mathbf{s}_0]\!]_{\mathsf{cks}} + [\![\mathbf{s}_1]\!]_{\mathsf{cks}}$ and $[\![\mathbf{a}]\!]_{\mathsf{cks}} := [\![\mathbf{a}_0]\!]_{\mathsf{cks}} + \alpha_1 \cdot [\![\mathbf{a}_1]\!]_{\mathsf{cks}} + \alpha_1^2 \cdot ([\![\mathbf{s}_0]\!]_{\mathsf{cks}} - [\![\mathbf{s}_1]\!]_{\mathsf{cks}})$ as instructed in Section 2.4.

Formal Description of $\Pi_{\mathsf{fold\text{-}gnr}}$. Assuming pp, $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C}) \in (\mathbb{F}^{n \times m})^3$, $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}') \in (\mathbb{F}^{n' \times m'})^3$ are commonly determined by prover and verifier. Protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ in Figure 5 is for folding $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$ into $[\![\mathbb{z}]\!]_{\mathsf{pp}}$, with public input $\mathbf{p} \in m'_{\mathsf{pub}}$ and secret vector $\mathbf{w} \in \mathcal{W}_{\mathsf{gnr\text{-}aux}}$ for the condition.

CFS $\mathcal{CF}_{\mathsf{gnr}}$. The construction of CFS $\mathcal{CF}_{\mathsf{gnr}}$ can be constructed from this protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ straightforwardly. Let $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C})$, $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$ be described as above. Let $\mathcal{C}$ be a homomorphic commitment scheme. We describe the generic CFS $\mathcal{CF}_{\mathsf{gnr}}[\mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}, \mathcal{R}_{\mathsf{gnr\text{-}cond}}^{\mathfrak{S}'}] = (\mathsf{CF.Setup}, \mathsf{CF.Fold}, \mathsf{CF.Prove})$ in Figure 6 (with security in Theorem 1) employing $\Pi_{\mathsf{fold\text{-}gnr}}$ as a building block. In the description, we do not strictly follow the syntax defined in Definition 3 due to abusing notations. However, recall from Section 3.1, it is understood that the witnesses Z_0, Z_1 of provers are $(\mathbb{z}_0, \hat{\mathbb{z}}_0), (\mathbb{z}_1, \hat{\mathbb{z}}_1)$, respectively, and the instances I_0, I_1 are $\tilde{\mathbb{z}}_0, \tilde{\mathbb{z}}_1$ which are contained in $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$, respectively. Here $\hat{\mathbb{z}}$ and $\tilde{\mathbb{z}}$ are the randomness and commitment of $\mathbb{z}$ respectively, as defined in Section 3.1.

Theorem 1 (Security of $\mathcal{CF}_{\mathsf{gnr}}$). *If* $\mathsf{CF.Prove}$ *is an (HV)ZKAoK and $\mathcal{C}$ is a secure homomorphic commitment scheme, then $\mathcal{CF}_{\mathsf{gnr}}$ is correct with correctness error* $\mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp})$*, HVZK and knowledge-sound with soundness error* $\mathcal{O}(1/|\mathbb{F}| + \mathsf{serr}_{\mathsf{prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda))$ *where* $\mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp})$ *and* $\mathsf{serr}_{\mathsf{prf}}(\mathsf{pp})$ *respectively are completeness and soundness error of* $\mathsf{CF.Prove}$*.*

Proof (Sketch). Completeness is straightforward. HVZK is implied from the hiding property of $\mathcal{C}$ and HVZK from the employed protocol realizing $\mathsf{CF.Prove}$. Knowledge soundness of $\mathcal{CF}_{\mathsf{gnr}}$ follows the knowledge soundness of $\mathsf{CF.Prove}$ and the binding of $\mathcal{C}$. The full security proof is presented in Appendix C.1. □

Efficiency. Recall the notations in Definition 2. The efficiency is as follows.

- *Size of Public Instance in* $[\![\mathsf{z}]\!]_{\mathsf{pp}}$ *in* (15). $\mathcal{O}(m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{c}(m_i) + \mathsf{c}(n) + m'_{\mathsf{pub}} + \sum_{i\in[3]} \mathsf{c}(m'_i) + \mathsf{c}(n') + \mathsf{c}(s))$.
- *Communication Cost of* $\mathsf{CF.Fold}$. $\mathcal{O}(\mathsf{c}(n) + \mathsf{c}(n'))$.
- *Prover Time of* $\mathsf{CF.Fold}$. $\mathcal{O}(n + \mathsf{tp}^{\mathsf{c}}(n) + m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tp}^{\mathsf{h}}(m_i) + \mathsf{tp}^{\mathsf{h}}(n) + n' + \mathsf{tp}^{\mathsf{c}}(n') + m'_{\mathsf{pub}} + \sum_{i\in[3]} \mathsf{tp}^{\mathsf{h}}(m'_i) + \mathsf{tp}^{\mathsf{h}}(n'))$.
- *Verifier Time of* $\mathsf{CF.Fold}$. $\mathcal{O}(\mathsf{tv}^{\mathsf{c}}(n) + m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tv}^{\mathsf{h}}(m_i) + \mathsf{tv}^{\mathsf{h}}(n) + \mathsf{tv}^{\mathsf{c}}(n') + m'_{\mathsf{pub}} + \sum_{i\in[3]} \mathsf{tv}^{\mathsf{h}}(m'_i) + \mathsf{tv}^{\mathsf{h}}(n'))$.
- *Prover and Verifier Time of* $\mathsf{CF.Prove}$. This depends on the employed ZKAoK.

See Appendix C.2 for a detailed discussion of efficiency.

6 RAMenPaSTA: Parallelizable Scalable Transparent Arguments of Knowledge for RAM Programs

We construct RAMenPaSTA, a parallelizable and generic argument of knowledge for RAM programs. Let pp be the tuple of commitment keys, described in (13). Recall that $\mathcal{F}' = \{F'_j\}_{j\in[T]}$ is an instruction set of T instructions describable by the PLONK structures $\mathcal{F}'_{\mathsf{struct}} = \{\mathsf{plkst}'_j\}_{j\in[T]}$. Given an output $\mathsf{val}_{\mathsf{out}}$ and commitment tuple $[\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}}$ to secret input $\mathsf{val}_{\mathsf{in}}$ with commitment key cki, prover proceeds an interactive argument with verifier for the statement that $\mathsf{val}_{\mathsf{out}}$ is the output of an N-step execution of the RAM program RAM with instruction set $\mathcal{F}'$, i.e., $\mathsf{RAM}(\mathsf{val}_{\mathsf{in}}) = \mathsf{val}_{\mathsf{out}}$. (See Section 2.6 and Appendix B.1 for the description of RAM programs.) This statement is formalized by $\mathcal{R}_{\mathsf{ram}}$ in (20).

$$\mathcal{R}_{\mathsf{ram}} = \left\{(\mathsf{cki}, \mathcal{F}'_{\mathsf{struct}}, [\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}}, \mathsf{val}_{\mathsf{out}}) \middle| [\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}} \wedge \mathsf{RAM}(\mathsf{val}_{\mathsf{in}}) = \mathsf{val}_{\mathsf{out}}\right\}. \tag{20}$$

In Section 6.1, we first reduce the constraints for proving a RAM program into those suitable for us to apply the generic CFS (Section 5). The adaptation to this generic CFS is presented in Section 6.2, by following the discussion from Section 2.6. Finally, in Section 6.3, we describe RAMenPaSTA.

6.1 Reducing Constraints

We first recall the components mentioned in Section 2.6. Let $N, T, M \in \mathbb{Z}_+$. An N-step RAM program has an instruction set $\mathcal{F}' = \{F'_j\}_{j\in[T]}$ representable by the PLONK structures $\{\mathsf{plkst}'_j\}_{j\in[T]}$. To prove the correct execution of a RAM program, as specified in (7), we need the following components for each $i \in [N]$.

$$\overline{\mathsf{pc}}_i,\ \overline{\mathsf{val}}_i,\ \mathsf{plkst}_i,\ \mathsf{auxplk}_i,\ \mathsf{pc}_i,\ \mathsf{macs}_i,\ \mathsf{macs}'_i \tag{21}$$

where $\overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}$, $\overline{\mathsf{val}}_i = \mathsf{val}_{i-1}$, $\mathsf{macs}_i = (\ell_i, \mathsf{time}_i, \mathsf{val}_i, \mathsf{mop}_i)$ and $\mathsf{macs}'_i = (\ell'_i, \mathsf{time}'_i, \mathsf{val}'_i, \mathsf{mop}'_i)$. With $\gamma, \delta \xleftarrow{\$} \mathbb{F}$ sampled in advance, the correctness of the RAM program, as in (20), with soundness error $\mathcal{O}(N \cdot n_{\mathsf{plk}}/|\mathbb{F}|)$ due to the use of PLONK's arithmetization with random γ, δ discussed in Section 2.6, by proving

$$\begin{cases} \mathsf{C}_{\mathsf{mem}}(i, \mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1}, \mathsf{macs}_i, \mathsf{macs}'_i) = 1\ \forall i \in [2, N], \\ \textit{Tuple Permutation:}\ (\mathsf{macs}_i)_{i\in[N]} \text{ is a permutation of } (\mathsf{macs}'_i)_{i\in[N]}, \\ \textit{Tuple Lookup:}\ \{(\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i)\}_{i\in[N]} \subseteq \{(j \| \mathsf{plkst}'_j)\}_{j\in[T]}, \\ \mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}(\mathsf{plkst}_i, \overline{\mathsf{val}}_i, (\mathsf{pc}_i \| \mathsf{macs}_i), \mathsf{auxplk}_i) = 1\ \forall i \in [N] \end{cases} \tag{22}$$

where $\mathsf{C}_{\mathsf{mem}}$ and $\mathsf{C}_{\mathsf{plk}}^{\gamma,\delta}$, parameterized by $\gamma, \delta \xleftarrow{\$} \mathbb{F}$, are public arithmetic circuits for memory checking and instruction execution, respectively. The detailed discussions of these two circuits are deferred to Appendices B.2 and B.3.

We now discuss the handling of constraints in (22).

Tuple Permutation in (22)**.** Recall the technique discussed in Section 2.4, to prove tuple argument. We use challenges $\tau, \omega \xleftarrow{\$} \mathbb{F}$ to compute, for all $i \in [N]$,

$$\mathsf{miv}_i = (\tau + \langle \mathsf{macs}_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \text{ and } \mathsf{miv}'_i = (\tau + \langle \mathsf{macs}'_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \tag{23}$$

with probability for division by zero bounded by $\mathcal{O}(N/|\mathbb{F}|)$. Then, we can prove this tuple lookup argument by proving that $\sum_{i\in[N]} \mathsf{miv}_i = \sum_{i\in[N]} \mathsf{miv}'_i$. The argument has soundness error $\mathcal{O}(4 \cdot N/|\mathbb{F}|) = \mathcal{O}(N/|\mathbb{F}|)$ (see (45) in Appendix B.4).

Tuple Lookup in (22)**.** Similarly, we use the technique discussed in Section 2.4 to prove this tuple lookup argument. Let $n_{\mathsf{plk}} = \left|\mathsf{plkst}'_j\right|$ for any $j \in [T]$. Then, we prove this tuple lookup argument by proving the existence of $(\mathsf{mul}_j)_{j\in[T]} \subseteq \mathbb{F}$ such that, for $\chi, \psi \xleftarrow{\$} \mathbb{F}$ sampled in advance, $\sum_{i\in[N]} \mathsf{plkiv}_i = \sum_{j\in[T]} \mathsf{plkiv}'_j$ where

$$\begin{aligned} \mathsf{plkiv}_i &= (\chi + \langle (\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1} \quad \forall i \in [N], \\ \mathsf{plkiv}'_j &= \mathsf{mul}_j \cdot (\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1} \quad \forall j \in [T]. \end{aligned} \tag{24}$$

This reduced argument implies the tuple lookup argument with probability for division by zero bounded by $\mathcal{O}((N+T)/|\mathbb{F}|)$ and soundness error $\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$ (see (48) in Appendix B.4).

Putting All Together. Let $\gamma, \delta, \tau, \omega, \chi, \psi \xleftarrow{\$} \mathbb{F}$ and $(\mathsf{mul}_j)_{j\in[T]} \in \mathbb{F}$ be prepared in advance. Let $\mathsf{plkiv}'_j = \mathsf{mul}_j \cdot (\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1}$ for $j \in [T]$. With those above discussions for tuple permutation and tuple lookup in (22), we enhance components in (21) to additionally contain miv_i, miv'_i and plkiv_i for $i \in [N]$. Hence, for each $i \in [N]$, the components include

$$\underbrace{\overline{\mathsf{pc}}_i,\ \overline{\mathsf{val}}_i,\ \mathsf{plkst}_i,\ \mathsf{auxplk}_i,\ \mathsf{pc}_i,\ \mathsf{macs}_i,\ \mathsf{macs}'_i}_{\text{as of (21)}},\ \underbrace{\mathsf{miv}_i,\ \mathsf{miv}'_i,\ \mathsf{plkiv}_i}_{\text{enhanced}} \tag{25}$$

where $\overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}$, $\overline{\mathsf{val}}_i = \mathsf{val}_{i-1}$, $\mathsf{macs}_i = (\ell_i, \mathsf{time}_i, \mathsf{val}_i, \mathsf{mop}_i)$ and $\mathsf{macs}'_i = (\ell'_i, \mathsf{time}'_i, \mathsf{val}'_i, \mathsf{mop}'_i)$. Then, system (22) is reduced, with some negligible completeness error, to the combination of (26), (27) and (28) below. See Lemma 2 for the soundness of this reduction.

$$\mathsf{C}_{\mathsf{mem}}(i, \mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1}, \mathsf{macs}_i, \mathsf{macs}'_i) = 1 \ \forall i \in [2, N], \tag{26}$$

$$\begin{cases} \mathsf{miv}_i = (\tau + \langle \mathsf{macs}_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \ \forall i \in [N], \\ \mathsf{miv}'_i = (\tau + \langle \mathsf{macs}'_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \ \forall i \in [N], \\ \mathsf{plkiv}_i = (\chi + \langle (\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1} \ \forall i \in [N], \\ \mathsf{C}_{\mathsf{plk}}^{\gamma,\delta}(\mathsf{plkst}_i, \overline{\mathsf{val}}_i, (\mathsf{pc}_i \| \mathsf{macs}_i), \mathsf{auxplk}_i) = 1 \ \forall i \in [N], \end{cases} \tag{27}$$

$$\sum_{i\in[N]} \mathsf{miv}_i = \sum_{i\in[N]} \mathsf{miv}'_i, \quad \sum_{i\in[N]} \mathsf{plkiv}_i = \sum_{j\in[T]} \mathsf{plkiv}'_j. \tag{28}$$

Lemma 2. *Let* $\gamma, \delta, \tau, \omega, \chi, \psi \xleftarrow{\$} \mathbb{F}$ *and* $(\mathsf{mul}_j)_{j\in[T]} \in \mathbb{F}$ *be prepared in advance. Let* $\mathsf{plkiv}'_j = \mathsf{mul}_j \cdot (\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1}$ *for* $j \in [T]$*. Then, (i)* (22) *implies* (26), (27) *and* (28) *with probability* $1 - \mathcal{O}((N+T)/|\mathbb{F}|)$ *due to division by zero; (ii) And* (26), (27) *and* (28) *together imply* (22) *with soundness error at most* $\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$.

The proof of Lemma 2 is deferred to Appendix D.1.

Remark 8. The purpose of splitting into (26), (27) and (28) rather than making them a single system of constraints is to distinguish them as follows. (26) is for the condition between two consecutive instance-witness pairs. (27) is for constraints those inside a single instance-witness pair. And, (28) is for proving tuple permutation and tuple lookup as discussed in Section 2.4. Hence, splitting them into three separate (system of) equations is convenient for us to design instance-witness pairs for folding in the following Section 6.2.

6.2 Partitioning Components and Adapting to Generic CFS

We first discuss how to put those components in (25), in Section 6.1, into instance-witness pairs suitable for applying generic CFS in Section 5.3. Then, we specify the necessary relations for folding, as required in Section 5.2.

Partition components in (25), into $\mathbf{z}_{ij}$ of length m_j, $\forall i \in [N], \forall j \in [5]$, below.

$$\begin{aligned} &\mathbf{z}_{i1} = (\overline{\mathsf{pc}}_i \| \overline{\mathsf{val}}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i) \in \mathbb{F}^{m_1}, \quad \mathbf{z}_{i2} = (\mathsf{pc}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i) \in \mathbb{F}^{m_2}, \\ &\mathbf{z}_{i3} = \mathsf{plkst}_i \in \mathbb{F}^{m_3}, \quad \mathbf{z}_{i4} = \mathsf{aux}_i \in \mathbb{F}^{m_4}, \quad \mathbf{z}_{i5} = (\mathsf{miv}_i \| \mathsf{miv}'_i \| \mathsf{plkiv}_i) \in \mathbb{F}^{m_5} \end{aligned} \tag{29}$$

where aux_i (containing auxplk_i) is the auxiliary input supporting the verification of system (27) w.r.t. $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{n \times m}$ defined in the following Definition 8.

Definition 8. *For $n \in \mathbb{Z}_+$ and $m = 1 + \sum_{j \in [5]} m_j$, $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}^{n \times m}$, with $\gamma, \delta, \tau, \omega, \chi, \psi$ hardwired, are defined s.t., for all $\mathbf{z}_{i1} \in \mathbb{F}^{m_1}, \ldots, \mathbf{z}_{i5} \in \mathbb{F}^{m_5}$,*

$$\mathbf{A} \cdot (1 \| \mathbf{z}_{i1} \| \ldots \| \mathbf{z}_{i5}) \circ \mathbf{B} \cdot (1 \| \mathbf{z}_{i1} \| \ldots \| \mathbf{z}_{i5}) = \mathbf{C} \cdot (1 \| \mathbf{z}_{i1} \| \ldots \| \mathbf{z}_{i5})$$

iff (27) holds and the suffixes of $\mathbf{z}_{i1}$ and $\mathbf{z}_{i2}$, w.r.t. macs_i and macs'_i, are equal.

Remark 9. We note that, for $i \in [N]$, $\mathbf{A}, \mathbf{B}, \mathbf{C}$, $\mathbf{z}_{i4}$ and $\mathbf{z}_{i5}$ are determined only after $\gamma, \delta, \tau, \omega, \chi, \psi$ are known. This is due to constraints in (27).

Condition for Two Consecutive Instance-Witness Pairs. For $i \in [2, N]$, according to our partition of components in (29), we define matrices $\mathbf{A}', \mathbf{B}', \mathbf{C}'$ to enforce the condition between $\mathbf{z}_{(i-1)2}$ and $\mathbf{z}_{i1}$ as in the following Definition 9.

Definition 9. *$\mathbf{A}', \mathbf{B}', \mathbf{C}'$ are defined in advance s.t., $\forall i \in [2, N]$, exist $\mathbf{w}_i$ s.t.*

$$\mathbf{A}' \cdot ((1,i) \| \mathbf{z}_{(i-1)2} \| \mathbf{z}_{i1} \| \mathbf{w}_i) \circ \mathbf{B}' \cdot ((1,i) \| \mathbf{z}_{(i-1)2} \| \mathbf{z}_{i1} \| \mathbf{w}_i) = \mathbf{C}' \cdot ((1,i) \| \mathbf{z}_{(i-1)2} \| \mathbf{z}_{i1} \| \mathbf{w}_i)$$

iff (i) $\overline{\mathsf{pc}}_i$ in $\mathbf{z}_{i1}$ is equal to pc_{i-1} in $\mathbf{z}_{(i-1)2}$; (ii) $\overline{\mathsf{val}}_i$ in $\mathbf{z}_{i1}$ is equal to val_{i-1} in macs_{i-1} in $\mathbf{z}_{(i-1)2}$; and (iii) $\mathsf{C}_{\mathsf{mem}}(i, \mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1}, \mathsf{macs}_i, \mathsf{macs}'_i) = 1$ (as of (26)) macs_{i-1} and macs'_{i-1} are in $\mathbf{z}_{(i-1)2}$, and macs'_i and macs'_i are in $\mathbf{z}_{i1}$.

We assume that $\mathbf{A}', \mathbf{B}', \mathbf{C}' \in \mathbb{F}^{n' \times m'}$ where, for $m'_{\mathsf{pub}} = 2$, $m'_1 = m_2$, $m'_2 = m_1$ and $m'_3 = |\mathbf{w}_i|$, n' is some integer in $\mathbb{Z}_+$ and $m' = m'_{\mathsf{pub}} + \sum_{j \in [3]} m'_j$. Here, setting $m'_1 = m_2$ and $m'_2 = m_1$ is due to $|\mathbf{z}_{(i-1)2}| = m_2$ and $|\mathbf{z}_{i1}| = m_1$.

In other words, $\mathbf{z}_{i1}$ in step i is designed to capture some constraints with $\mathbf{z}_{(i-1)2}$ in the previous step $i-1$. This includes the correct use of pc_{i-1} and val_{i-1} from the previous step $i-1$ to determine the computations in the current step i (see Section 2.6 and Appendix B.1) and the memory constraint in (26).

Capturing Tuple Permutation and Tuple Lookup. From the partition in (29), we see that $\mathbf{z}_{i5} = (\mathsf{miv}_i \| \mathsf{miv}'_i \| \mathsf{plkiv}_i)$. Since we need to take the sums $\sum_{i \in [N]} \mathsf{miv}_i$, $\sum_{i \in [N]} \mathsf{miv}'_i$ and $\sum_{i \in [N]} \mathsf{plkiv}'_i$ as specified in (28). Notice that, we do not need to consider the $\{\mathsf{plkst}'_j\}_{j \in [T]}$ since these can be computed publicly from the public set $\{\mathsf{plkst}'_j\}_{j \in [T]}$ and random challenges χ, ψ.

Putting All Together. Let

$$\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C}) \text{ and } \mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}') \tag{30}$$

specified above. Let HS be defined in Definition 1. We adapt the above discussion into the generic CFS $\mathcal{CF}_{\mathsf{gnr}}$ described in Section 5.3. Recall the form of instance-witness pairs in (15). For $i \in [N]$, we design $[\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}$ to be

$$([\![\mathbb{x}_{(i-1)i}]\!]_{\mathsf{tck}}, [\![\mathsf{front}_{(i-1)i}]\!]_{\mathsf{ck}_1}, [\![\mathsf{rear}_{(i-1)i}]\!]_{\mathsf{ck}_2}, [\![\mathbb{x}^{\star}_{(i-1)i}]\!]_{\mathsf{tck}'}, [\![\mathbf{s}_{(i-1)i}]\!]_{\mathsf{ck}_5}, [\![\mathbf{a}_{(i-1)i}]\!]_{\mathsf{ck}_5})$$

whose components are as follows. The rR1CS instance-witness pair $[\![\mathbb{x}_{(i-1)i}]\!]_{\mathsf{tck}}$ is the tuple (31) whose $[\![\mathbf{0}^n]\!]_{\mathsf{cke}}$ is committed as in Remark 7.

$$\begin{aligned} &(\mathbb{x}_{(i-1)i}.u, \mathbb{x}_{(i-1)i}.\mathsf{pub}, [\![\mathbb{x}_{(i-1)i}.\mathbf{z}_1]\!]_{\mathsf{ck}_1}, \ldots, [\![\mathbb{x}_{(i-1)i}.\mathbf{z}_5]\!]_{\mathsf{ck}_5}, [\![\mathbb{x}_{(i-1)i}.\mathbf{e}]\!]_{\mathsf{cke}}) \\ &= (1, 1, [\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1}, \ldots [\![\mathbf{z}_{i5}]\!]_{\mathsf{ck}_5}, [\![\mathbf{0}^n]\!]_{\mathsf{cke}}). \end{aligned} \tag{31}$$

Notice that we set d in Section 5.1 to be $d = 5$. The rR1CS instance-witness pair $[\![\mathbb{x}^{\star}_{(i-1)i}]\!]_{\mathsf{tck}'}$, as a condition accumulator, is set randomly s.t. $[\![\mathbb{x}^{\star}_{(i-1)i}]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ (see (19)) with $\mathfrak{S}'$ defined in (30). Since this accumulator is used for folding the instance-witness pairs for the conditions at the leaf nodes, as there is no related condition, we simply set it to be random.

$[\![\mathsf{front}_{(i-1)i}]\!]_{\mathsf{ck}_1}$ and $[\![\mathsf{rear}_{(i-1)i}]\!]_{\mathsf{ck}_2}$ respectively are $[\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1}$ and $[\![\mathbf{z}_{i2}]\!]_{\mathsf{ck}_2}$. Recall that, from Section 5.1, $\mathsf{tck}' = (\mathsf{ck}'_1, \mathsf{ck}'_2, \mathsf{ck}'_3, \mathsf{cke}')$. As we use $\mathsf{front}_{(i-1)i}$ and $\mathsf{rear}_{(i-1)i}$ for capturing the condition, we hence enforce $\mathsf{ck}'_1 = \mathsf{ck}_2$ and $\mathsf{ck}'_2 = \mathsf{ck}_1$.

Finally, we set $[\![\mathbf{s}_{(i-1)i}]\!]_{\mathsf{ck}_5} := [\![\mathbf{z}_{i5}]\!]_{\mathsf{ck}_5}$ and $[\![\mathbf{a}_{(i-1)i}]\!]_{\mathsf{ck}_5}$ to be $[\![\mathbf{0}^3]\!]_{\mathsf{ck}_5}$ (committed as in Remark 7). Since, as explained above, $\mathbf{z}_{i5}$ contains only 3 $\mathbb{F}$-elements. Therefore, we enforce cks (in Section 5.1) to be $\mathsf{cks} = \mathsf{ck}_5 = 3$.

We now construct RAMenPaSTA (Section 6.3) considering $\mathcal{CF}_{\mathsf{gnr}}$ (Section 5.3) as a building block with the setting of $\{[\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}\}_{i\in[N]}$, at the leaf nodes of HS.

6.3 Description of RAMenPaSTA

We describe RAMenPaSTA, as Π_{RP} (c.f. Figure 7), for relation $\mathcal{R}_{\mathsf{ram}}$ (c.f. (20)). We first need to prove the constraints specified in Lemma 2 clarified as follows.

Initial Setting. Due to Remark 9, $\{\mathbf{z}_{ij}\}_{i\in[N],j\in[4,5]}$ are determined only after $\gamma, \delta, \tau, \omega, \chi, \psi$ are known. Parse $\mathbf{z}_{11} = (\overline{\mathsf{pc}}_1 \| \overline{\mathsf{val}}_1 \| \mathsf{macs}_1 \| \mathsf{macs}'_1)$ and $\mathbf{z}_{N2} = (\mathsf{pc}_N \| \mathsf{macs}_N \| \mathsf{macs}'_N)$. Let $\overline{\mathsf{pc}}_1 = \mathsf{pc}_0 = 1$ be the initial program counter, $\overline{\mathsf{val}}_1 = \mathsf{val}_0 = \mathsf{val}_{\mathsf{in}}$ and $\mathsf{val}_N = \mathsf{val}_{\mathsf{out}}$. Moreover, $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$ can be determined without knowing $\gamma, \delta, \tau, \omega, \chi, \psi$ (see Definition 9).

When $\gamma, \delta, \tau, \omega, \chi, \psi$ are known, prover can compute $\{\mathbf{z}_{ij}\}_{i\in[N],j\in[4,5]}$. Moreover, both parties can commonly determine $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C})$ following Definition 8. Hence, at this stage, both parties can determine necessary components as specified in Section 6.2 to fold with public inputs $(\mathbf{p}_i)_{i\in[2,N]} = (1, i)_{i\in[2,N]}$. In the end, we need to run $\mathsf{CF.Prove}$ which is a ZKAoK for $[\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$. However, $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ does not suffice for proving relation $\mathcal{R}_{\mathsf{ram}}$. In fact, $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ *does not* imply those for related input and output (i.e., the values $\overline{\mathsf{val}}$ and val in $\mathbb{z}_{0N}$ are $\mathsf{val}_{\mathsf{in}}$ and $\mathsf{val}_{\mathsf{out}}$, respectively), tuple permutation and tuple lookup.

Capturing $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ with Enhanced Constraints. To capture the mentioned constraints, we must prove $\sum_{i\in[N]} \mathsf{miv}_i = \sum_{i\in[N]} \mathsf{miv}'_i$ and $\sum_{i\in[N]} \mathsf{plkiv}_i = \sum_{j\in[T]} \mathsf{plkiv}'_j$ (see (28)). After folding following HS, the final instance-witness pair $\mathbb{z}_{0N}$ contains $\mathbf{s}_{0N} = \sum_{i\in[N]} \mathbf{s}_{(i-1)i}$ as explained in Section 2.4. By parsing $\mathbf{s}_{0N} = (\mathsf{plkiv} \| \mathsf{miv} \| \mathsf{miv}')$, we need to prove $\mathsf{miv} = \sum_{j\in[T]} \mathsf{plkiv}'_j$ and $\mathsf{miv} = \mathsf{miv}'$ where $\mathsf{plkiv}'_j = \mathsf{mul}_j \cdot (\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1}$ for $j \in [T]$ (see Lemma 2). Moreover, we also need to prove the correct commitment of input $\mathsf{val}_{\mathsf{in}}$, i.e., $[\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}}$ and the consistency between the input $\mathsf{val}_{\mathsf{in}}$ and output $\mathsf{val}_{\mathsf{out}}$ of the RAM program. Observing that front_{0N} and rear_{0N} respectively are equal to front_{11} (in $\mathbb{z}_{11}$) and rear_{N2} (in $\mathbb{z}_{N2}$) due to the folding strategy specified in Sections 2.2 and 5.3. Hence, we need to additionally prove that input in front_{0N} and output in rear_{0N} respectively match $\mathsf{val}_{\mathsf{in}}$ and $\mathsf{val}_{\mathsf{out}}$. Thus, we have the relation

$$\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}} = \left\{ \begin{array}{l} (\mathsf{pp}, \mathsf{cki}, (\mathsf{ckm}_j)_{j\in[T]}, \\ [\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}}, \mathsf{val}_{\mathsf{out}}, \\ ([\![\mathsf{mul}_j]\!]_{\mathsf{ckm}_j})_{j\in[T]}, \\ [\![\mathbb{z}]\!]_{\mathsf{pp}}) \end{array} \middle| \begin{array}{l} [\![\mathbb{z}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} \wedge [\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}} \\ \wedge \overline{\mathsf{pc}} = 1 \wedge \overline{\mathsf{val}} = \mathsf{val}_{\mathsf{in}} \wedge \mathsf{val} = \mathsf{val}_{\mathsf{out}} \\ \wedge ([\![\mathsf{mul}_j]\!]_{\mathsf{ckm}_j} \in \mathcal{R}_{\mathsf{com}} \ \forall j \in [T]) \\ \wedge \mathsf{plkiv} = \sum_{j\in[T]} \frac{\mathsf{mul}_j}{\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle} \\ \wedge \mathsf{miv} = \mathsf{miv}' \end{array} \right\}. \tag{32}$$

where (i) cki and $\{\mathsf{ckm}_j\}_{j\in[T]}$ are for committing $\mathsf{val}_{\mathsf{in}}$ and $\{\mathsf{mul}_j\}_{j\in[T]}$, respectively; (ii) $[\![\mathbb{z}]\!]_{\mathsf{pp}} = ([\![\mathbb{x}]\!]_{\mathsf{tck}}, [\![\mathsf{front}]\!]_{\mathsf{ck}_2}, [\![\mathsf{rear}]\!]_{\mathsf{ck}_1}, [\![\mathbb{x}^{\star}]\!]_{\mathsf{tck}'}, [\![\mathbf{s}]\!]_{\mathsf{ck}_5}, [\![\mathbf{a}]\!]_{\mathsf{ck}_5})$, of the form (14), and $\mathbf{s} = (\mathsf{plkiv} \| \mathsf{miv} \| \mathsf{miv}')$ where commitment key setting is discussed below; (iii) $\mathsf{pp} = (\mathsf{tck}, \mathsf{tck}', \mathsf{cks})$, $\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_5, \mathsf{cke})$ and $\mathsf{tck}' = (\mathsf{ck}'_1, \mathsf{ck}'_2, \mathsf{ck}'_3, \mathsf{cke}')$ where $\mathsf{ck}'_1 = \mathsf{ck}_2$, $\mathsf{ck}'_2 = \mathsf{ck}_1$ and $\mathsf{cks} = \mathsf{ck}_5 = 3$ as explained in Section 6.1; and (iv) $\mathsf{front} = (\overline{\mathsf{pc}} \| \overline{\mathsf{val}} \| \ldots)$ and $\mathsf{rear} = (\mathsf{pc} \| \mathsf{macs} \| \mathsf{macs}')$ s.t. $\mathsf{macs} = (\ell, \mathsf{time}, \mathsf{val}, \mathsf{mop})$ as in memory consistency in Section 2.6.

Hence, proving $[\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ in (32) implies not only $[\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ but also the constraints in relation $\mathcal{R}_{\mathsf{ram}}$ in (20) due to Lemma 2. RAMenPaSTA is described in Figure 7 as Π_{RP} with security in Theorem 2.

$\underline{\Pi_{\mathsf{RP}}\left(\mathsf{pp}, \mathsf{cki}, (\mathsf{ckm}_j)_{j\in[T]}, \mathcal{F}'_{\mathsf{struct}}, [\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}}, \mathsf{val}_{\mathsf{out}};\ (\mathbf{z}_{ij})_{i\in[N],j\in[3]}, (\mathsf{mul}_j)_{j\in[T]}\right) \to \{0,1\}}$

With the parameters $m_1, \ldots, m_5$, n, $m'_1, \ldots, m'_5$, n' and

$$\mathsf{pp} = (\mathsf{tck}, \mathsf{tck}', \mathsf{cks}) = ((\mathsf{ck}_1, \ldots, \mathsf{ck}_5, \mathsf{cke}), (\mathsf{ck}'_1, \ldots, \mathsf{ck}'_3, \mathsf{cke}'))$$

s.t. $\mathsf{ck}'_1 = \mathsf{ck}_2$, $\mathsf{ck}'_2 = \mathsf{ck}_1$ and $\mathsf{cks} = \mathsf{ck}_5 = 3$ (see Section 5.1), Π_{RP} runs as follows.

1. Both parties run protocols, from Π_{com} (see Remark 4).
$$[\![\mathbf{z}_{ij}]\!]_{\mathsf{ck}_j} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ck}_j;\ \mathbf{z}_{ij})\ \forall i \in [N], \forall j \in [3],$$
$$[\![\mathbf{0}^n]\!]_{\mathsf{cke}} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke};\ \mathbf{0}^n), \quad [\![\mathbf{0}^{n'}]\!]_{\mathsf{cke}'} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke}';\ \mathbf{0}^{n'}),$$
$$[\![\mathbf{0}^3]\!]_{\mathsf{ck}_5} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ck}_5, \mathbf{0}^3), \quad [\![\mathsf{mul}_j]\!]_{\mathsf{ckm}_j} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ckm}_j;\ \mathsf{mul}_j)\ \forall j \in [T]$$
where committing to $\mathbf{0}^n$, $\mathbf{0}^{n'}$, $\mathbf{0}^3$ by cke, cke', cke_5, respectively can be done locally and independently by each party according to Remark 7.
2. Verifier: $\gamma, \delta, \tau, \omega, \chi, \psi \xleftarrow{\$} \mathbb{F}$ and send $\gamma, \delta, \tau, \omega, \chi, \psi$ to prover.
3. Each party determines $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C}), \mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$ from $\gamma, \delta, \tau, \omega, \chi, \psi$.
4. Prover:
   - Determine $(\mathbf{z}_{i4}, \mathbf{z}_{i5})_{i\in[N]}$ from $(\mathbf{z}_{i1}, \mathbf{z}_{i2}, \mathbf{z}_{i3})_{i\in[N]}$ and $\gamma, \delta, \tau, \omega, \chi, \psi$.
   - Find arbitrary $\mathbf{p}' \in \mathbb{F}^{m'_{\mathsf{pub}}}$ and $\{\mathbf{z}'_i\}_{i\in[3]}$ s.t. $\mathbf{z}'_i \in \mathbb{F}^{m'_i}$ and $\mathbf{A}' \cdot \mathbf{c}' \circ \mathbf{B}' \cdot \mathbf{c}' = \mathbf{C} \cdot \mathbf{c}'$ where $\mathbf{c}' = (\mathbf{p}' \| \mathbf{z}'_1 \| \mathbf{z}'_2 \| \mathbf{z}'_3)$.
5. Both parties run $[\![\mathbf{z}'_j]\!]_{\mathsf{ck}'_j} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ck}'_j;\ \mathbf{z}'_j)\ \forall j \in [3]$.
6. For each $i \in [N]$, both parties form tuple $[\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}$, of the form (15), by setting
$$[\![\mathsf{front}_{(i-1)i}]\!]_{\mathsf{ck}_1} := [\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1},\ [\![\mathsf{rear}_{(i-1)i}]\!]_{\mathsf{ck}_2} := [\![\mathbf{z}_{i2}]\!]_{\mathsf{ck}_2},$$
$$[\![\mathbb{x}_{(i-1)i}]\!]_{\mathsf{tck}} := (1, 1, [\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1}, \ldots, [\![\mathbf{z}_{i5}]\!]_{\mathsf{ck}_5}, [\![\mathbf{0}^n]\!]_{\mathsf{cke}}),$$
$$[\![\mathbb{x}^\star_{(i-1)i}]\!]_{\mathsf{tck}'} := (1, \mathbf{p}', [\![\mathbf{z}'_1]\!]_{\mathsf{ck}'_1}, \ldots, [\![\mathbf{z}'_3]\!]_{\mathsf{ck}'_3}, [\![\mathbf{0}^{n'}]\!]_{\mathsf{cke}'}),$$
$$[\![\mathbf{s}_{(i-1)i}]\!]_{\mathsf{ck}_5} := [\![\mathbf{z}_{i5}]\!]_{\mathsf{ck}_5}, \quad [\![\mathbf{a}_{(i-1)i}]\!]_{\mathsf{ck}_5} := [\![\mathbf{0}^3]\!]_{\mathsf{ck}_5}$$
where $[\![\mathbb{x}_{(i-1)i}]\!]_{\mathsf{tck}}$ and $[\![\mathbb{x}^\star_{(i-1)i}]\!]_{\mathsf{tck}'}$ are of the forms (16) and (17), respectively.
7. Both parties fold $\{[\![\mathbb{z}_{(i-1)i}]\!]\}_{i\in[N]}$ into $[\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}}$ following the topological order of HS s.t., when folding $[\![\mathbb{z}_{lj}]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}_{jr}]\!]_{\mathsf{pp}}$ into $[\![\mathbb{z}_{lr}]\!]_{\mathsf{pp}}$ (where $(l,j),(j,r),(l,r) \in \mathsf{HS}$),
$$[\![\mathbb{z}_{lr}]\!]_{\mathsf{pp}} \leftarrow \mathsf{CF.Fold}(\mathsf{pp}, [\![\mathbb{z}_{lj}]\!]_{\mathsf{pp}}, [\![\mathbb{z}_{jr}]\!]_{\mathsf{pp}}, (1, j+1);\ \mathbf{w})$$
where $(1, j+)$ is the additional public input required as in Definition 9. This step can be done in parallel, e.g., following the strategy in Section 2.3.
8. Finally, run $\mathsf{CF.Prove}$ as a ZKAoK (or ZKPoK) $\Pi_{\mathsf{RP\text{-}prf}}$ for showing
$$(\mathsf{pp}, \mathsf{cki}, (\mathsf{ckm}_j)_{j\in[T]}, [\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}}, \mathsf{val}_{\mathsf{out}}, ([\![\mathsf{mul}_j]\!]_{\mathsf{ckm}_j})_{j\in[T]}, [\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}}) \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}.$$

Fig. 7. RAMenPaSTA as protocol Π_{RP}.

Theorem 2 (Security of Π_{RP}). *If $\mathcal{C}$ is a homomorphic commitment scheme and $\Pi_{\mathsf{RP\text{-}prf}}$ is an HVZKAoK for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ then Π_{RP} is an HVZKAoK for $\mathcal{R}_{\mathsf{ram}}$ in (20) with completeness error $\mathcal{O}((N+T)/|\mathbb{F}| + \mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp}))$ and soundness error $\mathcal{O}\left(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}| + \mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda)\right)$ where $\mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp})$ and $\mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp})$ are respectively completeness and soundness error of $\Pi_{\mathsf{RP\text{-}prf}}$ and $\mathsf{negl}(\lambda)$ is the negligible probability for cheating prover to break the binding of $\mathcal{C}$.*

The security of Π_{RP} follows from the security of the underlying commitment scheme, CFS $\mathcal{CF}_{\mathsf{gnr}}$ and HVZKAoK. For knowledge soundness, we can extract the witnesses of Π_{RP}, satisfying requirements of Lemma 2, implying the satisfaction of (22) and (20). The full proof of Theorem 2 is deferred to Appendix D.2.

Efficiency. To analyze efficiency, we first analyze as follows.

- We first consider the components in (29). For $i \in [N]$, $|\overline{\mathsf{pc}}_i| = |\mathsf{pc}_i| = \mathcal{O}(1)$, $\left|\overline{\mathsf{val}}_i\right| = |\mathsf{val}_i| = \mathcal{O}(1)$, $|\mathsf{macs}|_i = |\mathsf{macs}'_i| = \mathcal{O}(1)$, $|\mathsf{plkst}_i| = n_{\mathsf{plk}}$, $|\mathsf{aux}_i| = \mathcal{O}(n_{\mathsf{plk}})$ since it contains witness for PLONK's arithmetization, which is bounded by n_{plk}, and other constraints related to $\mathcal{O}(1)$-size components.
- $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ have $\mathcal{O}(n_{\mathsf{plk}})$ non-zero entries (Remark 11 in Appendix B.3).
- $\mathbf{A}'$, $\mathbf{B}'$ and $\mathbf{C}'$ have $\mathcal{O}(\log|\mathbb{F}|)$ non-zero entries (Remark 10 in Appendix B.2).
- $\mathbf{w}_i$ in Definition 9 has size of $\mathcal{O}(\log|\mathbb{F}|)$ (Remark 10 in Appendix B.2).

Recall the notations in Definition 2. The efficiency of Π_{RP} is as follows.

- *Public Input Size.* $\mathcal{O}(1)$ for commitment in $[\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}}$ and size of $\mathsf{val}_{\mathsf{out}}$.

- *Communication Cost.* $\mathcal{O}(T + N \cdot (\mathsf{c}(n_{\mathsf{plk}}) + \mathsf{c}(\log|\mathbb{F}|)) + \mathsf{p})$ where we assume $\mathsf{c}(c) = \mathcal{O}(1)$ for any constant c and p is the proof size of $\mathsf{CF.Prove}$. This cost is from the commitments to $\{\mathsf{mul}_j\}_{j\in[T]}$, the $N-1$ times of running $\mathsf{CF.Fold}$ (step 7) and the proof size p of $\mathsf{CF.Prove}$.
- *Prover Time.* $\mathcal{O}(T + N \cdot (n_{\mathsf{plk}} + \log|\mathbb{F}| + \mathsf{tp}^{\mathsf{c}}(n_{\mathsf{plk}}) + \mathsf{tp}^{\mathsf{c}}(\log|\mathbb{F}|) + \mathsf{tp}^{\mathsf{h}}(n_{\mathsf{plk}}) + \mathsf{tp}^{\mathsf{h}}(\log|\mathbb{F}|)) + \mathsf{tp}^{\mathsf{p}})$ where $\mathsf{tp}^{\mathsf{c}}(c) = \mathsf{tp}^{\mathsf{h}}(c) = \mathcal{O}(1)$ for any constant c and tp^{p} is the prover time for $\mathsf{CF.Prove}$. The analysis of prover time is as from above, for communication cost and $\mathsf{CF.Fold}$ in Section 5.3. The factor $\mathcal{O}(n_{\mathsf{plk}} + \log|\mathbb{F}|)$ is because of garb (c.f. Section 3.2) w.r.t. $\mathbf{A}, \mathbf{B}, \mathbf{C}$ of $\mathcal{O}(n_{\mathsf{plk}})$ non-zero entries and $\mathbf{A}', \mathbf{B}', \mathbf{C}'$ of $\mathcal{O}(\log|\mathbb{F}|)$ non-zero entries.
- *Verifier Time.* $\mathcal{O}(T + N \cdot (\mathsf{tv}^{\mathsf{c}}(n_{\mathsf{plk}}) + \mathsf{tv}^{\mathsf{c}}(\log|\mathbb{F}|) + \mathsf{tv}^{\mathsf{h}}(n_{\mathsf{plk}}) + \mathsf{tv}^{\mathsf{h}}(\log|\mathbb{F}|)) + \mathsf{tv}^{\mathsf{p}})$ where $\mathsf{tv}^{\mathsf{c}}(c) = \mathsf{tv}^{\mathsf{h}}(c) = \mathcal{O}(1)$ for any constant c and tv^{p} is the verifier time for $\mathsf{CF.Prove}$. This is analyzed as from above.

When parallelizing with N threads employing the strategy in Section 2.3, we can achieve the following efficiency for Π_{RP}.

- *Public Input Size.* $\mathcal{O}(1)$.
- *Communication Cost.* $\mathcal{O}(T + N \cdot (\mathsf{c}(n_{\mathsf{plk}}) + \mathsf{c}(\log|\mathbb{F}|)) + \mathsf{p})$ in total.
- *Prover Time.* $\mathcal{O}(T/N + \log N \cdot (n_{\mathsf{plk}} + \log|\mathbb{F}| + \mathsf{tp}^{\mathsf{c}}(n_{\mathsf{plk}}) + \mathsf{tp}^{\mathsf{c}}(\log|\mathbb{F}|) + \mathsf{tp}^{\mathsf{h}}(n_{\mathsf{plk}}) + \mathsf{tp}^{\mathsf{h}}(\log|\mathbb{F}|)) + \mathsf{tp}^{\mathsf{p}})$.
- *Verifier Time.* $\mathcal{O}(T/N + \log N \cdot \big(\mathsf{tv}^{\mathsf{c}}(n_{\mathsf{plk}}) + \mathsf{tv}^{\mathsf{c}}(\log|\mathbb{F}|) + \mathsf{tv}^{\mathsf{h}}(n_{\mathsf{plk}}) + \mathsf{tv}^{\mathsf{h}}(\log|\mathbb{F}|)\big) + \mathsf{tv}^{\mathsf{p}})$.

Instantiating with compressed Σ-protocol from [AC20], we achieve the following efficiency for two settings, namely, (i) sequential and (ii) parallel with N threads.

- *Public Input Size.* $\mathcal{O}(1)$ for both settings.
- *Communication Cost.* $\mathcal{O}(T + N + \log n_{\mathsf{plk}} + \log\log|\mathbb{F}|)$ in total for both settings. For the parallel setting with N threads, the communication cost for each thread can be $\mathcal{O}(T/N + \log N + \log n_{\mathsf{plk}} + \log\log|\mathbb{F}|)$.
- *Prover Time.* $\mathcal{O}(T + N \cdot (n_{\mathsf{plk}} + \log|\mathbb{F}|))$ for (i) and $\mathcal{O}(T/N + \log N \cdot (n_{\mathsf{plk}} + \log|\mathbb{F}|))$ for (ii).
- *Verifier Time.* $\mathcal{O}(T + N + n_{\mathsf{plk}} + \log|\mathbb{F}|)$ for (i) and $\mathcal{O}(T/N + \log N + n_{\mathsf{plk}} + \log|\mathbb{F}|)$ for (ii).

We achieve the above efficiency because the employed Pedersen commitment [Ped92] is $\mathcal{O}(1)$ and homomorphically processing commitments (for any vector of any length) takes $\mathcal{O}(1)$ time. The prover also needs to homomorphically evaluate the witnesses behind the commitments. See Appendix 7.1 for more details.

7 Instantiation

We provide potential instantiation of RAMenPaSTA presented in Figure 7 from compressed Σ-protocol theory [AC20] in Appendix 7.1.

7.1 Instantiation from Compressed Σ-Protocol Theory

Recall that, in [AC20], they construct a succinct ZKAoK for circuit satisfiability [AC20, Section 6] by applying Lagrange interpolation to transform the witness of computation, through an affine transformation, into a single check of multiplication of two finite field elements. Their construction employs the Pedersen commitment scheme [Ped92] (recalled in Appendix B.6) as a building block. Moreover, the Pedersen commitment scheme is doubly homomorphic [BMM+21] (homomorphic not only in commitment, message, and randomness, but also in commitment key), perfectly hiding and computationally binding, and succinct ZKAoK of [AC20] (recalled in Appendix B.12) for circuit satisfiability meets required properties in Theorem 2. Therefore, it is expected that RAMenPaSTA in Figure 7 can be instantiated by Pedersen commitment scheme to achieve a sub-linear[5] statistical ZKAoK for RAM programs.

Nevertheless, applying ZKAoK for circuit satisfiability in [AC20] is not direct. In fact, for proving $\mathsf{C}(\mathbf{x}) = 0$ given the public circuit C, the authors transform the witness of the computation $\mathsf{C}(\mathbf{x})$ into a witness vector $\mathbf{w} \in \mathbb{F}^w$ for some positive integer $w \in \mathbb{Z}_+$. Then, they commit to $\mathbf{w}$ to obtain a commitment $c \in \mathbb{G}$ for some group $\mathbb{G}$. Notice that to commit such a vector $\mathbf{w}$, they

[5] The proof size is linear only in N, and hence sub-linear in $N \cdot W$.

employ a grand commitment key $\mathsf{ck} = (g_1, \ldots, g_{w'}) \in \mathbb{G}^{w'}$ for some $w' \geq w$ such that each entry of ck is sampled independently and uniformly from $\mathbb{G}$. Nevertheless, commitments in $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ may be computed from the same keys, e.g., $\mathsf{ck}'_1 = \mathsf{ck}_2$ (namely, the same key used for committing both $\mathbb{x}.\mathbf{z}_2$ in (16) and $\mathbb{x}^\star.\mathbf{z}_1$ in (17)) as specified in Section 5.1. Hence, we cannot perform the proof for $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ at once because some keys, e.g., ck'_1 and ck_2 satisfying $\mathsf{ck}'_1 = \mathsf{ck}_2$, are not independently generated.

To overcome the issue, we can split constraints in relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ into L split-relations of $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$, for some constant $L \in \mathbb{Z}_+$, such that

- w.r.t. the same statement and witness, $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ is satisfied iff all split-relations are satisfied, and
- any two commitments involving constraints of the same split-relation do not correspond to the same commitment key.

Then, we devise a sufficiently long commitment key $\mathsf{ck} \in \mathbb{G}^{w'}$, for some $w' \in \mathbb{Z}_+$, that contains commitment keys of each split-relation. To commit components of any split-relation, we simply use entries of ck that are related to those components. Other unrelated entries of ck are used to commit to 0 to become $1 \in \mathbb{G}$. Moreover, the verifier can compute commitments w.r.t. those split-relations and grand commitment key ck by simply manipulating the commitments to components of each split-relation.

Now we need to construct a proof/argument that simultaneously proves all L split-relations. However, each split-relation has a specific affine transform to check the multiplication of two single field elements. Therefore, the prover and verifier proceed all L parallel proofs/arguments, for those L-split-relations, independently in $2\mu+1$ rounds, where $\mu = \mathcal{O}(\log w')$, in a way that the transcript of proof/argument for the i-th split-relation is represented by the sequence $(a^{(i)}_1, b^{(i)}_1, \ldots, a^{(i)}_\mu, b^{(i)}_\mu, a^{(i)}_{\mu+1})$ where $a^{(i)}_1, \ldots, a^{(i)}_{\mu+1}$ are prover's messages while $b^{(i)}_1, \ldots, b^{(i)}_\mu \in \mathbb{F}$ are verifier's challenges. Notice that, to simplify the proof of knowledge soundness and reduce the communication cost, we can enforce $b^{(1)}_i = \cdots = b^{(L)}_i$, for all $i \in [\mu]$, since those L proofs/arguments are independent with the same commitment key ck.

A Transformation into L Split-Relations for Constant L. Above we claimed that L is constant. Here, we provide a proof for this fact. We first take out all constraints in relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ in (32) as follows:

$$\begin{cases} [\![\mathbb{z}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}, \\ [\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}} \\ \overline{\mathsf{pc}} = 1 \wedge \overline{\mathsf{val}} = \mathsf{val}_{\mathsf{in}} \wedge \mathsf{val} = \mathsf{val}_{\mathsf{out}} \\ ([\![\mathsf{mul}_j]\!]_{\mathsf{ckm}_j} \in \mathcal{R}_{\mathsf{com}} \ \forall j \in [T]) \\ \mathsf{plkiv} = \sum_{j\in[T]} \frac{\mathsf{mul}_j}{\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle} \\ \mathsf{miv} = \mathsf{miv}' \end{cases}$$

where

- $\mathsf{pp} = (\mathsf{tck}, \mathsf{tck}') = ((\mathsf{ck}_1, \ldots, \mathsf{ck}_5, \mathsf{cke}), (\mathsf{ck}'_1, \ldots, \mathsf{ck}'_3, \mathsf{cke}'))$;
- $\mathbb{z} = (\mathbb{x}, \mathsf{front}, \mathsf{rear}, \mathbb{x}^\star, \mathbf{s}, \mathbf{a})$;
- $[\![\mathbb{z}]\!]_{\mathsf{pp}} = ([\![\mathbb{x}]\!]_{\mathsf{tck}}, [\![\mathsf{front}]\!]_{\mathsf{ck}_1}, [\![\mathsf{rear}]\!]_{\mathsf{ck}_2}, [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}, [\![\mathbf{s}]\!]_{\mathsf{ck}_5}, [\![\mathbf{a}]\!]_{\mathsf{ck}_5})$ as in (15);
- $\overline{\mathsf{pc}}$, $\overline{\mathsf{val}}$, pc, macs, macs' are obtained by parsing $\mathsf{front} = (\overline{\mathsf{pc}} \| \overline{\mathsf{val}} \| \ldots)$ and $\mathsf{rear} = (\mathsf{pc} \| \mathsf{macs} \| \mathsf{macs}')$ (see (32));
- $\mathsf{val}_{\mathsf{in}}$ and $\mathsf{val}_{\mathsf{out}}$ are input and output of a RAM program;
- $\gamma, \delta, \tau, \omega, \chi, \psi$ are challenges specified in Lemma 2;
- values miv, miv' and plkiv are obtained by parsing vector $\mathbf{s} = (\mathsf{miv} \| \mathsf{miv}' \| \mathsf{plkiv})$.

Moreover, from (18), $[\![\mathbb{z}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ is equivalent to

$$\begin{cases} [\![\mathsf{front}]\!]_{\mathsf{ck}_1}, [\![\mathsf{rear}]\!]_{\mathsf{ck}_2}, [\![\mathbf{s}]\!]_{\mathsf{ck}_5}, [\![\mathbf{a}]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}, \\ [\![\mathbb{x}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}, \\ [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}. \end{cases}$$

Recall, from Section 5.1, that $\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_5, \mathsf{cke})$ and $\mathsf{tck}' = (\mathsf{ck}'_1, \mathsf{ck}'_2, \mathsf{ck}'_3, \mathsf{cke}')$ where $\mathsf{ck}'_1 = \mathsf{ck}_2$ and $\mathsf{ck}'_2 = \mathsf{ck}_1$. Notice that cki, $\mathsf{ckm}_1, \ldots, \mathsf{ckm}_T$, $\mathsf{ck}_1, \ldots, \mathsf{ck}_5$, ck'_3 and cke' are generated independently. We now split the above constraints into the following sets of constraints:

- *Constraint Set 1.* $[\![\mathbb{x}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ involving commitment keys $\mathsf{ck}_1, \ldots, \mathsf{ck}_5$ and cke;
- *Constraint Set 2.* $[\![\mathbb{x}^\star]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}$ involving commitment keys ck_1, ck_2, ck'_3 and cke';
- *Constraint Set 3.*

$$\begin{cases} [\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}}, \\ [\![\mathsf{mul}_j]\!]_{\mathsf{ckm}_j} \in \mathcal{R}_{\mathsf{com}} \ \forall j \in [T], \\ [\![\mathsf{front}]\!]_{\mathsf{ck}_1} \in \mathcal{R}_{\mathsf{com}}, \\ [\![\mathsf{rear}]\!]_{\mathsf{ck}_2} \in \mathcal{R}_{\mathsf{com}}, \\ [\![\mathbf{s}]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}, \\ \overline{\mathsf{pc}} = 1 \wedge \overline{\mathsf{val}_{\mathsf{in}}} = \mathsf{front} \wedge \mathsf{val}_{\mathsf{out}} = \mathsf{rear}, \\ \mathsf{plkiv} = \sum_{j\in[T]} \mathsf{mul}_j \cdot \left(\chi + \left\langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k=0}^{n_{\mathsf{plk}}} \right\rangle\right)^{-1}, \\ \mathsf{miv} = \mathsf{miv}' \wedge \mathsf{macs}^\star = \mathsf{macs}' \end{cases}$$

  where $\sum_{j\in[T]} \mathsf{mul}_j \cdot \left(\chi + \left\langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k=0}^{n_{\mathsf{plk}}} \right\rangle\right)^{-1}$ can be computed in advance, and

$$\mathsf{front} = (\overline{\mathsf{pc}} \| \overline{\mathsf{val}} \| \ldots), \mathsf{rear} = (\mathsf{pc} \| \mathsf{macs} \| \mathsf{macs}'), \mathbf{s} = (\mathsf{plkiv} \| \mathsf{miv} \| \mathsf{miv}'),$$

  involving commitment keys cki, $\mathsf{ckm}_1, \ldots, \mathsf{ckm}_T$, ck_1, ck_2 and ck_5; (Notice that by Definition 9, the suffixes of front and rear w.r.t. $(\mathsf{macs} \| \mathsf{macs}')$ are the same due to $[\![\mathbb{x}^\star]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}$. We hence simply write $\mathsf{front} = (\overline{\mathsf{pc}} \| \overline{\mathsf{val}} \| \ldots)$ and only use $(\mathsf{macs} \| \mathsf{macs}')$ from rear.)
- *Constraint Set 4.* $[\![\mathbf{a}]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}$ involving ck_5.

Thus, with $L = 4$, we can split $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ into 4 split-relations w.r.t. those above constraints.

Efficiency. Notice that proving the above constraints can result in proof size $\mathcal{O}(\log n_{\mathsf{plk}} + \log\log|\mathbb{F}|) = \mathcal{O}(\log n_{\mathsf{plk}})$ since n_{plk} dominates in the lengths of involved vectors and $\log|\mathbb{F}|$ involves those vectors for proving memory consistency (see Remark 10). The prover time and verifier time are both $\mathcal{O}(n_{\mathsf{plk}} + \log|\mathbb{F}|)$. See Appendix B.12 for the proof size and running time.

Instantiating Π_{RP} (Figure 7) with compressed Σ-protocol from [AC20], we achieve the following efficiency for two settings, namely, (i) sequential and (ii) parallel with N threads:

- *Public Input Size.* $\mathcal{O}(1)$ for both settings. For the parallel setting with N threads, the communication cost for each thread can be $\mathcal{O}(T/N + \log N + \log n_{\mathsf{plk}} + \log\log|\mathbb{F}|)$.
- *Communication Cost.* $\mathcal{O}(T + N + \log n_{\mathsf{plk}} + \log\log|\mathbb{F}|)$ in total for both settings.
- *Prover Time.* $\mathcal{O}(T + N \cdot (n_{\mathsf{plk}} + \log|\mathbb{F}|))$ for (i) and $\mathcal{O}(T/N + \log N \cdot (n_{\mathsf{plk}} + \log|\mathbb{F}|))$ for (ii).
- *Verifier Time.* $\mathcal{O}(T + N + n_{\mathsf{plk}} + \log|\mathbb{F}|)$ for (i) and $\mathcal{O}(T/N + \log N + n_{\mathsf{plk}} + \log|\mathbb{F}|)$ for (ii).

We achieve the above efficiency because the employed Pedersen commitment [Ped92] is $\mathcal{O}(1)$ and homomorphically processing commitments (for any vector of any length) takes $\mathcal{O}(1)$ time. Note that the prover still needs to homomorphically evaluate the witnesses behind the commitments. See Appendix 7.1 for more details.

Acknowledgments

The work of Khai Hanh Tang was supported by Singapore Ministry of Education Academic Research Fund Tier 2 Grant MOE2019-T2-2-083. The work of Minh Pham was supported by Ethereum Foundation Ecosystem Support Program (Grant ID FY23-1101).

References

AC20. Thomas Attema and Ronald Cramer. Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, volume 12172 of *Lecture Notes in Computer Science*, pages 513–543. Springer International Publishing, 2020.

ACK21. Thomas Attema, Ronald Cramer, and Lisa Kohl. A Compressed Σ-Protocol Theory for Lattices. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, volume 12826 of *Lecture Notes in Computer Science*, pages 549–579. Springer International Publishing, 2021.

AS24. Arasu Arun and Srinath Setty. Nebula: Efficient read-write memory and switchboard circuits for folding schemes. Cryptology ePrint Archive, Paper 2024/1605, 2024.

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy – S&P 2018*, pages 315–334. IEEE, 2018.

BC23. Benedikt Bünz and Binyi Chen. Protostar: Generic Efficient Accumulation/Folding for Special-Sound Protocols. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, volume 14439 of *Lecture Notes in Computer Science*, pages 77–110. Springer Nature Singapore, 2023.

BC24a. Dan Boneh and Binyi Chen. LatticeFold: A Lattice-based Folding Scheme and its Applications to Succinct Proof Systems. Cryptology ePrint Archive, Paper 2024/257, 2024. `https://eprint.iacr.org/2024/257`.

BC24b. Benedikt Bünz and Jessica Chen. Proofs for Deep Thought: Accumulation for large memories and deterministic computations. Cryptology ePrint Archive, Paper 2024/325, 2024. `https://eprint.iacr.org/2024/325`.

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 *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference – ITCS 2012*, pages 326–349. Association for Computing Machinery, 2012.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing – STOC 2013*, pages 111–120. Association for Computing Machinery, 2013.

BCG+17. Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-Time Zero-Knowledge Proofs for Arithmetic Circuit Satisfiability. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, volume 10626 of *Lecture Notes in Computer Science*, pages 336–365. Springer International Publishing, 2017.

BCG+18. Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Arya: Nearly Linear-Time Zero-Knowledge Proofs for Correct Program Execution. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626. Springer International Publishing, 2018.

BCG24. Annalisa Barbara, Alessandro Chiesa, and Ziyi Guan. Relativized succinct arguments in the rom do not exist. Cryptology ePrint Archive, Paper 2024/728, 2024. `https://eprint.iacr.org/2024/728`.

BCL+21. Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-Carrying Data Without Succinct Arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710. Springer International Publishing, 2021.

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive Proof Composition from Accumulation Schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography – TCC 2020*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18. Springer International Publishing, 2020.

BFR+13. Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles – SOSP 2013*, pages 341–357. Association for Computing Machinery, 2013.

BMM+21. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for Inner Pairing Products and Applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, volume 13092 of *Lecture Notes in Computer Science*, pages 65–97. Springer International Publishing, 2021.

BP04. Mihir Bellare and Adriana Palacio. The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 273–289. Springer Berlin Heidelberg, 2004.

BS08. Eli Ben-Sasson and Madhu Sudan. Short pcps with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, 2008.

BSBC+17. Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational Integrity with a Public Random String from Quasi-Linear PCPs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, volume 10212 of *Lecture Notes in Computer Science*, pages 551–579. Springer International Publishing, 2017.

BSBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable Zero Knowledge with No Trusted Setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, volume 11694 of *Lecture Notes in Computer Science*, pages 701–732. Springer International Publishing, 2019.

BSCG+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 Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer Berlin Heidelberg, 2013.

BSCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent Succinct Arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer International Publishing, 2019.

BSCTV14a. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable Zero Knowledge via Cycles of Elliptic Curves. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 276–294. Springer Berlin Heidelberg, 2014.

BSCTV14b. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *23rd USENIX Security Symposium – USENIX Security 2014*, pages 781–796. USENIX Association, 2014.

CDP12. Ronald Cramer, Ivan Damgård, and Valerio Pastro. On the amortized complexity of zero knowledge protocols for multiplicative relations. In Adam Smith, editor, *Information Theoretic Security – ICITS 2012*, volume 7412 of *Lecture Notes in Computer Science*, pages 62–79. Springer Berlin Heidelberg, 2012.

CGG+24. Arka Rai Choudhuri, Sanjam Garg, Aarushi Goel, Sruthi Sekar, and Rohit Sinha. SublonK: Sublinear Prover PlonK. In *Proceedings on Privacy Enhancing Technologies – PETS 2024*, volume 2024, pages 314–335, 2024.

CHM+20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer International Publishing, 2020.

CJS14. Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a Global Random Oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security – CCS 2014*, page 597–608. Association for Computing Machinery, 2014.

DdSGOTV22. Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, Titouan Tanguy, and Michiel Verbauwhede. Efficient Proof of RAM Programs from Any Public-Coin Zero-Knowledge System. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks – SCN 2022*, volume 13409 of *Lecture Notes in Computer Science*, pages 615–638. Springer International Publishing, 2022.

DGP+24. Moumita Dutta, Chaya Ganesh, Sikhar Patranabis, Shubh Prakash, and Nitin Singh. Batching-efficient RAM using updatable lookup arguments. Cryptology ePrint Archive, Paper 2024/840, 2024.

DXNT23. Zijing Di, Lucas Xia, Wilson Nguyen, and Nirvan Tyagi. MUXProofs: Succinct Arguments for Machine Computation from Tuple Lookups. Cryptology ePrint Archive, Paper 2023/974, 2023. `https://eprint.iacr.org/2023/974`.

EFKP20. Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. SPARKs: Succinct Parallelizable Arguments of Knowledge. In Anne Canteaut and Yuval Ishai, editors, *Ad-*

vances in Cryptology – EUROCRYPT 2020, volume 12826 of *Lecture Notes in Computer Science*, pages 707–737. Springer International Publishing, 2020.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The Algebraic Group Model and its Applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62. Springer International Publishing, 2018.

FKL+21. Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. Constant-Overhead Zero-Knowledge for RAM Programs. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security – CCS 2021*, pages 178–191. Association for Computing Machinery, 2021.

FS87. Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer Berlin Heidelberg, 1987.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic Span Programs and Succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer Berlin Heidelberg, 2013.

GHAK23. Aarushi Goel, Mathias Hall-Andersen, and Gabriel Kaptchuk. Dora: Processor Expressiveness is (Nearly) Free in Zero-Knowledge for RAM Programs. Cryptology ePrint Archive, Paper 2023/1749, 2023. `https://eprint.iacr.org/2023/1749`.

Gro10. Jens Groth. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340. Springer Berlin Heidelberg, 2010.

Gro16. Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer Berlin Heidelberg, 2016.

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

H+. Daira-Emma Hopwood et al. Pasta curves. `https://github.com/zcash/pasta`.

Hab22. Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530, 2022. `https://eprint.iacr.org/2022/1530`.

HAN23. Mathias Hall-Andersen and Jesper Buus Nielsen. On Valiant's Conjecture. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, volume 14005 of *Lecture Notes in Computer Science*, pages 438–469. Springer Nature Switzerland, 2023.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing – STOC 2007*, pages 21–30. Association for Computing Machinery, 2007.

IOS23. Yuval Ishai, Rafail Ostrovsky, and Akash Shah. Succinct Arguments for RAM Programs via Projection Codes. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, volume 14082 of *Lecture Notes in Computer Science*, pages 159–192. Springer Nature Switzerland, 2023.

JJ24. Jehyuk Jang and Jamie Judd. An efficient SNARK for field-programmable and RAM circuits. Cryptology ePrint Archive, Paper 2024/507, 2024.

KS22. Abhiram Kothapalli and Srinath Setty. SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive, Report 2022/1758, 2022. `https://eprint.iacr.org/2022/1758`.

KS23a. Abhiram Kothapalli and Srinath Setty. CycleFold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Paper 2023/1192, 2023. `https://eprint.iacr.org/2023/1192`.

KS23b. Abhiram Kothapalli and Srinath Setty. HyperNova: Recursive arguments for customizable constraint systems. Cryptology ePrint Archive, Paper 2023/573, 2023. `https://eprint.iacr.org/2023/573`.

KST22. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388. Springer Nature Switzerland, 2022.

LFKN92. Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992.

LGZ+23. Xun Liu, Shang Gao, Tianyu Zheng, Yu Guo, and Bin Xiao. SnarkFold: Efficient Proof Aggregation from Incrementally Verifiable Computation and Applications. Cryptology ePrint Archive, Paper 2023/1946, 2023.

LXZ+24. Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkRollups via Fully Distributed Zero-Knowledge Proofs. In *2024 IEEE Symposium on Security and Privacy - S&P 2024*, pages 1777–1793. IEEE Computer Society, 2024.

MAGABMMT23. Héctor Masip-Ardevol, Marc Guzmán-Albiol, Jordi Baylina-Melé, and Jose Luis Muñoz-Tapia. eSTARK: Extending STARKs with Arguments. Cryptology ePrint Archive, Paper 2023/474, 2023.

MRS17. Payman Mohassel, Mike Rosulek, and Alessandra Scafuro. Sublinear Zero-Knowledge Arguments for RAM Programs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 501–531. Springer International Publishing, 2017.

NDC+24. Wilson Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A Scalable Framework for Folding-Based SNARKs. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, volume 14929 of *Lecture Notes in Computer Science*, pages 308–344. Springer Nature Switzerland, 2024.

NTWZ19. Khoa Nguyen, Hanh Tang, Huaxiong Wang, and Neng Zeng. New Code-Based Privacy-Preserving Cryptographic Constructions. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, volume 11922 of *Lecture Notes in Computer Science*, pages 25–55. Springer International Publishing, 2019.

Ped92. Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Berlin Heidelberg, 1992.

Sch80. J. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM*, 27(4):701–717, 1980.

Val76. Leslie G. Valiant. Universal circuits (Preliminary Report). In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing – STOC 1976*, pages 196–203. Association for Computing Machinery, 1976.

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 *Network and Distributed System Security Symposium – NDSS 2015*. The Internet Society, 2015.

WYKW21. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits. In *2021 IEEE Symposium on Security and Privacy – S&P 2021*, pages 1074–1091. IEEE, 2021.

WZC+18. Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A Distributed Zero Knowledge Proof System. In *27th USENIX Security Symposium – USENIX Security 2018*, pages 675–692. USENIX Association, 2018.

XZC+22. Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless Cross-chain Bridges Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security – CCS 2022*, pages 3003–3017. Association for Computing Machinery, 2022.

XZZ+19. Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation, 2019.

YH24. Yibin Yang and David Heath. Two Shuffles Make a RAM: Improved Constant Overhead Zero Knowledge RAM. In *33rd USENIX Security Symposium – USENIX Security 2024*, pages 1435–1452. USENIX Association, 2024.

ZGGX23. Tianyu Zheng, Shang Gao, Yu Guo, and Bin Xiao. KiloNova: Non-Uniform PCD with Zero-Knowledge Property from Generic Folding Schemes. Cryptology ePrint Archive, Paper 2023/1579, 2023. `https://eprint.iacr.org/2023/1579`.

ZGK+18. Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster Verifiable RAM with Program-Independent Preprocessing. In *2018 IEEE Symposium on Security and Privacy – S&P 2018*, pages 908–925. IEEE, 2018.

Zip79. Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation – EUROSAM 1979*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer Berlin Heidelberg, 1979.

ZXZS20. Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy – S&P 2020*, pages 859–876. IEEE, 2020.

A Related Work (Extended)

We provide a more detailed comparison of our work and several related works from Section 1.2.

Distributed ZKP. Recall that, by employing an existing distributed ZKP protocol in the literature on input a RAM program execution trace and viewing each prover as a thread, we could trivially get a somewhat parallelizable ZKP protocol for RAM programs. Hence, it would be necessary to compare our result with other distributed ZKP protocols regarding efficiency and minimal assumption required. To the best of our knowledge, there are only 3 existing distributed ZKP protocols, namely DIZK [WZC+18], deVirgo [XZC+22] and Pianist [LXZ+24] by proposing distributed versions of an existing SNARK construction [Gro16, ZXZS20, GWC19].

We now compare the efficiency and minimal assumptions of our work and theirs. For communication complexity, if a succinct commitment scheme like Pedersen is used, then our communication cost is $\mathcal{O}(N)$, while DIZK's cost is $\mathcal{O}(|W| \cdot N)$ and deVirgo, Pianist's cost are $\mathcal{O}(N)$. For proving time, their total proving time is equal to the cost of a single prover, which ranges from $\mathcal{O}(|W| \log |W|)$ to $\mathcal{O}(|W| \log^2 |W|)$, while ours is $\mathcal{O}(|W| \log N)$. For verification time, all of them incur $\mathcal{O}(1)$ verification time, while ours depends on the ZKAoK instantiation. Hence, their constructions could be more efficient than ours when $N \gg |W|$. However, to achieve such efficiency, DIZK and Pianist have to rely on Groth16 and PLONK, respectively, which require *non-standard assumptions* for security such as AGM and trusted setup, while deVirgo uses sumcheck argument and thus, could only rely on SNARKs for sumcheck-type constraints like Virgo. This makes all these constructions have only limited choice for ZKP instantiations. On the other hand, our construction requires the relatively standard and minimal assumption, **it only requires the existence of a homomorphic commitment scheme** and can be instantiated from any ZKP protocol with that requirement. Finally, while these protocols claim that they could achieve zero knowledge, there is no detailed method so far in each construction nor a formal proof for this.

Dora. Dora [GHAK23] also leverages the folding scheme to propose a ZKP for RAM programs. However, as of ours, their construction does not follow the IVC approach to achieve succinct proof size but instead designs an interactive proof system that aims to achieve linear communication cost and prover time like [FKL+21, YH24]. In addition, their construction also requires minimal assumption since only a homomorphic commitment scheme is needed, making it compatible with many ZKP paradigms. Due to these similarities, it would be straightforward to compare their approach against us. Unlike our construction based on CF, Dora proposed *ZK-bag*, a novel primitive that allows the prover to insert (send)/retrieve items to/from the bag such that

- The retrieved items do not reveal when it has been sent.
- The prover cannot retrieve items not in the bag.
- The prover can convince the verifier that the bag is empty.

Using ZK-bag, handling memory consistency in Dora is straightforward as follows. Initially, the prover inserts all the tuples $([\![\ell]\!], [\![\mathsf{val}]\!])$ of **every** memory cells to the ZK-bag. In each step, suppose we need to read/write the value val' from/to address ℓ, the prover retrieves the current tuple $([\![\ell]\!], [\![\mathsf{val}]\!])$ from the ZK-bag and "remove" it, then insert the tuple $([\![\ell]\!], [\![\mathsf{val}']\!])$ into the bag. Finally, the prover proves that the bag is empty, implying that the inserted and "removed" tuples are permutations of each other. For handling the correctness of instructions, suppose there are T instructions, and each instance-witness pair of a single computation step has the form of an R1CS instance-witness pair. The prover initializes T accumulators $([\![z'_i]\!])_{i=1}^T$, which represents T instructions. In each step, if the instruction F_j is executed, then the value $\mathbb{z}'_j$ will first be retrieved from the bag, then updated by folding with the R1CS instance-witness pair of the current step, and finally be sent to the bag again. Finally, the prover and verifier engage in a ZKAoK for checking the validity of $[\![z'_i]\!]_{i=1}^T$, which implies the correctness of the whole execution process. However, in this way, Dora's construction does not support generating proofs in parallel. Indeed, in each step, the prover has to "remove" the instance-witness pair of the current step from the ZK-bag and then insert a new one into it. These operations can only be performed sequentially during the whole N steps, and we are unaware of any method to parallize it.

A Note on SPARK. SPARK [EFKP20] is also a parralizable succinct argument for RAM program. However, the construction is not zero-knowledge because the construction requires revealing the hash digest of the memory in intermediate steps and then later using a SNARK to prove the correctness of the hash. For the hash function to be modeled as a circuit and proved by a SNARK,

it must be a standard hash function, not modeled as a random oracle. Since the hash function is not known to be hiding, SPARK is not believed to be zero-knowledge. Also, SPARK's parallization method differs from ours: SPARK's method is to have k threads, and divides a N-step RAM program into k sub-programs, where the i-th thread executes the i-th sub-program and prove the correctness of it. The computation time of SPARK is $WN/k + O(\log(WN))$, as analyzed by the authors (since one of the threads has to run WN/k computation steps). On the other hand, our construction runs the program sequentially and records the witness, then uses the threads to provide the proof in parallel using the binary tree-like folding method, which requires $O(T + W \log N)$ proof generation time in parallel.

B Preliminaries (Extended)

We recall the necessary preliminaries in complement to Section 3.

B.1 RAM Program

We model a RAM program as a combination of the Non-Uniform Incremental Computation [KS22] and RAM Program [FKL+21]. This model of computation contains a memory mem, a program counter pc and an instruction set $\mathcal{F}'$ of cardinality T described as follows.

- The memory mem can be viewed as a sequence of M elements, i.e., $\mathsf{mem} = (\mathsf{mem}_i)_{i\in[M]}$. For each $i \in [M]$, mem_i belongs to the set $\mathbb{F} \cup \{\perp\}$ where $\perp$ is understood to be an uninitialized value which cannot be read by the instructions. At the beginning, every mem_i is set to be $\perp$.
- The instruction set $\mathcal{F}'$ is a set of T instructions containing $F_1, \ldots, F_T$. There exists a program counter $\mathsf{pc} \in [T]$ that determines the next instruction F_{pc} to be executed. Initially, pc is set to be 1 and the RAM program receives as input a value val. For each step of the RAM program, it determines the instruction F_{pc} and runs F_{pc} on input val. F_{pc} then returns a new tuple $(\mathsf{pc}', \mathsf{val}', \ell, \mathsf{val}', \mathsf{mop})$ containing a new program counter pc', new value val', an index $\ell \in [M]$, new value val' and a memory access operation $\mathsf{mop} \in \{\text{WRITE}, \text{READ}\}$. Then, it updates the state of the system as follows:
  - Set $\mathsf{val} := \mathsf{val}'$ and $\mathsf{pc} := \mathsf{pc}'$. Here, pc is set to be the new value pc' to determine the instruction in the next step.
  - If $\mathsf{mop} = \text{WRITE}$, set $\mathsf{mem}_\ell := \mathsf{val}'$ and $\mathsf{val} := \mathsf{val}'$. Otherwise, if $\mathsf{mop} = \text{READ}$, set $\mathsf{val} := \mathsf{mem}_\ell$.

B.2 Memory Consistency Check

We recall the technique for checking memory consistency in [BCG+18,ZGK+18,FKL+21]. Roughly speaking, let $N \in \mathbb{Z}_+$, for each $i \in [N]$, the i-th memory access is represented by a tuple

$$\mathsf{macs}_i = (\ell_i, \mathsf{time}_i, \mathsf{val}_i, \mathsf{mop}_i) \in \mathbb{F}^4,$$

where ℓ_i is the index of the accessed memory cell mem_{ℓ_i}, time_i is the time logged for this access, val_i is the access value and mop_i is either READ or WRITE

A sequence of memory access $(\mathsf{macs}_i)_{i\in[N]}$ is valid if, for each memory cell, over time, the first access is of type WRITE and the value val achieved from any READ access must be equal to the previous value read from or written to the same cell. To capture the above condition, [FKL+21] show that there exists a sequence $(\mathsf{macs}'_i)_{i\in[N]}$ s.t.

- $(\mathsf{macs}'_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}_i)_{i\in[N]}$.
- $(\mathsf{macs}'_i)_{i\in[N]}$ is well-sorted (sorted via address and time-log). In addition, $(\mathsf{macs}'_i)_{i\in[N]}$ must satisfies the below conditions:
- If $\ell'_i = \ell'_{i+1}$ and $\mathsf{mop}'_i = 0$ then $\mathsf{val}'_i = \mathsf{val}'_{i+1}$ and,
- The first access to each memory cell must be equal to WRITE.

Let us explain the meaning of the third constraint. Because $(\mathsf{macs}'_i)_{i\in[N]}$ is sorted by address, we can easily check that: If $\ell = \ell'_i = \ell'_{i+1}$, then time'_i is the latest previous time we have accessed mem_ℓ and therefore it must holds that $\mathsf{val}'_i = \mathsf{val}'_{i+1}$ if we read from mem_ℓ at time time_{i+1} (i.e., $\mathsf{mop}'_i = 0$). This is also sufficient to capture the consistency of reading values from the memory at any time. Now, to verify the second, third, and fourth constraints, we observe that:

– Verifying the well-sorted property of the sequence $(\mathsf{macs}'_i)_{i\in[N]}$ requires checking $\big(\ell'_{i-1} < \ell'_i\big) \vee \big((\ell'_{i-1} = \ell'_i) \wedge (\mathsf{time}'_{i-1} < \mathsf{time}'_i)\big)\ \forall\ i \in [2, N]$. This implies that $(\mathsf{macs}'_i)_{i\in[N]}$ is sorted via address first, then time-log.
– Verifying the consistency of reading between two steps can be captured by checking that $(\ell'_{i-1} \neq \ell'_i) \vee (\mathsf{val}'_{i-1} = \mathsf{val}'_i) \vee (\mathsf{mop}'_i = 0)\ \forall\ i \in [2, N]$..
– We also require all the first access of the cell must be equal to WRITE. This can be done by checking $(\ell'_{i-1} = \ell'_i) \vee (\mathsf{mop}'_i = 0)\ \forall\ i \in [2, N]$, and $(\mathsf{macs}_{i-1} = 0) \vee (i - 1 > 1)\ \forall\ i \in [2, N]$.
– We also need to check the validity of the address and operations. This can be done by checking $1 \leq \ell_i \leq M \wedge \mathsf{mop}_i \in \{0, 1\}\ \forall\ i \in [N]$ and $\mathsf{time}_{i-1} < \mathsf{time}_i\ \forall\ i \in [2, N]$.
– Finally, we need the original sequence $(\mathsf{macs}_i)_{i\in[N]}$ represents the process of accessing the memory *during the course of time*. This can be done by checking $\mathsf{time}_{i-1} < \mathsf{time}_i$ for all $i \in [2, N]$.

In summary, to prove memory consistency, it suffices to compute a sequence $(\mathsf{macs}'_i)_{i\in[N]}$ where $\mathsf{macs}'_i = (\ell'_i, \mathsf{time}'_i, \mathsf{val}'_i, \mathsf{mop}'_i)$, and show that

$$(\mathsf{macs}'_i)_{i\in[N]} \text{ is a permutation of } (\mathsf{macs}_i)_{i\in[N]} \tag{33}$$

and

$$\begin{cases} 1 \leq \ell_i \leq M \wedge \mathsf{mop}_i \in \{0, 1\} & \forall i \in [N], \\ \mathsf{time}_{i-1} < \mathsf{time}_i & \forall i \in [2, N], \\ \big(\ell'_{i-1} < \ell'_i\big) \vee \big((\ell'_{i-1} = \ell'_i) \wedge (\mathsf{time}'_{i-1} < \mathsf{time}'_i)\big) & \forall i \in [2, N], \\ (\mathsf{macs}_{i-1} = 1) \vee (i - 1 > 1) & \forall i \in [2, N], \\ (\ell'_{i-1} = \ell'_i) \vee (\mathsf{mop}'_i = 1) & \forall i \in [2, N], \\ (\ell'_{i-1} \neq \ell'_i) \vee (\mathsf{val}'_{i-1} = \mathsf{val}'_i) \vee (\mathsf{mop}'_i = 1) & \forall i \in [2, N]. \end{cases} \tag{34}$$

We note that the first line of (34) has index $i \in [N]$ while, in the remaining lines, $i \in [2, N]$. Our purpose is to use this system to guarantee the condition between memory accesses at steps $i - 1$ and i. Therefore, we transform this condition into

$$\begin{cases} 1 \leq \ell_i \leq M \wedge \mathsf{mop}_i \in \{0, 1\} & \forall i \in [2, N], \\ 1 \leq \ell_{i-1} \leq M \wedge \mathsf{mop}_{i-1} \in \{0, 1\} & \forall i \in [2, N], \\ \mathsf{time}_{i-1} < \mathsf{time}_i & \forall i \in [2, N], \\ \big(\ell'_{i-1} < \ell'_i\big) \vee \big((\ell'_{i-1} = \ell'_i) \wedge (\mathsf{time}'_{i-1} < \mathsf{time}'_i)\big) & \forall i \in [2, N], \\ (\mathsf{macs}_{i-1} = 1) \vee (i - 1 > 1) & \forall i \in [2, N], \\ (\ell'_{i-1} = \ell'_i) \vee (\mathsf{mop}'_i = 1) & \forall i \in [2, N], \\ (\ell'_{i-1} \neq \ell'_i) \vee (\mathsf{val}'_{i-1} = \mathsf{val}'_i) \vee (\mathsf{mop}'_i = 1) & \forall i \in [2, N] \end{cases} \tag{35}$$

s.t. the first line in (34) is equivalently split into the first two lines in (35).

In summary, there is a circuit $\mathsf{C}_{\mathsf{mem}}$ defined as follows. On inputs i, macs_{i-1}, macs'_{i-1}, macs_i and macs'_i,

$$\mathsf{C}_{\mathsf{mem}}(i, \mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1}, \mathsf{macs}_i, \mathsf{macs}'_i) \in \{0, 1\},$$

for all $i \in [2, N]$, s.t., $\mathsf{C}_{\mathsf{mem}}$ returns 1 iff the constraints of (35) are satisfied. Here, we need index i in the parameters of $\mathsf{C}_{\mathsf{mem}}$ since we need to check whether $i-1 > i$ as in the 4-th line of system (35). Thus, (33) and $\mathsf{C}_{\mathsf{mem}}$ together capture the memory consistency of sequence $(\mathsf{macs}_i)_{i\in[N]}$.

Remark 10. In our construction of RAMenPaSTA, we realize $\mathsf{C}_{\mathsf{mem}}$ as R1CS matrices $\mathbf{A}', \mathbf{B}'$ and $\mathbf{C}'$ (see Definition 9 excluding those for checking equality). It is well-known that the size of R1CS matrices only depends on the multiplication gates of the circuit. Hence, we count the number of multiplication gates in $\mathsf{C}_{\mathsf{mem}}$ as follows.

In (35), handling the three final constraints requires $\mathcal{O}(1)$ multiplication gates. Finally, handling the first four constraints requires proving that $a < b$ for two values $a, b \ll \mathbb{F}$. It is well-known that such handling $a < b$ when $a, b \in \mathbb{F}$ requires $\mathcal{O}(\log|\mathbb{F}|)$ multiplication gates (e.g., see [NTWZ19, Theorem 2]). Therefore, handling the first four constraints requires $\mathcal{O}(\log|\mathbb{F}|)$ multiplication gates. Thus, we conclude that the circuit $\mathsf{C}_{\mathsf{mem}}$ has $\mathcal{O}(\log|\mathbb{F}|)$ multiplication gates. Consequently, we can see that $\mathbf{A}', \mathbf{B}'$, and $\mathbf{C}'$ have $\mathcal{O}(\log|\mathbb{F}|)$ rows while each row has a constant number of non-zero entries.

B.3 PLONK's Arithmetization

We recall PLONK's arithmetization [GWC19] for representing a circuit in the form of *gate constraints* and *copy constraints*. Let C be an arithmetic circuit representing the computation of a function $F : \mathbb{F}^{n_{\mathsf{in}}} \to \mathbb{F}^{n_{\mathsf{out}}}$ that maps an input vector $\mathbf{x} = (x_1, \ldots, x_{n_{\mathsf{in}}}) \in \mathbb{F}^{n_{\mathsf{in}}}$ to an output vector $\mathbf{y} = (y_1, \ldots, y_{n_{\mathsf{out}}}) \in \mathbb{F}^{n_{\mathsf{out}}}$. Each gate in C is of the following four types:

- *Addition.* An addition gate takes as inputs $a, b \in \mathbb{F}$ and returns $a + b \in \mathbb{F}$.
- *Addition with Constant d.* This gate takes as input $a \in \mathbb{F}$ and return $a + d \in \mathbb{F}$.
- *Multiplication.* This gate takes as inputs $a, b \in \mathbb{F}$ and returns $a \cdot b \in \mathbb{F}$.
- *Multiplication with Constant d.* This gate takes as input $a \in \mathbb{F}$ and returns $a \cdot d \in \mathbb{F}$.

We denote by n_{gate} as the total number of gates in C. Hence, by indexing each gate of C to be a number in $[n_{\mathsf{gate}}]$, we denote by a_i, b_i, c_i to be the values on the left, right, and output wires, respectively, of the i-th gate. For addition and multiplication with constant, we assume that b_i can be any value in $\mathbb{F}$ since it does not affect the computation following the structure of circuit C. Moreover, the i-th gate is associated with the selectors $s_i^{\mathsf{left}}, s_i^{\mathsf{right}}, s_i^{\mathsf{mul}}, s_i^{\mathsf{const}}$ s.t. the relation between a_i, b_i and c_i is captured by the equation

$$s_i^{\mathsf{left}} \cdot a_i + s_i^{\mathsf{right}} \cdot b_i + s_i^{\mathsf{mul}} \cdot (a_i \cdot b_i) + s_i^{\mathsf{const}} - c_i = 0. \tag{36}$$

Each equation in the form of (36) is a *gate constraint*. Hence, we define a witness satisfying circuit C to be

$$\begin{aligned} \mathbf{w}_{\mathsf{plk}} &= (x_1, \ldots, x_{n_{\mathsf{in}}}, y_1, \ldots, y_{n_{\mathsf{out}}}, a_1, \ldots, a_{n_{\mathsf{gate}}}, b_1, \ldots, b_{n_{\mathsf{gate}}}, c_1, \ldots, c_{n_{\mathsf{gate}}}) \\ &= (w_1, \ldots, w_{n_{\mathsf{wit}}}) \in \mathbb{F}^{n_{\mathsf{wit}}} \end{aligned} \tag{37}$$

where $n_{\mathsf{wit}} = n_{\mathsf{in}} + n_{\mathsf{out}} + 3n_{\mathsf{gate}}$ and $c_{n_{\mathsf{gate}}}$ is value of output wire.

In addition, we would require constraints to ensure that the wires are connected. For example, in some circuits, we would require that the output of the first wire is equal to the left input of the second wire, which can be captured by the constraint $c_1 = a_2$. We name these constraints *copy constraint*. To guarantee the connection between wires, namely, copy constraint, there exists a public permutation $\varphi : [n_{\mathsf{wit}}] \to [n_{\mathsf{wit}}]$ based on C s.t. the copy constraint is satisfied iff $((1, w_1), \ldots, (n_{\mathsf{wit}}, w_{n_{\mathsf{wit}}}))$ is a permutation of $((\varphi(1), w_1), \ldots, (\varphi(n_{\mathsf{wit}}), w_{n_{\mathsf{wit}}}))$. According to [GWC19], for value $\gamma, \delta \xleftarrow{\$} \mathbb{F}$, if

$$\prod_{i=1}^{n_{\mathsf{wit}}} (\gamma + i \cdot \delta + w_i) = \prod_{i=1}^{n_{\mathsf{wit}}} (\gamma + \varphi(i) \cdot \delta + w_i) \tag{38}$$

holds, then it would imply that $((1, w_1), \ldots, (n_{\mathsf{wit}}, w_{n_{\mathsf{wit}}}))$ is a permutation of $((\varphi(1), w_1), \ldots, (\varphi(n_{\mathsf{wit}}), w_{n_{\mathsf{wit}}}))$ with probability at least $1 - \frac{n_{\mathsf{wit}}}{|\mathbb{F}|}$ by Schwartz-Zippel lemma [Zip79, Sch80].

In summary, the structure of circuit C can be compactly represented by the PLONK structure

$$\begin{aligned} \mathsf{plkst} = (&s_1^{\mathsf{left}}, s_1^{\mathsf{right}}, s_1^{\mathsf{mul}}, s_1^{\mathsf{const}}, \ldots, s_{n_{\mathsf{gate}}}^{\mathsf{left}}, s_{n_{\mathsf{gate}}}^{\mathsf{right}}, s_{n_{\mathsf{gate}}}^{\mathsf{mul}}, s_{n_{\mathsf{gate}}}^{\mathsf{const}}, \\ &\varphi(1), \ldots, \varphi(n_{\mathsf{wit}})) \in \mathbb{F}^{n_{\mathsf{plk}}} \end{aligned} \tag{39}$$

where

$$n_{\mathsf{plk}} = 4n_{\mathsf{gate}} + n_{\mathsf{wit}} = n_{\mathsf{in}} + n_{\mathsf{out}} + 7n_{\mathsf{gate}}. \tag{40}$$

We assume that $n_{\mathsf{plk}} = \mathcal{O}(n_{\mathsf{gate}})$ because $n_{\mathsf{wit}} = 4n_{\mathsf{gate}} + n_{\mathsf{in}} + n_{\mathsf{out}}$ with n_{in} and n_{out} are at most $\mathcal{O}(n_{\mathsf{gate}})$.

From (36) and , by sampling $\gamma, \delta \xleftarrow{\$} \mathbb{F}$, if the system

$$\begin{cases} s_i^{\mathsf{left}} \cdot a_i + s_i^{\mathsf{right}} \cdot b_i + s_i^{\mathsf{mul}} \cdot (a_i \cdot b_i) + s_i^{\mathsf{const}} - c_i = 0 \quad \forall i \in [n_{\mathsf{gate}}], \\ \prod_{i=1}^{n_{\mathsf{wit}}} (\gamma + i \cdot \delta + w_i) = \prod_{i=1}^{n_{\mathsf{wit}}} (\gamma + \varphi(i) \cdot \delta + w_i) \end{cases} \tag{41}$$

is satisfied, we see that $\mathbf{w}_{\mathsf{plk}}$ is a valid witness of C w.r.t. the compact PLONK structure plkst with probability at least $1 - \frac{n_{\mathsf{wit}}}{|\mathbb{F}|}$. Notice that (41) can be represented under the form of an R1CS constraint system with public matrices determined based on γ and δ. The witness vector for this

R1CS constraint system contains both PLONK structure plkst and witness vector $\mathbf{w}_{\mathsf{plk}}$ specified above.

Let plkst be the PLONK structure of a function F. Notice that the witness $\mathbf{w}_{\mathsf{plk}}$ in (37) contains 3 components: (i) input $\mathbf{x}$ to and (ii) output $\mathbf{y}$ from F, and (iii) middle values known as supporting witness $\mathbf{w}$. Hence, a valid witness $\mathbf{w}_{\mathsf{plk}}$ (for proving $F(\mathbf{x}) = \mathbf{y}$) can be parsed as $\mathbf{w}_{\mathsf{plk}} = (\mathbf{x}\|\mathbf{y}\|\mathbf{w})$. Then, system (41) implies the existence of a circuit $\mathsf{C}_{\mathsf{plk}}^{\gamma,\delta} : \mathbb{F}^{n_{\mathsf{plk}}+n_{\mathsf{wit}}} \to \{0,1\}$, parameterized by $\gamma, \delta \xleftarrow{\$} \mathbb{F}$, s.t.

- if $\mathbf{w}_{\mathsf{plk}}$ is a valid witness for plkst, then $\mathsf{C}_{\mathsf{plk}}^{\gamma,\delta}(\mathsf{plkst}, \mathbf{x}, \mathbf{y}, \mathbf{w}) = 1$.
- if $\mathbf{w}_{\mathsf{plk}}$ is not a valid witness corresponding for plkst, then $\mathsf{C}_{\mathsf{plk}}^{\gamma,\delta}(\mathsf{plkst}, \mathbf{x}, \mathbf{y}, \mathbf{w})$ returns 1 with probability at most $\mathcal{O}(n_{\mathsf{plk}}/|\mathbb{F}|)$, namely, the soundness error.

Remark 11. In our construction of RAMenPaSTA, we realize $\mathsf{C}_{\mathsf{plk}}$ as R1CS matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ (see Definition 8). It is well-known that the size of these R1CS matrices only depends on the multiplication gates of the circuit. As of Remark 10, we count the number of multiplication gates in $\mathsf{C}_{\mathsf{plk}}^{\gamma,\delta}$ as follows.

Each gate constraint requires 4 multiplication gates (see the first row of (41)). Hence, handling the gate constraints requires $4n_{\mathsf{gate}}$ multiplication gates. Finally, the two products in the second row of (41) can be decomposed into $3n_{\mathsf{wit}}$ multiplication gates. Hence, the number of multiplication gates in $\mathsf{C}_{\mathsf{plk}}^{\gamma,\delta}$ is equal to $4n_{\mathsf{gate}} + 3n_{\mathsf{wit}} = \mathcal{O}(n_{\mathsf{plk}})$ where $n_{\mathsf{plk}} = 4n_{\mathsf{gate}} + n_{\mathsf{wit}} = n_{\mathsf{in}} + n_{\mathsf{out}} + 7n_{\mathsf{gate}}$ as defined in (40).

B.4 Logarithmic Derivative Supporting Permutation and Lookup Arguments

Permutations. We recall the following lemma from [Hab22] for supporting checking permutation arguments.

Lemma 3 (Consequence of Lemma 3 of [Hab22]). *Let n be a positive integer. Let $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$ be sequence over a field $\mathbb{F}$ with characteristic $p > n$. Then $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^n$ are permutation of each other iff*

$$\sum_{i=1}^{n} \frac{1}{X + a_i} = \sum_{i=1}^{n} \frac{1}{X + b_i} \tag{42}$$

in the rational function field $\mathbb{F}(X)$.

Permutations of Sequences of Tuples. We adapt the above lemma to support permutations of sequences of tuples in the following sense. We say that $\mathbf{a} = (\mathbf{a}_i)_{i=1}^n \in (\mathbb{F}^s)^n$ is a permutation of $\mathbf{b} = (\mathbf{b}_i)_{i=1}^n \in (\mathbb{F}^s)^n$ for some positive integers s and n iff there exists some $\varphi \in \mathsf{S}_n$, where S_n is a symmetric group over $[n]$, satisfying $\mathbf{a}_i = \mathbf{b}_{\varphi(i)}$ for all $i \in [n]$. We have the following Lemma 4:

Lemma 4 (Permutations of Sequences of Tuples). *Given sequences $\mathbf{a} = (\mathbf{a}_i)_{i=1}^n, \mathbf{b} = (\mathbf{b}_i)_{i=1}^n \in (\mathbb{F}^s)^n$ where s and n are positive integers. Then, $\mathbf{a}$ is a permutation of of $\mathbf{b}$ iff*

$$\sum_{i=1}^{n} \frac{1}{X + \langle \mathbf{a}_i, (Y^k)_{k=0}^{s-1} \rangle} = \sum_{i=1}^{n} \frac{1}{X + \langle \mathbf{b}_i, (Y^k)_{k=0}^{s-1} \rangle} \tag{43}$$

in the rational function field $\mathbb{F}(X, Y)$ where X and Y are variables over $\mathbb{F}$.

Proof. If $(\mathbf{a}_i)_{i=1}^n$ is a permutation of $(\mathbf{b}_j)_{j=1}^n$ then (43) trivially holds.

We consider the other direction. Let $\mathbf{a}'_1, \ldots, \mathbf{a}'_u$ be distinct vectors satisfying $\{\mathbf{a}'_i\}_{i=1}^u = \{\mathbf{a}_i\}_{i=1}^n$ for some positive integer $u \leq n$ and consider $\mathsf{mul}_i = \sum_{j=1}^n (\mathbf{a}'_i = \mathbf{a}_j)$, for all $i \in [u]$. Similarly, let $\{\mathbf{b}'_i\}_{i=1}^v = \{\mathbf{b}_i\}_{i=1}^n$ for some positive integer $v \leq n$ and consider $\mathsf{mul}'_i = \sum_{j=1}^n (\mathbf{b}'_i = \mathbf{b}_j)$, for all $i \in [v]$. It suffices to prove the following:

$$\begin{cases} u = v, \\ \exists \sigma \in \mathsf{S}_u \text{ s.t. } \mathbf{a}'_i = \mathbf{b}'_{\sigma(i)} \wedge \mathsf{mul}_i = \mathsf{mul}'_{\sigma(i)} \ \forall\ i \in [u] \end{cases} \tag{44}$$

where S_u is the symmetric group over $[u]$.

For a vector $\mathbf{v} \in \mathbb{F}^s$, define $f_{\mathbf{v}}(Y) = \left\langle \mathbf{v}, (Y^k)_{k=0}^{s-1} \right\rangle$. Let

$$\begin{aligned} f(X,Y) &= \sum_{i=1}^{n} \frac{1}{X + f_{\mathbf{a}_i}(Y)} - \sum_{i=1}^{n} \frac{1}{X + f_{\mathbf{b}_i}(Y)} \\ &= \sum_{i=1}^{u} \frac{\mathsf{mul}_i}{X + f_{\mathbf{a}'_i}(Y)} - \sum_{j=1}^{v} \frac{\mathsf{mul}'_j}{X + f_{\mathbf{b}'_j}(Y)}. \end{aligned}$$

We can see that (43) holds iff $f(X,Y) = 0 \in \mathbb{F}(X,Y)$. Since we assume that (43) holds, we have $f(X,Y) = 0$. We define

$$g(X,Y) = f(X,Y) \cdot \prod_{i=1}^{u} (X + f_{\mathbf{a}'_i}(Y)) \cdot \prod_{i=1}^{v} (X + f_{\mathbf{b}'_i}(Y)).$$

Then it holds that $g(X,Y) = 0 \in \mathbb{F}(X,Y)$. We see that the explicit form of $g(X,Y)$ can be written to be

$$\begin{aligned} g(X,Y) = &\sum_{i=1}^{u} \mathsf{mul}_i \cdot \prod_{j \in [u] \text{ s.t. } j \neq i} (X + f_{\mathbf{a}'_j}(Y)) \cdot \prod_{j=1}^{v} (X + f_{\mathbf{b}'_j}(Y)) \\ &- \sum_{i=1}^{v} \mathsf{mul}'_i \cdot \prod_{j=1}^{u} (X + f_{\mathbf{a}'_j}(Y)) \cdot \prod_{j \in [v] \text{ s.t. } j \neq i} (X + f_{\mathbf{b}'_j}(Y)). \end{aligned}$$

For each $k \in [u]$ by replacing X by $-f_{\mathbf{a}'_k}(Y)$, we see that

$$g(-f_{\mathbf{a}'_k}(Y), Y) = \mathsf{mul}_k \cdot \prod_{j \in [u] \text{ s.t. } j \neq k} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{a}'_j}(Y)) \cdot \prod_{j=1}^{v} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{b}'_j}(Y)).$$

Since we assumed that $g(X,Y) = 0$, $\mathsf{mul}_k \neq 0$ and $\mathbf{a}'_1, \ldots, \mathbf{a}'_u$ are pairwise distinct, thus $f_{\mathbf{a}'_k}(Y) \neq f_{\mathbf{a}'_j}(Y)$ in $\mathbb{F}(X,Y)$ for any $j \neq k$ and thus

$$\mathsf{mul}_k \cdot \prod_{j \in [u] \text{ s.t. } j \neq k} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{a}'_j}(Y)) \neq 0.$$

Hence, it can be seen that $\prod_{j=1}^{v} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{b}'_j}(Y)) = 0 \in \mathbb{F}(X,Y)$ for each $k \in [u]$. This means that each $\mathbf{a}'_k(Y)$ is equal to $\mathbf{b}'_h(Y)$ for some $h \in [v]$ and each value k gives a distinct value h. Hence, $u \leq v$.

Similarly, by replacing X with $\mathbf{b}'_k(Y)$ for each $k \in [v]$, we see that each $\mathbf{b}'_k(Y)$ is equal to $\mathbf{a}'_h(Y)$ for some $h \in [u]$, each value k gives a distinct value h as well. Hence, $v \leq u$.

Therefore, for two equalities happen, we must have $u = v$ and there exists a permutation $\sigma \in \mathsf{S}_u$ s.t. $\mathbf{a}'_k = \mathbf{b}'_{\sigma(k)}$ for all $k \in [u]$.

Finally, we need to prove that $\mathsf{mul}_i = \mathsf{mul}'_{\sigma(i)}$ for all $i \in [u]$. We see that $f(X,Y)$ can now be written as

$$f(X,Y) = \sum_{i=1}^{u} \frac{\mathsf{mul}_i - \mathsf{mul}'_{\sigma(i)}}{X + f_{\mathbf{a}'_i}(Y)}.$$

We define

$$g'(X,Y) = f(X,Y) \cdot \prod_{i=1}^{u} (X + f_{\mathbf{a}'_i}(Y)).$$

Since we assumed that $f(X,Y) = 0$, it implies that $g'(X,Y) = 0 \in F[X,Y]$. For each $k \in [u]$, by letting $X = -f_{\mathbf{a}'_k}(Y))$, we see that

$$g'(-f_{\mathbf{a}'_k}(Y), Y) = (\mathsf{mul}_k - \mathsf{mul}'_{\sigma(k)}) \cdot \prod_{j \in [u] \text{ s.t. } j \neq k} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{a}'_j}(Y)).$$

Since we have assumed that $\mathbf{a}'_i \neq \mathbf{a}'_j$ for all $i, j \in [u]$ satisfying $i \neq j$ thus $f_{\mathbf{a}'_i}(Y) \neq f_{\mathbf{a}'_j}(Y)$, consequently we must have

$$\mathsf{mul}_k = \mathsf{mul}'_{\sigma(k)}.$$

Hence, it holds that $(\mathbf{a}_i)_{i=1}^n$ is indeed a permutation of $(\mathbf{b}_i)_{i=1}^n$, as desired, according to (44). $\square$

Testing Permutations of Sequences of Tuples. If $\mathbf{a}$ and $\mathbf{b}$ are not permutation of each other, then by sampling $(\tau, \omega) \xleftarrow{\$} \mathbb{F} \times \mathbb{F}$ and setting $X = \tau$ and $Y = \omega$, then (43) holds with probability at most

$$\mathsf{err}_{\mathsf{perm}}(\mathbb{F}, s, n) = \frac{(s-1)(2n-1)}{|\mathbb{F}|} = \mathcal{O}\left(\frac{s \cdot n}{|\mathbb{F}|}\right). \tag{45}$$

according the following Lemma 5.

Lemma 5 (Tuple Permutation Error Probability). *Given sequences $\mathbf{a} = (\mathbf{a}_i)_{i=1}^n \in (\mathbb{F}^s)^n$ and $\mathbf{b} = (\mathbf{b}_i)_{i=1}^n \in (\mathbb{F}^s)^n$ where s and n are positive integers. For a vector $\mathbf{v} \in \mathbb{F}^s$, define $f_{\mathbf{v}}(Y) = \left\langle \mathbf{v}, (Y^k)_{k=0}^{s-1} \right\rangle$. Assume that*

$$\sum_{i=1}^n \frac{1}{X + f_{\mathbf{a}_i}(Y)} \neq \sum_{i=1}^n \frac{1}{X + f_{\mathbf{b}_i}(Y)}.$$

Then,

$$\Pr\left[f(\tau, \omega) = 0 \middle| \tau \xleftarrow{\$} \mathbb{F} \wedge \omega \xleftarrow{\$} \mathbb{F}\right] \leq \frac{(s-1)(2n-1)}{|\mathbb{F}|}$$

where

$$f(X, Y) = \sum_{i=1}^n \frac{1}{X + f_{\mathbf{a}_i}(Y)} - \sum_{i=1}^n \frac{1}{X + f_{\mathbf{b}_i}(Y)}.$$

Proof. Define

$$\begin{aligned} g(X, Y) &= f(X, Y) \prod_{i=1}^n (X + f_{\mathbf{a}_i}(Y)) \prod_{i=1}^n (X + f_{\mathbf{b}_i}(Y)) \\ &= \sum_{i=1}^n \prod_{j \in [n] \text{ s.t. } j \neq i} (X + f_{\mathbf{a}_j}(Y)) \cdot \prod_{j=1}^n (X + f_{\mathbf{b}_j}(Y)) \\ &\quad - \sum_{i=1}^n \prod_{j=1}^n (X + f_{\mathbf{a}_j}(Y)) \cdot \prod_{j \in [n] \text{ s.t. } j \neq i} (X + f_{\mathbf{b}_j}(Y)). \end{aligned}$$

Notice that each term $(X + f_{\mathbf{a}_i}(Y))$ and $(X + f_{\mathbf{b}_i}(Y))$ has degree at most $s-1$ for all $i \in [n]$, hence $g(X, Y)$ has of total degree at most $(s-1)(2n-1)$. We see that if $f(X, Y) \neq 0$ iff $g(X, Y) \neq 0$.

Notice that there are some bad pair (τ, ω) such that $f(\tau, \omega)$ cannot be computable, i.e., $\tau + f_{\mathbf{a}_j}(\omega) = 0$ for some $j \in [n]$. However, in such cases, $g(\tau, \omega)$ is still computable since there is no denominator in $g(X, Y) \in \mathbb{F}(X, Y)$. We see that, for any $(\tau, \omega) \in \mathbb{F} \times \mathbb{F}$, if $f(\tau, \omega)$ is computable and $f(\tau, \omega) = 0$, then $g(\tau, \omega) = 0$. We deduce that

$$\begin{aligned} &\Pr\left[f(\tau, \omega) = 0 \middle| \tau \xleftarrow{\$} \mathbb{F} \wedge \omega \xleftarrow{\$} \mathbb{F}\right] \\ &\leq \Pr\left[g(\tau, \omega) = 0 \middle| \tau \xleftarrow{\$} \mathbb{F} \wedge \omega \xleftarrow{\$} \mathbb{F}\right] \\ &\leq \frac{\left|\left\{\tau \in \mathbb{F} \wedge \omega \in \mathbb{F} \,\middle|\, g(\tau, \omega) = 0\right\}\right|}{|\mathbb{F}|^2} \\ &\leq \frac{\frac{(s-1)(2n-1)}{|\mathbb{F}|} \cdot |\mathbb{F}|^2}{|\mathbb{F}|^2} && \text{(Schwartz–Zippel lemma [Zip79, Sch80])} \\ &= \frac{(s-1)(2n-1)}{|\mathbb{F}|} \end{aligned}$$

as desired. □

Lookup. We recall the following lemma from [Hab22] for supporting lookup arguments.

Lemma 6 (Lemma 5 of [Hab22] and Lemma 4 of [BC23]). *Let n and t be positive integers. Let $(a_i)_{i=1}^n$ and $(b_i)_{i=1}^t$ be sequence over a field $\mathbb{F}$ with characteristic $p > \max(n, t)$. Then, $\{a_i\}_{i=1}^n \subseteq \{b_i\}_{i=1}^t$ iff there exists $(\mathsf{mul}_j)_{j=1}^t$ over $\mathbb{F}$ satisfying*

$$\sum_{i=1}^n \frac{1}{X + a_i} = \sum_{j=1}^t \frac{\mathsf{mul}_i}{X + b_j} \tag{46}$$

in the rational function field $\mathbb{F}(X)$.

Remark 12. In Lemma 6, it suffices to choose a random $\gamma \in \mathbb{F}$ and check if both sides are equal for $X = \gamma$. By multiplying $\prod_{i=1}^{n}(X + a_i)\prod_{j=1}^{t}(X + b_i)$ and subtracting the two sides, we see that the check of Lemma 6 is reduced into proving that a certain polynomial $p(X)$ of degree $d = n + t - 1$ is equal to zero. By Schwartz-Zippel lemma [Zip79, Sch80], if $p(X)$ is not equal to zero, then the probability that $p(\gamma) = 0$ is at most $\frac{d}{\mathbb{F}}$, which is negligible. We can argue the same way for Lemma 3.

Tuple Lookup. We may encounter a tuple lookup argument in our construction. Specifically, let s, n, t be positive integers. Given list $\mathbf{a} = (\mathbf{a}_i)_{i=1}^{n}$ and $\mathbf{b} = (\mathbf{b}_j)_{j=1}^{t}$ where $\mathbf{a}_i \in \mathbb{F}^s$ for all $i \in [n]$ and $\mathbf{b}_j \in \mathbb{F}^s$ for all $j \in [t]$, we would like to establish necessary and sufficient conditions to guarantee that every $\mathbf{a}_i$ is equal to some $\mathbf{b}_j$ in $\mathbf{b}$. We adapt the above lemma, namely, Lemma 6, to achieve the following Lemma 7:

Lemma 7. *Given sequences* $\mathbf{a} = (\mathbf{a}_i)_{i=1}^{n} \in (\mathbb{F}^s)^n$ *and* $\mathbf{b} = (\mathbf{b}_j)_{j=1}^{t} \in (\mathbb{F}^s)^t$ *where* s, n *and* t *are positive integers. Then,* $\{\mathbf{a}_i\}_{i=1}^{n} \subseteq \{\mathbf{b}_j\}_{j=1}^{t}$ *iff there exists* $\mathsf{mul}_1, \ldots, \mathsf{mul}_t \in \mathbb{F}$ *satisfying*

$$\sum_{i=1}^{n} \frac{1}{X + \left\langle \mathbf{a}_i, (Y^k)_{k=0}^{s-1} \right\rangle} = \sum_{j=1}^{t} \frac{\mathsf{mul}_j}{X + \left\langle \mathbf{b}_i, (Y^k)_{k=0}^{s-1} \right\rangle} \tag{47}$$

in the rational function field $\mathbb{F}(X, Y)$ *where* X *and* Y *are variables over* $\mathbb{F}$.

Proof. If $\{\mathbf{a}_i\}_{i=1}^{n} \subseteq \{\mathbf{b}_j\}_{j=1}^{t}$, then it can be seen that (47) trivially holds.

Now, we consider the other direction. Assume that $\mathbf{a}'_1, \ldots, \mathbf{a}'_m$ be distinct vectors satisfying $\{\mathbf{a}'_i\}_{i=1}^{m} = \{\mathbf{a}_i\}_{i=1}^{n}$ for some positive integer $m \leq n$.

For a vector $\mathbf{v} \in \mathbb{F}^s$, define $f_{\mathbf{v}}(Y) = \left\langle \mathbf{v}, (Y^k)_{k=0}^{s-1} \right\rangle$. By defining $\mathsf{mul}'_i = \sum_{j=1}^{n}(\mathbf{a}'_i = \mathbf{a}_j)$, for all $i \in [m]$, we see that

$$\sum_{i=1}^{m} \frac{\mathsf{mul}'_i}{X + f_{\mathbf{a}'_i}(Y)} = \sum_{i=1}^{n} \frac{1}{X + f_{\mathbf{a}_i}(Y)}.$$

Hence, it is sufficient for us to prove that

$$\sum_{i=1}^{m} \frac{\mathsf{mul}'_i}{X + f_{\mathbf{a}'_i}(Y)} = \sum_{j=1}^{t} \frac{\mathsf{mul}_j}{X + f_{\mathbf{b}_j}(Y)}$$

which implies $\{\mathbf{a}_i\}_{i=1}^{n} \subseteq \{\mathbf{b}_j\}_{j=1}^{t}$.

Define

$$f(X, Y) = \sum_{i=1}^{n} \frac{\mathsf{mul}'_i}{X + f_{\mathbf{a}'_i}(Y)} - \sum_{j=1}^{t} \frac{\mathsf{mul}_j}{X + f_{\mathbf{b}_j}(Y)}$$

and

$$g(X, Y) = f(X, Y)\prod_{i=1}^{m}(X + f_{\mathbf{a}'_i}(Y))\prod_{j=1}^{t}(X + f_{\mathbf{b}_j}(Y)).$$

We can see the followings are equivalent:

- (47) is satisfied,
- $f(X, Y) = 0 \in \mathbb{F}(X, Y)$ and
- $g(X, Y) = 0 \in \mathbb{F}(X, Y)$.

Notice that the explicit form of $g(X, Y)$ can be written to be

$$\begin{aligned} g(X, Y) = &\sum_{i=1}^{m} \mathsf{mul}'_i \cdot \prod_{j \in [m] \text{ s.t. } j \neq i} (X + f_{\mathbf{a}'_j}(Y)) \cdot \prod_{j=1}^{t}(X + f_{\mathbf{b}_j}(Y)) \\ &- \sum_{i=1}^{t} \mathsf{mul}_i \cdot \prod_{j=1}^{m}(X + f_{\mathbf{a}'_j}(Y)) \cdot \prod_{j \in [t] \text{ s.t. } j \neq i} (X + f_{\mathbf{b}_j}(Y)). \end{aligned}$$

Notice that the total degree of $g(X,Y)$ is at most $n+t-1$. Assume that $g(X,Y)=0 \in \mathbb{F}(X,Y)$. Then, we see that, for each $k \in [m]$, by replacing X by $-f_{\mathbf{a}'_k}(Y)$, we see that

$$\begin{aligned}
&g(-f_{\mathbf{a}'_k}(Y), Y) \\
&= \sum_{i=1}^{m} \mathsf{mul}'_i \cdot \prod_{j\in[m] \text{ s.t. } j\neq i} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{a}'_j}(Y)) \cdot \prod_{j=1}^{t} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{b}_j}(Y)) \\
&\quad - \sum_{i=1}^{t} \mathsf{mul}_i \cdot \prod_{j=1}^{m} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{a}'_j}(Y)) \cdot \prod_{j\in[t] \text{ s.t. } j\neq i} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{b}_j}(Y)) \\
&= \mathsf{mul}'_k \cdot \prod_{j\in[m] \text{ s.t. } j\neq k} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{a}'_j}(Y)) \cdot \prod_{j=1}^{t} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{b}_j}(Y)).
\end{aligned}$$

Since we assumed that $g(X,Y)=0$, $\mathsf{mul}'_k \neq 0$ and $\mathbf{a}'_1, \ldots, \mathbf{a}'_m$ are distinct, we see that

$$\mathsf{mul}'_k \cdot \prod_{j\neq k} (-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{a}'_j}(Y)) \neq 0.$$

Hence, it can be seen that $\prod_{j=1}^{t}(-f_{\mathbf{a}'_k}(Y) + f_{\mathbf{b}_j}(Y)) = 0$. Therefore, $\prod_{j=1}^{t}(X + f_{\mathbf{b}_j}(Y))$ contains a factor $X + f_{\mathbf{a}'_k}(Y)$ which implies $\mathbf{a}'_k \in \{\mathbf{b}_j\}_{j=1}^{t}$, and this holds for all $k \in [m]$. □

Testing Tuple Lookup. If sequences $\mathbf{a}$ and $\mathbf{b}$ do not satisfy Lemma 7, i.e., exists $\mathbf{a}_i \notin \{\mathbf{b}_j\}_{j=1}^{t}$, by sampling $\psi \xleftarrow{\$} \mathbb{F}$ and $\chi \xleftarrow{\$} \mathbb{F} \setminus \{\langle \mathbf{b}_i, (\psi^k)_{k=0}^{s-1} \rangle\}_{i=1}^{t}$, and setting $X = \chi$ and $Y = \psi$, we can see that (47) holds with probability at most

$$\mathsf{err}_{\mathsf{lookup}}(\mathbb{F}, s, n, t) = \frac{(s-1)(n+t-1)}{|\mathbb{F}|} = \mathcal{O}\left(\frac{s \cdot (n+t)}{|\mathbb{F}|}\right). \tag{48}$$

according to the following Lemma 8

Lemma 8 (Tuple Lookup Error Probability). *Given sequences $\mathbf{a} = (\mathbf{a}_i)_{i=1}^{n} \in (\mathbb{F}^s)^n$ and $\mathbf{b} = (\mathbf{b}_j)_{j=1}^{t} \in (\mathbb{F}^s)^t$ where s, n and t are positive integers. For a vector $\mathbf{v} \in \mathbb{F}^s$, define $f_{\mathbf{v}}(Y) = \langle \mathbf{v}, (Y^k)_{k=0}^{s-1} \rangle$. Assume that*

$$\sum_{i=1}^{n} \frac{1}{X + f_{\mathbf{a}_i}(Y)} \neq \sum_{j=1}^{t} \frac{\mathsf{mul}_j}{X + f_{\mathbf{b}_j}(Y)}$$

for some $\mathsf{mul}_1, \ldots, \mathsf{mul}_t \in \mathbb{F}$. Then,

$$\Pr\left[f(\chi,\psi) = 0 \middle| \psi \xleftarrow{\$} \mathbb{F} \wedge \chi \xleftarrow{\$} \mathbb{F}\right] \leq \frac{(s-1)(n+t-1)}{|\mathbb{F}|}$$

where

$$f(X,Y) = \sum_{i=1}^{n} \frac{1}{X + f_{\mathbf{a}_i}(Y)} - \sum_{j=1}^{t} \frac{\mathsf{mul}_j}{X + f_{\mathbf{b}_j}(Y)}.$$

Proof. Define

$$\begin{aligned}
g(X,Y) &= f(X,Y) \prod_{i=1}^{n} (X + f_{\mathbf{a}_i}(Y)) \prod_{j=1}^{t} (X + f_{\mathbf{b}_j}(Y)) \\
&= \sum_{i=1}^{n} \prod_{j\in[n] \text{ s.t. } j\neq i} (X + f_{\mathbf{a}_j}(Y)) \cdot \prod_{j=1}^{t} (X + f_{\mathbf{b}_j}(Y)) \\
&\quad - \sum_{i=1}^{t} \mathsf{mul}_i \cdot \prod_{j=1}^{n} (X + f_{\mathbf{a}_j}(Y)) \cdot \prod_{j\in[t] \text{ s.t. } j\neq i} (X + f_{\mathbf{b}_j}(Y))
\end{aligned}$$

of total degree at most $(s-1)(n+t-1)$. We see that if $f(X,Y)=0$, it must hold that $g(X,Y)=0$ as well. Similarly to Lemma 5, since the degree of $g(X,Y)$ is at most $n+t-1$, we deduce that

$$\begin{aligned}
&\Pr\left[f(\chi,\psi)=0 \middle| \psi \xleftarrow{\$} \mathbb{F} \wedge \chi \xleftarrow{\$} \mathbb{F}\right] \\
&\leq \Pr\left[g(\chi,\psi)=0 \middle| \psi \xleftarrow{\$} \mathbb{F} \wedge \chi \xleftarrow{\$} \mathbb{F}\right] \\
&\leq \frac{\frac{(s-1)(n+t-1)}{|\mathbb{F}|} \cdot |\mathbb{F}|^2}{|\mathbb{F}|^2} && \text{(Schwartz–Zippel lemma [Zip79, Sch80])} \\
&= \frac{(s-1)(n+t-1)}{|\mathbb{F}|}
\end{aligned}$$

as desired. □

Remark 13. Notice that, there are some bad χ such that $f(\chi,\psi)$ cannot be computable, i.e., $\chi + f_{\mathbf{a}_j}(\psi) = 0$ for some $j \in [n]$. For such challenges, the prover could never produce a valid witness leading to completeness error, and in fact, it reveals partial information of $\mathbf{a}_j$, since $f_{\mathbf{a}_j}(\psi) = -\chi$. Fortunately, such bad challenges happen with probability at most $t/|\mathbb{F}|$. Hence we also conclude that the tuple lookup argument above has completeness error $\mathcal{O}(t/|\mathbb{F}|)$.

B.5 Schwartz-Zippel Lemma

We recall the Schwartz-Zippel lemma [Zip79, Sch80] in the following Lemma 9.

Lemma 9 (Schwartz-Zippel Lemma). *Let $\mathbb{F}$ be a field and multivariate polynomial $f \in \mathbb{F}[X_1, X_2, \ldots, X_n]$ be non-zero and of total degree d. Let S be a finite subset of $\mathbb{F}$ and suppose $|S| > d$, then it holds that*

$$\Pr\left[f(x_1,x_2,\ldots,x_n)=0 \middle| (x_1,x_2,\ldots,x_n) \xleftarrow{\$} S^n\right] \leq \frac{d}{|S|}.$$

B.6 Commitment Scheme (Extended)

This appendix is an extension of Section 3.1. We recall the syntax of commitment schemes, denoted by $\mathcal{C}$, in Definition 10. Additionally, we require $\mathcal{C}$ in our construction to be additively homomorphic.

Definition 10 (Syntax of Commitment Scheme). *A commitment scheme $\mathcal{C}$ is a tuple of algorithms $\mathcal{C} = (\mathsf{C.Setup}, \mathsf{C.Commit}, \mathsf{C.Verify})$ defined as follows:*

$\mathsf{C.Setup}(1^\lambda) \to \mathsf{ck}$: *On input 1^λ, output a commitment key ck and determine randomness distribution R_{ck}.*

$\mathsf{C.Commit}(\mathsf{ck}, M, R) \to C$: *On input key ck, message M and randomness R sampled from some randomness distribution R_{ck}, output commitment C.*

$\mathsf{C.Verify}(\mathsf{ck}, M, R, C) \to \{0,1\}$: *On input commitment key ck, message M, randomness R and commitment C, output a bit $b \in \{0,1\}$.*

$\mathcal{C}$ should satisfy perfect correctness (Definition 11) and two additional security properties, namely *binding* and *hiding*, formally defined in Definitions 12 and 13, respectively.

Security of Commitment Schemes. Let $\mathcal{C}$ be a commitment scheme with syntax in Definition 10. We now define the completeness, binding and hiding of $\mathcal{C}$ in the following Definitions 11, 12 and 13, respectively.

Definition 11 (Perfect Correctness of $\mathcal{C}$). *$\mathcal{C}$ satisfies correctness if for all message M, randomness R, it holds that*

$$\Pr\left[\mathsf{C.Verify}(\mathsf{ck}, M, R, C) = 1 \middle| \begin{matrix} \mathsf{ck} \leftarrow \mathsf{C.Setup}(1^\lambda) \\ C \leftarrow \mathsf{C.Commit}(\mathsf{ck}, M, R) \end{matrix}\right] = 1.$$

Definition 12 (Binding of $\mathcal{C}$). *$\mathcal{C}$ is binding if for all (PPT) adversaries $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{array}{l} M_1 \neq M_2 \\ \wedge \mathsf{C.Verify}(\mathsf{ck}, M_1, R_1, C) = 1 \\ \wedge \mathsf{C.Verify}(\mathsf{ck}, M_2, R_1, C) = 1 \end{array} \middle| \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{C.Setup}(1^\lambda) \\ (M_1, R_1, M_2, R_2, C) \leftarrow \mathcal{A}(\mathsf{ck}) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

If $\mathcal{A}$ is PPT, we say that $\mathcal{C}$ is computationally binding. Otherwise, if $\mathcal{A}$ is computationally unbounded, we say that $\mathcal{C}$ is statistically binding.

Definition 13 (Hiding of $\mathcal{C}$). *A commitment scheme satisfies hiding if for any messages M and M', then the two following distributions are close:*

$$\left\{ C \middle| \begin{array}{l} R \xleftarrow{\$} \mathsf{R_{ck}} \\ C \leftarrow \mathsf{C.Commit}(\mathsf{ck}, M, R) \end{array} \right\} \text{ and } \left\{ C' \middle| \begin{array}{l} R' \xleftarrow{\$} \mathsf{R_{ck}} \\ C' \leftarrow \mathsf{C.Commit}(\mathsf{ck}, M', R') \end{array} \right\}.$$

If the above two distributions are computationally close, we say that $\mathcal{C}$ is computationally hiding. Otherwise, if they are statistically close, we say that $\mathcal{C}$ is statistically hiding.

Homomorphic Commitment Schemes. A commitment scheme $\mathcal{C}$, with syntax defined in Definition 10, is said to be homomorphic if

$$\begin{aligned} &\mathsf{C.Commit}(\mathsf{ck}, M_1, R_1) + \mathsf{C.Commit}(\mathsf{ck}, M_2, R_2) \\ &= \mathsf{C.Commit}(\mathsf{ck}, M_1 + M_2, R_1 + R_2). \end{aligned}$$

Remark 14. By using notation in Remark 4, for two commitment tuples, e.g., $[\![\mathbf{c}_0]\!]_{\mathsf{ck}}$ and $[\![\mathbf{c}_1]\!]_{\mathsf{ck}}$. If either $[\![\mathbf{c}_0]\!]_{\mathsf{ck}} \notin \mathcal{R}_{\mathsf{com}}$ or $[\![\mathbf{c}_1]\!]_{\mathsf{ck}} \notin \mathcal{R}_{\mathsf{com}}$, then, in many situations in this paper, we may face the form $[\![\mathbf{c}]\!]_{\mathsf{ck}} := [\![\mathbf{c}_0]\!]_{\mathsf{ck}} + \alpha \cdot [\![\mathbf{c}_1]\!]_{\mathsf{ck}}$ for some α chosen uniformly from $\mathbb{F}$. Due to the binding property of commitment schemes, the probability that $[\![\mathbf{c}]\!]_{\mathsf{ck}} \in \mathcal{R}_{\mathsf{com}}$ is negligible.

An Instantiation of Homomorphic Commitment Schemes. Let $n \in \mathbb{Z}_+$. Below, we describe the Pedersen commitment scheme [Ped92], a secure and homomorphic commitment for committing to length-n vectors. As a remark, just for this instantiation, the addition "+" is a group operation between group elements in $\mathbb{G}$ while multiplication "$\cdot$" is an action between a scalar and a group element in $\mathbb{G}$.

$\mathsf{C.Setup}(1^\lambda) \to \mathsf{ck}$**:** Sample $g_1, \ldots, g_n, h \xleftarrow{\$} \mathbb{G}$ and return $\mathsf{ck} = (g_1, \ldots, g_n, h)$.
$\mathsf{C.Commit}(\mathsf{ck}, \mathbf{x} = (x_1, \ldots, x_n)) \to (C, R)$**:** Sample R uniformly and output $C = R \cdot h + \sum_{i=1}^n x_i \cdot g_i$.
$\mathsf{C.Verify}(\mathsf{ck}, \mathbf{x} = (x_1, \ldots, x_n), R, C) \to \{0, 1\}$**:** Output 1 if $C = R \cdot h + \sum_{i=1}^n x_i \cdot g_i$ and 0 otherwise.

B.7 Special Soundness

We recall the special soundness property for multi-round protocols [ACK21]. We first recall the special soundness property for 3-move protocols in Definition 14, then we recall the generalization for multi-round protocols in Definition 16.

Definition 14 ($(k;n)$-Special Soundness). *Let $k, n \in \mathbb{N}$. Let* CH *be a set such that $|\mathrm{CH}| = n$. Let Π be public-coin* 3*-move protocol for a relation $\mathcal{R}$ with challenge set* CH*. Then Π is $(k;n)$-special sound if there exists a PPT extractor $\mathcal{E}$, such that, given an instance Z and k accepting transcripts $(a, c_i, z_i)_{i=1}^k$ with the same first message a and pairwise distinct challenges $c_i \in \mathrm{CH}$, extractor $\mathcal{E}$ can output a witness W such that $(Z, W) \in \mathcal{R}$.*

To generalize the above property for multi-round protocols, we recall the notion of the tree of transcripts [ACK21] in Definition 15. Then, we state the special soundness property for multi-round protocols in Definition 16.

Definition 15 (Tree of Transcripts). *Let Π be a public-coin $(2\mu{+}1)$-move protocol. A $(k_1, \ldots, k_\mu)$-tree of transcript is a set of transcripts arranged in the following tree format: Each node corresponds to a message of prover and each edge corresponds to a challenge of $\mathcal{V}$. For each $i < \mu$, each node of depth i has exactly k_i children, corresponding to k_i pairwise distinct challenges of $\mathcal{V}$. Every transcript corresponds to exactly one path from the root to a leaf of the tree.*

Definition 16 ($(k_1, \ldots, k_\mu; n_1, \ldots, n_\mu)$-Special Soundness). *Let $k_1, \ldots, k_\mu$, and $n_1, \ldots, n_u$ be positive integers in $\mathbb{Z}_+$. Let $\mathrm{CH}_1, \mathrm{CH}_2, \ldots, \mathrm{CH}_\mu$ be sets such that $|\mathrm{CH}_i| = n_i$ for each $1 \leq i \leq \mu$. Let Π be a public-coin $(2\mu+1)$-move protocol for a relation $\mathcal{R}$ where the i-th challenge is sampled from CH_i. Then Π satisfies $(k_1, \ldots, k_\mu; n_1, \ldots, n_\mu)$-special soundness if there exists a PPT algorithm $\mathcal{E}$, such that, given an instance $\mathbb{z}$ and a $(k_1, \ldots, k_\mu)$-tree of accepted transcripts, outputs a witness W such that $(Z, W) \in \mathcal{R}$.*

B.8 Folding Scheme

We recall the definition and security properties of folding schemes of [KST22] in Definitions 18, 19 and 20 respectively.

Definition 17 (Syntax of Folding Scheme). *Let $\mathcal{PP}$, $\mathcal{I}$, and $\mathcal{Z}$ be the sets of public parameters, instances, and witnesses, respectively. Let $\mathcal{R} \subseteq \mathcal{PP} \times \mathcal{I} \times \mathcal{Z}$ be an NP relation. A folding scheme $\mathcal{FS}$ for relation $\mathcal{R}$, is a tuple $\mathcal{FS}[\mathcal{R}] = (\mathsf{FS.Setup}, \mathsf{FS.Fold}, \mathsf{FS.Verify})$ of algorithms $\mathsf{FS.Setup}, \mathsf{FS.Fold}, \mathsf{FS.Verify}$, run as follows:*

- $\mathsf{FS.Setup}(1^\lambda) \to \mathsf{pp}$**:** *Run public parameter generator, on input a security parameter 1^λ, it returns a public parameter pp.*
- $\mathsf{FS.Fold}(\mathsf{pp}, I_0, I_1;\ Z_0, Z_1) \to (I;\ Z)$**:** *This algorithm is run by prover. On inputs public parameter pp and instance-witness pairs $(I_0, Z_0), (I_1, Z_1) \in \mathcal{I} \times \mathcal{Z}$, it returns an instance-witness pair $(I, Z) \in \mathcal{I} \times \mathcal{Z}$.*
- $\mathsf{FS.Verify}(\mathsf{pp}, I_0, I_1) \to I$**:** *This algorithm is run by verifier. On inputs public parameter pp and instances $I_0, I_1 \in \mathcal{I}$, it returns an instance $Z \in \mathcal{I}$.*

Defining Transcript. For common public inputs pp, I_0, I_1, we denote

$$(I, Z) \leftarrow \Pi_{\mathcal{FS}}(\mathsf{pp}, (I_0, I_1;\ Z_0, Z_1))$$

the output of prover and verifier when prover executes $\mathsf{FS.Fold}$ with inputs $\mathsf{pp}, I_0, I_1, Z_0, Z_1$ while verifier executes $\mathsf{FS.Verify}$ with inputs pp, I_0, I_1. Define the public transcript

$$\mathsf{tr} \leftarrow \mathsf{View}(\Pi_{\mathcal{FS}}(\mathsf{pp}, (I_0, I_1;\ Z_0, Z_1))$$

to contains all the inputs, outputs, and public messages between the prover and verifier when executing $\Pi_{\mathcal{FS}}$.

Definition 18 (Perfect Correctness of $\mathcal{FS}$). *Let $\mathsf{pp} \leftarrow \mathsf{FS.Setup}(1^\lambda)$. Then for any $\{(\mathsf{pp}, I_i;\ Z_i)\}_{i \in \{0,1\}} \subseteq \mathcal{R}$, it holds that*

$$\Pr\left[(\mathsf{pp}, I;\ Z) \in \mathcal{R} \middle| (I, Z) \leftarrow \Pi_{\mathcal{FS}}(\mathsf{pp}, (I_0, I_1;\ Z_0, Z_1))\right] = 1.$$

Definition 19 (Knowledge Soundness of $\mathcal{FS}$). *$\mathcal{FS}$ is said to satisfy knowledge soundness if for any positive integer N, any (PPT) algorithm $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$, it holds that $\Pr[\mathsf{E}_{\mathcal{A}}^{\mathsf{fs\text{-}sound}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ where $\mathsf{E}_{\mathcal{A}}^{\mathsf{fs\text{-}sound}}(\lambda)$ is described in Figure 8. Note that in this definition (which is also the FS definition of [KST22], the extractor is given the private randomness ρ of the prover, while in our CFS definition, no such ρ is given to it. A folding scheme $\mathcal{FS}$ is statistically (respectively, computationally) knowledge-sound if $\mathcal{A}$ is computationally unbounded (respectively, bounded). $\mathcal{FS}$ has soundness error ϵ if $\mathcal{A}$ can break knowledge soundness with probability at most ϵ.*

Definition 20 (HVZK of $\mathcal{FS}$). *$\mathcal{FS}$ is HVZK if there exists a PPT simulator $\mathcal{S}$ s.t., for any distinguisher $\mathcal{A}$, any valid instance-witness pairs $(Z_0, W_0), (Z_1, W_1)$, it holds that*

$$\begin{aligned}\Big|\Pr\left[\mathcal{A}(\mathsf{tr}) = 1 \middle| \mathsf{pp} \leftarrow \mathsf{FS.Setup}(1^\lambda), \mathsf{tr} \leftarrow \mathsf{View}(\Pi_{\mathcal{FS}}(\mathsf{pp}, I_0, I_1;\ Z_0, Z_1))\right] \\ - \Pr\left[\mathcal{A}(\mathsf{tr}) = 1 \middle| \mathsf{pp} \leftarrow \mathsf{FS.Setup}(1^\lambda), \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{pp}, I_0, I_1)\right]\Big| \leq \mathsf{negl}(\lambda)\end{aligned}$$

$\mathsf{pp} \leftarrow \mathsf{FS.Setup}(1^\lambda)$.
$(I_0, I_1, \rho) \leftarrow \mathcal{A}(\mathsf{pp})$.
$(I, Z) \leftarrow \Pi_{\mathcal{FS}}(\mathsf{pp}, (I_0, I_1;\ Z_0, Z_1))$
$(Z_0, Z_1) \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{pp}, Z_0, Z_1, \rho)$.
$b_0 = ((\mathsf{pp}, I, Z) \in \mathcal{R})$
$b_1 = ((\mathsf{pp}, I_0, Z_0) \notin \mathcal{R}) \vee ((\mathsf{pp}, I_1, Z_1) \notin \mathcal{R})$
Return $b_0 \wedge b_1$.

Fig. 8. Experiment $\mathsf{E}_{\mathcal{A}}^{\mathsf{fs\text{-}sound}}(\lambda)$.

B.9 Interactive Folding Protocol for Folding rR1CS Instance-Witness Pairs (Extended)

Explanation of Equations in Section 3.2. We now explain in detail the equations in Section 3.2. First, (11) is fully written to be

$$\begin{aligned}
&\mathbf{A}\cdot\mathbf{z}'\circ\mathbf{B}\cdot\mathbf{z}'\\
&=\mathbf{A}\cdot\mathbf{z}'_0\circ\mathbf{B}\cdot\mathbf{z}'_0+\alpha(\mathbf{A}\cdot\mathbf{z}'_0\circ\mathbf{B}\cdot\mathbf{z}'_1+\mathbf{A}\cdot\mathbf{z}'_1\circ\mathbf{B}\cdot\mathbf{z}'_0)\\
&\quad+\alpha^2(\mathbf{A}\cdot\mathbf{z}'_1\circ\mathbf{B}\cdot\mathbf{z}'_1)\\
&=(\mathbb{x}_0.u\cdot\mathbf{C}\cdot\mathbf{z}'_0+\mathbb{x}_0.\mathbf{e})+\alpha(\mathbf{A}\cdot\mathbf{z}'_0\circ\mathbf{B}\cdot\mathbf{z}'_1+\mathbf{A}\cdot\mathbf{z}'_1\circ\mathbf{B}\cdot\mathbf{z}'_0)\\
&\quad+\alpha^2(\mathbb{x}_1.u\cdot\mathbf{C}\cdot\mathbf{z}'_1+\mathbb{x}_1.\mathbf{e})\\
&=(\mathbb{x}_0.u+\alpha\cdot\mathbb{x}_1.u)\cdot\mathbf{C}\cdot(\mathbf{z}'_0+r\cdot\mathbf{z}'_1)\\
&\quad-\alpha\cdot\mathbb{x}_1.u\cdot\mathbf{C}\cdot\mathbf{z}'_0-\alpha\cdot\mathbb{x}_0.u\cdot\mathbf{C}\cdot\mathbf{z}'_1+\mathbb{x}_0.\mathbf{e}\\
&\quad+\alpha(\mathbf{A}\cdot\mathbf{z}'_0\circ\mathbf{B}\cdot\mathbf{z}'_1+\mathbf{A}\cdot\mathbf{z}'_1\circ\mathbf{B}\cdot\mathbf{z}'_0)+\alpha^2\cdot\mathbb{x}_1.\mathbf{e}\\
&=\mathbb{x}.u\cdot\mathbf{C}\cdot\mathbf{z}'+\mathbb{x}.\mathbf{e}
\end{aligned}$$

and the error $\mathbb{x}.\mathbf{e}$, as show in (3.2), can be written in details to be

$$\begin{aligned}
&\alpha(-\mathbb{x}_1.u\cdot\mathbf{C}\cdot\mathbf{z}'_0-\mathbb{x}_0.u\cdot\mathbf{C}\cdot\mathbf{z}'_1)+\mathbb{x}_0.\mathbf{e}\\
&\quad+\alpha(\mathbf{A}\cdot\mathbf{z}'_0\circ\mathbf{B}\cdot\mathbf{z}'_1+\mathbf{A}\cdot\mathbf{z}'_1\circ\mathbf{B}\cdot\mathbf{z}'_0)+\alpha^2\cdot\mathbb{x}_1.\mathbf{e}\\
&=\mathbb{x}_0.\mathbf{e}+\alpha^2\cdot\mathbb{x}_1.\mathbf{e}\\
&\quad+\alpha(\underbrace{\mathbf{A}\cdot\mathbf{z}'_0\circ\mathbf{B}\cdot\mathbf{z}'_1+\mathbf{A}\cdot\mathbf{z}'_1\circ\mathbf{B}\cdot\mathbf{z}'_0-\mathbb{x}_1.u\cdot\mathbf{C}\cdot\mathbf{z}'_0-\mathbb{x}_0.u\cdot\mathbf{C}\cdot\mathbf{z}'_1}_{\text{garbage term}}).
\end{aligned}$$

Proof of Lemma 1. Here, we provide a proof of Lemma 1. We recall Lemma 1 for readability purpose as following Lemma 10.

Lemma 10 (Recall of Lemma 1). *Let $\mathcal{C}$ be a homomorphic commitment scheme. Assume that Π_{rr1cs} in Figure 3 are rewinded thrice (from step 4), with the same $[\![\mathbf{g}]\!]_{\mathsf{cke}}$ and distinct $\{\alpha^{(i)}\}_{i\in[3]}$, to produce $\{[\![\mathbb{x}^{(i)}]\!]\}_{i\in[3]}$, respectively. If we have the valid witnesses s.t. $[\![\mathbb{x}^{(i)}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathcal{S}}_{\mathsf{rr1cs}}\ \forall i \in [3]$, then we can extract witnesses to construct $([\![\mathbb{x}_i]\!]_{\mathsf{tck}})_{i\in\{0,1\}}$ s.t. $[\![\mathbb{x}_i]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathcal{S}}_{\mathsf{rr1cs}}\ \forall i \in \{0,1\}$.*

As a remark, this lemma satisfies $(3; |\mathbb{F}|)$-special soundness defined in Appdx. B.7.

Before going to the proof, we first recall the notations. Recall in Section 3.2, for a vector $\mathbb{x}$, we can parse

$$\mathbb{x} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, \mathbb{x}.\mathbf{z}_1, \ldots, \mathbb{x}.\mathbf{z}_d, \mathbb{x}.\mathbf{e}) \tag{49}$$

where $\mathbb{x}.u$ and $\mathbb{x}.\mathsf{pub}$ are scalars and $\mathbb{x}.\mathbf{z}_1, \ldots, \mathbb{x}.\mathbf{z}_d, \mathbb{x}.\mathbf{e}$ are vectors. Also, for a tuple commitment key $\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_d, \mathsf{cke})$, from (9), we can parse $[\![\mathbb{x}]\!]_{\mathsf{tck}}$ as

$$[\![\mathbb{x}]\!]_{\mathsf{tck}} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, [\![\mathbb{x}.\mathbf{z}_1]\!]_{\mathsf{ck}_1}, \ldots, [\![\mathbb{x}.\mathbf{z}_d]\!]_{\mathsf{ck}_d}, [\![\mathbb{x}.\mathbf{e}]\!]_{\mathsf{cke}})$$

According to Remark 4, we additionally parse

$$[\![\mathbb{x}.\mathbf{z}_i]\!]_{\mathsf{ck}_i} = (\mathsf{ck}_i, \mathbb{x}.\tilde{\mathbf{z}}_i;\ \mathbb{x}.\mathbf{z}_i, \mathbb{x}.\hat{\mathbf{z}}_i)\ \forall i \in [d] \text{ and } [\![\mathbb{x}.\mathbf{e}]\!]_{\mathsf{cke}} = (\mathsf{cke}, \mathbb{x}.\tilde{\mathbf{e}};\ \mathbb{x}.\mathbf{e}, \mathbb{x}.\hat{\mathbf{e}})$$

where $\mathbb{x}.\tilde{\mathbf{z}}_i$ and $\mathbb{x}.\hat{\mathbf{z}}_i$ are the commitment and randomness, respectively, to vector $\mathbb{x}.\mathbf{z}_i$ for all $i \in [d]$. We denote by

$$\tilde{\mathbb{x}} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, \mathbb{x}.\tilde{\mathbf{z}}_1, \ldots, \mathbb{x}.\tilde{\mathbf{z}}_d, \mathbb{x}.\tilde{\mathbf{e}}) \tag{50}$$

to contains all public information and commitments,

$$\hat{\mathbb{x}} = (\mathbb{x}.\hat{\mathbf{z}}_1, \ldots, \mathbb{x}.\hat{\mathbf{z}}_d, \mathbb{x}.\hat{\mathbf{e}}) \tag{51}$$

to contain all associated randomness to the component vectors of $\mathbb{x}$ employed for the commitments, such that we can write $[\![\mathbb{x}]\!]_{\mathsf{tck}} = (\mathsf{tck}, \tilde{\mathbb{x}};\ \mathbb{x}, \hat{\mathbb{x}})$. Finally, recall in (10), $\mathbb{x}$ is a valid pair of rR1CS if it satisfies the relation

$$\mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}} = \left\{ [\![\mathbb{x}]\!]_{\mathsf{tck}} \;\middle|\; \begin{array}{l} \mathbb{x}.u \in \mathbb{F} \wedge \mathbb{x}.\mathsf{pub} \in \mathbb{F} \wedge (\mathbb{x}.\mathbf{z}_i \in \mathbb{F}^{m_i}\ \forall i \in [d]) \wedge \mathbb{x}.\mathbf{e} \in \mathbb{F}^n \\ \wedge [\![\mathbb{x}.\mathbf{z}_i]\!]_{\mathsf{ck}_i} \in \mathcal{R}_{\mathsf{com}}\ \forall i \in [d] \wedge [\![\mathbb{x}.\mathbf{e}]\!]_{\mathsf{cke}} \in \mathcal{R}_{\mathsf{com}} \\ \wedge \mathbf{A} \cdot \mathbf{z}' \circ \mathbf{B} \cdot \mathbf{z}' = \mathbb{x}.u \cdot \mathbf{C} \cdot \mathbf{z}' + \mathbb{x}.\mathbf{e} \end{array} \right\}$$

where $\mathbf{z}' = (\mathbb{x}.\mathsf{pub} \| \mathbb{x}.\mathbf{z}'_1 \| \ldots \| \mathbb{x}.\mathbf{z}'_d)$ and $\circ$ is the entry-wise multiplication.

Proof (Proof of Lemma 1 (recalled in Lemma 10)). First, consider the system

$$\begin{cases} \mathbb{x}^{(i)}.\mathsf{pub} = \mathbb{x}_0.\mathsf{pub} + \alpha^{(i)} \cdot \mathbb{x}_1.\mathsf{pub} & \forall i \in [2], \\ \mathbb{x}^{(i)}.\mathbf{z}_j = \mathbb{x}_0.\mathbf{z}_j + \alpha^{(i)} \cdot \mathbb{x}_1.\mathbf{z}_j & \forall i \in [2], \forall j \in [d], \\ \mathbb{x}^{(i)}.\mathbf{e} = \mathbb{x}_0.\mathbf{e} + \alpha^{(i)} \cdot \mathbf{g} + (\alpha^{(i)})^2 \cdot \mathbb{x}_1.\mathbf{e} & \forall i \in [3], \\ \mathbb{x}^{(i)}.\hat{\mathbf{z}}_j = \mathbb{x}_0.\hat{\mathbf{z}}_j + \alpha^{(i)} \cdot \mathbb{x}_1.\hat{\mathbf{z}}_j & \forall i \in [2], \forall j \in [d], \\ \mathbb{x}^{(i)}.\hat{\mathbf{e}} = \mathbb{x}_0.\hat{\mathbf{e}} + \alpha^{(i)} \cdot \hat{\mathbf{g}} + (\alpha^{(i)})^2 \cdot \mathbb{x}_1.\hat{\mathbf{e}} & \forall i \in [3]. \end{cases} \tag{52}$$

As each $\alpha^{(i)}$, for $i \in [3]$, has powers at most 3 in system (52) and we have 3 distinct challenges $\{\alpha^{(i)}\}_{i\in[3]}$, hence, we can solve system (52). By solving (52), we can extract $\mathbb{x}_i.\mathsf{pub}$, $\mathbb{x}_i.\mathbf{z}_j$ and $\mathbb{x}_i.\hat{\mathbf{z}}_j$ for all $i \in [2]$ and all $j \in [d]$. Hence, we can extract $\mathbb{x}_0$ and $\mathbb{x}_1$ of the form (49), $\hat{\mathbb{x}}_0$ and $\hat{\mathbb{x}}_1$ of the form (51), vector $\mathbf{g}$ and randomness $\hat{\mathbf{g}}$ and thus form the pairs $[\![\mathbb{x}_0]\!]_{\mathsf{tck}} = (\mathsf{tck}, \tilde{\mathbb{x}}_0;\ \mathbb{x}_0, \hat{\mathbb{x}}_0)$ and $[\![\mathbb{x}_1]\!]_{\mathsf{tck}} = (\mathsf{tck}, \tilde{\mathbb{x}}_1;\ \mathbb{x}_1, \hat{\mathbb{x}}_1)$.

It suffices to prove that the extracted vectors $\mathbb{x}_0, \mathbb{x}_1$ above are valid, i.e., they satisfy $[\![\mathbb{x}_0]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ and $[\![\mathbb{x}_1]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$. We proceed as follows.

First, let us prove that $[\![\mathbb{x}_i.\mathbf{z}_j]\!]_{\mathsf{ck}_j}, [\![\mathbb{x}_i.\mathbf{e}]\!]_{\mathsf{cke}} \in \mathcal{R}_{\mathsf{com}}\ \forall i \in \{0,1\}, \forall j \in [d]$. From the statement of this lemma, notice that we have $[\![\mathbb{x}^{(i)}.\mathbf{z}_j]\!]_{\mathsf{ck}_j} \in \mathcal{R}_{\mathsf{cond}}$ for all $i \in [3]$ and $j \in [d]$. In addition, according to Π_{rr1cs} in Figure 3 and due to the homomorphism and binding properties of the commitment scheme, for each $i \in [2]$, it implies that

$$\begin{aligned} &\mathbb{x}_0.\tilde{\mathbf{z}}_j + \alpha^{(i)} \cdot \mathbb{x}_1.\tilde{\mathbf{z}}_j = \mathsf{C.Commit}(\mathbb{x}_0.\mathbf{z}_j, \mathbb{x}_0.\hat{\mathbf{z}}_j) + \alpha^{(i)} \cdot \mathsf{C.Commit}(\mathbb{x}_1.\mathbf{z}_j, \mathbb{x}_1.\hat{\mathbf{z}}_j) \\ &= \mathsf{C.Commit}(\mathbb{x}_0.\mathbf{z}_j + \alpha^{(i)} \cdot \mathbb{x}_1.\mathbf{z}_j, \mathbb{x}_0.\hat{\mathbf{z}}_j + \alpha^{(i)} \cdot \mathbb{x}_1.\hat{\mathbf{z}}_j) = \mathsf{C.Commit}(\mathbb{x}^{(i)}.\mathbf{z}_j, \mathbb{x}^{(i)}.\hat{\mathbf{z}}_j) \\ &= \mathbb{x}^{(i)}.\tilde{\mathbf{z}}_j. \end{aligned}$$

Since the equation above holds for three distinct $\{\alpha^{(i)}\}_{i\in[3]}$, we must have $\mathbb{x}_i.\tilde{\mathbf{z}}_j = \mathsf{C.Commit}(\mathbb{x}_i.\mathbf{z}_j, \mathbb{x}_i.\hat{\mathbf{z}}_j)$ for all $j \in \{0,1\}$, or equivalently, $[\![\mathbb{x}_i.\mathbf{z}_j]\!]_{\mathsf{ck}_j} \in \mathcal{R}_{\mathsf{com}}\ \forall i \in \{0,1\}, \forall j \in [d]$. In fact, by binding property of $\mathcal{C}$, if prover does not have $[\![\mathbb{x}_{i'}.\mathbf{z}_j]\!]_{\mathsf{ck}_j} \in \mathcal{R}_{\mathsf{com}}$ for some $i' \in \{0,1\}, j \in [d]$, then the case that all $[\![\mathbb{x}^{(i)}.\mathbf{z}_j]\!] \in \mathcal{R}_{\mathsf{com}}\ \forall i \in [3]$ hold is with negligible probability according to Remark 14.

Similarly, by considering $\mathbb{x}^{(i)}.\tilde{\mathbf{e}} = \mathbb{x}_0.\tilde{\mathbf{e}} + \alpha^{(i)} \cdot \tilde{\mathbf{g}} + \left(\alpha^{(i)}\right)^2 \cdot \mathbb{x}_1.\tilde{\mathbf{e}}$, we must have $[\![\mathbb{x}_i.\mathbf{e}]\!]_{\mathsf{cke}} \in \mathcal{R}_{\mathsf{com}}\ \forall i \in \{0,1\}$ and $[\![\mathbf{g}]\!]_{\mathsf{cke}} \in \mathcal{R}_{\mathsf{com}}$.

Finally, we need to prove that $\mathbf{A} \cdot \mathbf{z}'_i \circ \mathbf{B} \cdot \mathbf{z}'_i = \mathbb{x}_i.u \cdot \mathbf{C} \cdot \mathbf{z}'_i + \mathbb{x}_i.\mathbf{e}$ for all $i \in \{0,1\}$ where $\mathbf{z}'_i = (\mathbb{x}_i.\mathsf{pub} \| \mathbb{x}_i.\mathbf{z}_1 \| \ldots \| \mathbb{x}_i.\mathbf{z}_d)$. Note that, since

$$\begin{aligned} &\mathsf{C.Commit}(\mathbb{x}^{(3)}.\mathbf{z}_j, \mathbb{x}^{(3)}.\hat{\mathbf{z}}_j) = \mathbb{x}^{(3)}.\tilde{\mathbf{z}}_j = \mathbb{x}_0.\tilde{\mathbf{z}}_j + \alpha^{(3)} \cdot \mathbb{x}_1.\tilde{\mathbf{z}}_j \\ &= \mathsf{C.Commit}(\mathbb{x}_0.\mathbf{z}_j, \mathbb{x}_0.\hat{\mathbf{z}}_j) + \alpha^{(3)} \cdot \mathsf{C.Commit}(\mathbb{x}_1.\mathbf{z}_j, \mathbb{x}_1.\hat{\mathbf{z}}_j) \\ &= \mathsf{C.Commit}(\mathbb{x}_0.\mathbf{z}_j + \alpha^{(3)} \cdot \mathbb{x}_1.\mathbf{z}_j, \mathbb{x}_0.\hat{\mathbf{z}}_j + \alpha^{(3)} \cdot \mathbb{x}_1.\hat{\mathbf{z}}_j) \end{aligned}$$

and

$$\mathbb{x}^{(3)}.\mathsf{pub} = \mathbb{x}_0.\mathsf{pub} + \alpha^{(3)} \cdot \mathbb{x}_1.\mathsf{pub}.$$

Hence, we must have $\mathbb{x}^{(3)}.\mathbf{z}_j = \mathbb{x}_0.\mathbf{z}_j + \alpha^{(3)} \cdot \mathbb{x}_1.\mathbf{z}_j$ with overwhelming probability due to the binding property of commitment scheme. Thus we have that

$$\mathbf{z}'^{(i)} = \mathbf{z}'_0 + \alpha^{(i)} \cdot \mathbf{z}'_1 \ \forall i \in [3].$$

In addition, since, for each $i \in [3]$, $[\![\mathbb{x}^{(i)}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ and the following equations hold, due to the statement of this lemma:

$$\begin{cases} \mathbf{A} \cdot \mathbf{z}'^{(i)} \circ \mathbf{B} \cdot \mathbf{z}'^{(i)} = \mathbb{x}^{(i)}.u \cdot \mathbf{C} \cdot \mathbf{z}'^{(i)} + \mathbb{x}^{(i)}.\mathbf{e} & \forall i \in [3] \\ \mathbf{z}'^{(i)} = \mathbf{z}'_0 + \alpha^{(i)} \cdot \mathbf{z}'_1 & \forall i \in [3] \\ \mathbb{x}^{(i)}.u = \mathbb{x}_0.u + \alpha^{(i)} \cdot \mathbb{x}_1.u & \forall i \in [3] \\ \mathbb{x}^{(i)}.\mathbf{e} = \mathbb{x}_0.\mathbf{e} + \alpha^{(i)} \cdot \mathbf{g} + (\alpha^{(i)})^2 \cdot \mathbb{x}_1.\mathbf{e} & \forall i \in [3]. \end{cases}$$

From the system above, by replacing $\mathbf{z}'^{(i)}$, $\mathbb{x}^{(i)}$ and $\mathbb{x}^{(i)}.\mathbf{e}$ with $\mathbf{z}'_0 + \alpha^{(i)} \cdot \mathbf{z}'_1$, $\mathbb{x}_0.u + \alpha^{(i)} \cdot \mathbb{x}_1.u$ and $\mathbb{x}_0.\mathbf{e} + \alpha^{(i)} \cdot \mathbf{g} + (\alpha^{(i)})^2 \cdot \mathbb{x}_1.\mathbf{e}$, respectively, in the first equation of the system, then expanding everything and considering that the first equation holds for three distinct values $\alpha^{(i)}$ for all $i \in [3]$, it holds that $\mathbf{A} \cdot \mathbf{z}'_i \circ \mathbf{B} \cdot \mathbf{z}'_i = \mathbb{x}_i.u \cdot \mathbf{C} \cdot \mathbf{z}'_i + \mathbb{x}_i.\mathbf{e}$ for all $i \in \{0, 1\}$ and hence $[\![\mathbb{x}_i]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ for all $i \in \{0, 1\}$ as desired. □

Efficiency of Π_{rr1cs} (Extended). Recall notations in Definition 2. Efficiency of Π_{rr1cs} in Figure 3 as follows:

- *Size of* $[\![\mathbb{x}]\!]_{\mathsf{tck}}$. $\mathcal{O}(m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{c}(m_i) + \mathsf{c}(n))$ which is straightforward from the design in (9).
- *Communication Cost of* Π_{rr1cs}. $\mathsf{c}(n)$ which is because both prover and verifier run Π_{com} to obtain $[\![\mathbf{g}]\!]_{\mathsf{cke}}$.
- *Prover Time of* Π_{rr1cs}. $\mathcal{O}(n + \mathsf{tp}^{\mathsf{c}}(n) + m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tp}^{\mathsf{h}}(m_i) + \mathsf{tp}^{\mathsf{h}}(n))$ analyzed as follows. $\mathcal{O}(n)$ is due to the computation of garb (defined in Section 3.2). $\mathsf{tp}^{\mathsf{c}}(n)$ is for committing to $\mathbf{g}$ from Π_{com}. $\mathcal{O}(m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tp}^{\mathsf{h}}(m_i) + \mathsf{tp}^{\mathsf{h}}(n))$ is for homomorphically evaluating after receiving challenge α.
- *Verifier Time of* Π_{rr1cs}. $\mathcal{O}(\mathsf{tv}^{\mathsf{c}}(n) + m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tv}^{\mathsf{h}}(m_i) + \mathsf{tv}^{\mathsf{h}}(n))$ analyzed as follows. $\mathsf{tv}^{\mathsf{c}}(n)$ is for committing to $\mathbf{g}$ from Π_{com}. $\mathcal{O}(m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tv}^{\mathsf{h}}(m_i) + \mathsf{tv}^{\mathsf{h}}(n))$ is for homomorphically evaluating after sending challenge α.

B.10 Honest-Verifier Zero-Knowledge Argument/Proof of Knowledge

We recall the syntax of honest-verifier zero-knowledge arguments/proofs of knowledge (ZKAoKs/ZKPoKs) in Definition 21 and their security properties in Definitions 22, 23 and 24.

Definition 21 (Syntax of Honest-Verifier ZKAoK/ZKPoK). *Let $\mathcal{I}$ and $\mathcal{Z}$ denote the instance and witness set, respectively. Let $\mathcal{R} \subseteq \mathcal{I} \times \mathcal{Z}$ be a relation. A ZKAoK for $\mathcal{R}$ is a tuple*

$$\mathcal{ZK} = (\mathsf{ZK.Setup}, \mathsf{ZK.Prove})$$

consists of the algorithm $\mathsf{ZK.Setup}$ *and interactive protocol* $\mathsf{ZK.Prove}$*, working as follows:*

$\mathsf{ZK.Setup}(1^\lambda) \to \mathsf{pp}$: *On input a security parameter 1^λ, this PPT algorithm returns a public parameter* pp.

$\mathsf{ZK.Prove}(\mathsf{pp}, I;\ Z) \to \{0, 1\}$ *This is an interactive protocol between prover and verifier, where prover holds an instance-witness pair $(I, Z) \in \mathcal{I} \times \mathcal{Z}$ and verifier holds an instance $Z \in \mathcal{I}$, such that prover tries to convince verifier that he knows $W \in \mathcal{Z}$ satisfying $(Z, W) \in \mathcal{R}$. At the end of the interaction, the verifier outputs a bit $b \in \{0, 1\}$ for deciding whether to accept $(b = 1)$ or reject $(b = 0)$.*

Security of Honest-Verifier ZKAoK/ZKPoK. Let $\mathcal{ZK}$ be a system whose syntax is defined in Definition 21. We now recall the completeness, knowledge soundness, and (honest-verifier) zero-knowledge for $\mathcal{ZK}$ in the following Definitions 22, 23 and 24, respectively.

Definition 22 (Completeness of $\mathcal{ZK}$). *$\mathcal{ZK}$ satisfies completeness if for any $(I;\ Z) \in \mathcal{R}$ it holds that*

$$\Pr\left[b = 1 \middle| \begin{matrix} \mathsf{pp} \leftarrow \mathsf{ZK.Setup}(1^\lambda) \\ b \leftarrow \mathsf{ZK.Prove}(\mathsf{pp}, I;\ Z) \end{matrix}\right] = 1$$

Definition 23 (Knowledge Soundness of $\mathcal{ZK}$). *$\mathcal{ZK}$ satisfies knowledge soundness if for any (PPT) adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$ with rewindable oracle access to $\mathcal{A}$ such that*

$$\Pr\left[b = 1 \wedge (I, Z) \notin \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{ZK.Setup}(1^\lambda), \\ (\mathsf{pp}, I) \leftarrow \mathcal{A}(\mathsf{pp}), \\ Z \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{pp}, Z), \\ b \leftarrow \mathsf{ZK.Prove}(\mathsf{pp}, I; Z) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

We write $\mathcal{E}^{\mathcal{A}}(\mathsf{pp}, I)$ to indicate $\mathcal{E}$ (having oracle access to $\mathcal{A}$) and $\mathcal{A}$, playing the roles of verifier and prover, respectively, to run $\mathsf{CF.Prove}$ on common inputs (pp, I). In addition, $\mathcal{E}$ can rewind $\mathcal{A}$ to any previous state and run other instances of $\mathsf{CF.Prove}$ with different randomness. Finally, if all instances of $\mathsf{ZK.Prove}$ return 1 after interacting with $\mathcal{A}$ with rewinding, $\mathcal{E}$ can return the valid witnesses W satisfying $(I, Z) \in \mathcal{R}$. We call this system an argument of knowledge (AoK) if $\mathcal{A}$ is PPT. Otherwise, if $\mathcal{A}$ is of unbounded computation, we call this system a proof of knowledge (PoK).

Definition 24 (Statistical Honest-Verifier Zero-Knowledge of $\mathcal{ZK}$). *$\mathcal{ZK}$ satisfies statistical (honest-verifier) zero-knowledge if there exists a PPT simulator $\mathcal{S}$ such that for any $(I, Z) \in \mathcal{R}$ and for any (PPT) adversary $\mathcal{A}$, then the two following distributions are indistinguishable from the view of $\mathcal{A}$:*

$$\left\{\mathsf{tr} \,\middle|\, \mathsf{tr} \leftarrow \mathsf{View}(\mathsf{ZK.Prove}(\mathsf{pp}, I;\ Z))\right\} \text{ and } \left\{\mathsf{tr}^* \,\middle|\, \mathsf{tr}^* \leftarrow \mathcal{S}(\mathsf{pp}, I)\right\}.$$

where $\mathsf{tr} \leftarrow \mathsf{View}(\mathsf{ZK.Prove}(\mathsf{pp}, I; Z))$ denotes the public transcript, which contains all the public inputs and exchanged messages between the prover and verifier during the execution of $\mathsf{ZK.Prove}$. We say that $\mathcal{ZK}$ is computationally zero-knowledge if $\mathcal{A}$ is PPT. Otherwise, if $\mathcal{A}$ is of unbounded computation, $\mathcal{ZK}$ is statistically zero-knowledge.

B.11 Lagrange Interpolation

We recall the Lagrange interpolation theorem in the following Theorem 3

Theorem 3 (Lagrange Interpolation). *Let $\mathbb{F}$ be a field. Given a set $\mathcal{X} = \{x_0, x_1, \ldots, x_m\}$ of $m+1$ pairwise distinct values in $\mathbb{F}$, then for any set $\mathcal{Y} = \{y_0, y_1, \ldots, y_m\} \subseteq \mathbb{F}$, there exists an unique polynomial $f(X) \in \mathbb{F}[X]$ of degree at most m satisfying $f(x_i) = y_i$ for all $i \in [0, m]$. Moreover, the exact formula of $f(X)$ is given by*

$$f(X) = \sum_{i=0}^{m} y_i \cdot \ell_i(X)$$

where $\{\ell_0(X), \ell_1(X), \ldots, \ell_m(X)\}$ is the Lagrange basis of $\mathcal{X}$, given by the formula $\ell_i(X) = \prod_{\substack{j \in [0,m] \\ s.t.\ j \neq i}} \frac{X - x_j}{x_i - x_j}$.

B.12 Basic Circuit Satisfiability From Compressed Σ-Protocol Theory

We recall the technique for handling basic circuit satisfiability from compressed Σ-protocol theory (Section 6 in [AC20], adapted from [CDP12]) to ensure statistical (honest-verifier) zero-knowledge.

Let $n \in \mathbb{Z}_+$. Assume that $\mathsf{C}(\mathbf{x}) = 0$ for some $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$ and C is some arithmetic circuit of m multiplication gates. Let $w_1, \ldots, w_m$ be the outputs of those m multiplication gates. Moreover, let $u_i \in \mathbb{F}$ and $v_i \in \mathbb{F}$, for all $i \in [m]$, be the left and right inputs to each multiplication gate such that $u_i \cdot v_i = w_i$.

Assume that $q = |\mathbb{F}|$ is a prime and there is an isomorphism from $\mathbb{Z}_q$ to $\mathbb{F}$. Hence, when saying that $0, \ldots, m \in \mathbb{F}$, these values are understood to be the output of the mentioned isomorphism from inputs $0, \ldots, m \in \mathbb{Z}_q$. By sampling $u_0, v_0 \xleftarrow{\$} \mathbb{F}$, applying Lagrange interpolation we can achieve polynomials $f_u(X), f_v(X) \in \mathbb{F}[X]$ of degrees at most m such that

$$f_u(i) = u_i \text{ and } f_v(i) = v_i$$

for all $i \in [0, m] \subseteq \mathbb{F}$. Specifically, we have a Lagrange basis $\{\ell_0(X), \ldots, \ell_m(X)\} \subseteq \mathbb{F}[X]$ of polynomials in $\mathbb{F}[X]$ of degree m such that

$$f_u(X) = \sum_{i=0}^{m} u_i \cdot \ell_i(X) \text{ and } f_v(X) = \sum_{i=0}^{m} v_i \cdot \ell_i(X).$$

By setting $f(X) := f_u(X) \cdot f_v(X)$ and setting $w_0 := u_0 \cdot v_0$, we see that $f(X)$ is of degree $2m$ and $w_i = f(i) = f_u(i) \cdot f_v(i)$ for all $i \in [0, m]$. Define $w_{m+1} := f(m+1), \ldots, w_{2m} := f(2m)$. We have a Lagrange basis $\{\ell'_0(X), \ldots, \ell'_{2m}(X)\} \subseteq \mathbb{F}[X]$ of polynomials in $\mathbb{F}[X]$ of degree $2m$ such that

$$f(X) = \sum_{i=0}^{2m} w_i \cdot \ell'_i(X).$$

Thus, we can test whether $u_i \cdot v_i = w_i$ for all $i \in [m]$ by testing whether $f_u(X) \cdot f_v(X) = f(X)$. This can be done by sampling $\zeta \xleftarrow{\$} \mathbb{F}$ and check whether $f_u(\zeta) \cdot f_v(\zeta) = f(\zeta)$, by revealing $f_u(\zeta), f_v(\zeta)$ and $f(\zeta)$, with error probability at most $\frac{2m}{|F|}$ according to Schwartz-Zippel lemma [Zip79, Sch80] (see Appendix B.5).

However, when conducting the proof, if ζ is among $[m]$, the values $u_\zeta = f_u(\zeta), v_\zeta = f_v(\zeta)$ and $w_\zeta = f(\zeta)$ must be revealed compromising zero-knowledge or witness indistinguishability of the proof. Moreover, if $\zeta \in \mathbb{F} \setminus [m]$, the values u_i, v_i and w_i, for all $i \in [m]$, are secured. In fact, since u_0 and v_0 are uniformed sampled from $\mathbb{F}$, we know that

$$f_u(\zeta) = u_0 \cdot \ell_0(\zeta) + \sum_{i=1}^{m} u_i \cdot \ell_i(\zeta) \text{ and } f_v(\zeta) = v_0 \cdot \ell_0(\zeta) + \sum_{i=1}^{m} v_i \cdot \ell_i(\zeta).$$

and $\ell_0(\zeta) \neq 0$. Hence, $f_u(\zeta)$ and $f_v(\zeta)$ are uniform in $\mathbb{F}$. Thus, by sampling $\zeta \xleftarrow{\$} \mathbb{F} \setminus [m]$, revealing $f_u(\zeta)$, $f_v(\zeta)$ and $f(\zeta)$ for checking $f_u(\zeta) \cdot f_v(\zeta) = f(\zeta)$ does not compromise u_i, v_i and w_i, for all $i \in [m]$.

Strategy for Making the Proofs/Arguments. To proceed the proofs, [AC20] indicates that

$$u_i = f_u^{(i)}(x_1, \ldots, x_n, u_0, v_0, w_0, \ldots, w_{2m}) \text{ and}$$
$$v_i = f_v^{(i)}(x_1, \ldots, x_n, u_0, v_0, w_0, \ldots, w_{2m}),$$

for all $i \in [m]$, where $f_u^{(i)}(\cdot)$ and $f_v^{(i)}(\cdot)$ are pre-determined affine functions. Hence, for a given challenge ζ, the values $f_u(\zeta)$ and $f_v(\zeta)$ are obtained by affine mappings from

$$(x_1, \ldots, u_0, v_0, w_0, \ldots, w_{2m}, \zeta).$$

Since [AC20] supports protocols for nullity checks of affine maps, we hence can deduce the design of interactive proofs/arguments for basic circuit satisfiability.

Efficiency. According to [AC20], the proof size of the above discussion in $\mathcal{O}(\log m)$ assuming $m \gg n$ while prover time and verifier time are both $\mathcal{O}(m)$.

C Generic CFS $\mathcal{CF}_{\mathsf{gnr}}$ (Extended)

We provide a proof of Theorem 1 in Appendix C.1. In Appendix C.2, we provide a detailed efficiency analysis of $\mathcal{CF}_{\mathsf{gnr}}$ in Figure 5.

C.1 Proof of Theorem 1

We first recall Theorem 1 in the following Theorem 4.

Theorem 4 (Recall of Theorem 1). *If* $\mathsf{CF.Prove}$ *is an (HV)ZKAoK and* $\mathcal{C}$ *is a secure homomorphic commitment scheme, then* $\mathcal{CF}_{\mathsf{gnr}}$ *is correct with correctness error* $\mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp})$*, HVZK and knowledge-sound with soundness error* $\mathcal{O}(1/|\mathbb{F}| + \mathsf{serr}_{\mathsf{prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda))$ *where* $\mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp})$ *and* $\mathsf{serr}_{\mathsf{prf}}(\mathsf{pp})$ *respectively are completeness and soundness error of* $\mathsf{CF.Prove}$.

Before going to the proof, let us define the notations that will be used in the proof. First, recall that Section 5.1, for a witness vector

$$\mathbb{z} = (\mathbb{x}, \mathsf{front}, \mathsf{rear}, \mathbb{x}^\star, \mathbf{s}, \mathbf{a})$$

having the form of (14), we can parse

$$[\![\mathbb{z}]\!]_{\mathsf{pp}} = ([\![\mathbb{x}]\!]_{\mathsf{tck}}, [\![\mathsf{front}]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}]\!]_{\mathsf{ck}'_1}, [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}, [\![\mathbf{s}]\!]_{\mathsf{cks}}, [\![\mathbf{a}]\!]_{\mathsf{cks}})$$

where, for $\tilde{\mathbb{x}}$ and $\tilde{\mathbb{x}}^\star$ of the form (50), and for $\hat{\mathbb{x}}$ and $\hat{\mathbb{x}}^\star$ of the form (51),

$$\begin{aligned}
&[\![\mathbb{x}]\!]_{\mathsf{tck}} = (\mathsf{tck}, \tilde{\mathbb{x}};\ \mathbf{x}, \hat{\mathbb{x}}), && [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'} = (\mathsf{tck}', \tilde{\mathbb{x}}^\star;\ \mathbb{x}^\star, \hat{\mathbb{x}}^\star),\\
&[\![\mathsf{front}]\!]_{\mathsf{ck}'_2} = (\mathsf{ck}'_2, \widetilde{\mathsf{front}};\ \mathsf{front}, \widehat{\mathsf{front}}), && [\![\mathsf{rear}]\!]_{\mathsf{ck}'_1} = (\mathsf{ck}_1, \widetilde{\mathsf{rear}};\ \mathsf{rear}, \widehat{\mathsf{rear}}),\\
&[\![\mathbf{s}]\!]_{\mathsf{cks}} = (\mathsf{cks}, \tilde{\mathsf{s}};\ \mathbf{s}, \hat{\mathsf{s}}), && [\![\mathbf{a}]\!]_{\mathsf{cks}} = (\mathsf{cks}, \tilde{\mathsf{a}};\ \mathbf{a}, \hat{\mathsf{a}}).
\end{aligned}$$

We denote $\tilde{\mathbb{z}}$ to define its public instance part that contains all the public information and commitments of the components in $\mathbb{z}$, i.e.,

$$\tilde{\mathbb{z}} = (\tilde{\mathbb{x}}, \widetilde{\mathsf{front}}, \widetilde{\mathsf{rear}}, \tilde{\mathbb{x}}^\star, \tilde{\mathsf{s}}, \tilde{\mathsf{a}}), \tag{53}$$

and $\hat{\mathbb{z}}$ to define the corresponding randomness used for committing the components in $\mathbb{z}$, i.e.,

$$\hat{\mathbb{z}} = (\hat{\mathbb{x}}, \widehat{\mathsf{front}}, \widehat{\mathsf{rear}}, \hat{\mathbb{x}}^\star, \hat{\mathsf{s}}, \hat{\mathsf{a}}). \tag{54}$$

The notation $[\![\mathbb{z}]\!]_{\mathsf{pp}}$ is the same as (15).

The proof of Theorem 1 (recalled in Theorem 4) is as follows.

Proof (Proof of Theorem 1). The proof follows Lemmas 11, 13 and 14 for correctness, knowledge soundness and HVZK, respectively. □

Correctness of $\mathcal{CF}_{\mathsf{gnr}}$. Correctness follows the following Lemma 11.

Lemma 11 (Correctness of $\mathcal{CF}_{\mathsf{gnr}}$). *$\mathcal{CF}_{\mathsf{gnr}}$ is correct with correctness error $\mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp})$ if $\mathcal{C}$ is a homomorphic and perfectly correct commitment scheme and $\mathsf{CF.Prove}$, for relation $\mathcal{R}^{\mathbb{S},\mathbb{S}'}_{\mathsf{gnr\text{-}inst}}$, is complete with completeness error $\mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp})$ for relation $\mathcal{R}^{\mathbb{S},\mathbb{S}'}_{\mathsf{gnr\text{-}inst}}$.*

Proof. The proof is straightforward. □

Knowledge Soundness of $\mathcal{CF}_{\mathsf{gnr}}$. We first analyze the extraction in each folding by $\Pi_{\mathsf{fold\text{-}gnr}}$ according to Lemma 12. Then, we formalize Lemma 13 the knowledge soundness of $\mathcal{CF}_{\mathsf{gnr}}$ by extracting following a binary-tree-like HS and employing Lemma 12 as a building block. Details are as follows.

Lemma 12 ($(3,3;|\mathbb{F}|,|\mathbb{F}|)$-Special Soundness of $\Pi_{\mathsf{fold\text{-}gnr}}$). *Assume that $\mathcal{C}$, w.r.t. relation $\mathcal{R}_{\mathsf{com}}$, is homomorphic and binding. Assume that, on inputs $\mathbf{p} \in \mathbb{F}^{m'_{\mathsf{pub}}}$, $\mathbf{w} \in \mathbb{F}^{\mathsf{ck}'_3}$, $[\![\mathbb{z}_0]\!]_{\mathsf{pp}}$ and $[\![\mathbb{z}_1]\!]_{\mathsf{pp}}$ of the forms*

$$[\![\mathbb{z}_i]\!]_{\mathsf{pp}} = ([\![\mathbb{x}_i]\!]_{\mathsf{tck}}, [\![\mathsf{front}_i]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}_i]\!]_{\mathsf{ck}'_1}, [\![\mathbb{x}_i^\star]\!]_{\mathsf{tck}'}, [\![\mathbf{s}_i]\!]_{\mathsf{cks}}, [\![\mathbf{a}_i]\!]_{\mathsf{cks}})\ \forall i \in \{0,1\},$$

protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ in Figure 5 are rewinded 9 times following a $(3,3)$-tree of transcripts, w.r.t. challenges $\{\alpha_1^{(i_1)}\}_{i_1\in[3]}$ and $\{\alpha_2^{(i_1,i_2)}\}_{i_1\in[3],i_2\in[3]}$, into

$$[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} = ([\![\mathbb{x}^{(i_1)}]\!]_{\mathsf{tck}}, [\![\mathsf{front}]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}]\!]_{\mathsf{ck}'_1}, [\![(\mathbb{x}^\star)^{(i_1,i_2)}]\!]_{\mathsf{tck}'}, [\![\mathbf{s}]\!]_{\mathsf{cks}}, [\![\mathbf{a}^{(i_1)}]\!]_{\mathsf{cks}}),$$

in the following sense (imitating the folding process of $\Pi_{\mathsf{fold\text{-}gnr}}$ in Figure 5):

- $[\![\mathbb{y}]\!]_{\mathsf{tck}'} = (1, \mathbf{p}, [\![\mathsf{rear}_0]\!]_{\mathsf{ck}'_1}, [\![\mathsf{front}_1]\!]_{\mathsf{ck}'_2}, [\![\mathbf{w}]\!]_{\mathsf{ck}'_3}, [\![\mathbf{0}^{n'}]\!]_{\mathsf{cke}'})$ *(step 2 in Figure 5).*
- $\{\alpha_1^{(i_1)}\}_{i_1\in[3]}$ *are distinct and, for each $i_1 \in [3]$, by running step 4 of Π_{rr1cs},*
  - $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$ *and* $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$ *are folded into* $[\![\mathbb{x}^{(i_1)}]\!]_{\mathsf{tck}}$ *w.r.t.* $\alpha_1^{(i_1)}$, *and*
  - $[\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}$ *and* $[\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'}$ *are folded into* $[\![\mathbb{y}'^{(i_1)}]\!]_{\mathsf{tck}'}$ *w.r.t.* $\alpha_1^{(i_1)}$.

  (The above process is similar to step 4 in Figure 5.)

– *For each* $i_1 \in [3]$, $\{\alpha_2^{(i_1,i_2)}\}_{i_2\in[3]}$ *are distinct, and for each* $i_2 \in [3]$, *by running step 4 of* Π_{rr1cs},
  - $[\![\mathsf{y}'^{(i_1)}]\!]_{\mathsf{tck}'}$ *and* $[\![\mathsf{y}]\!]_{\mathsf{tck}'}$ *are folded into* $[\![(\mathbb{x}^\star)^{(i_1,i_2)}]\!]$ *w.r.t.* $\alpha_2^{(i_1,i_2)}$.

  (The above process is similar to step 8 in Figure 5.)
– $[\![\mathsf{front}]\!]_{\mathsf{ck}'_2} = [\![\mathsf{front}_0]\!]_{\mathsf{ck}'_2}$, $[\![\mathsf{rear}]\!]_{\mathsf{ck}'_1} = [\![\mathsf{rear}_1]\!]_{\mathsf{ck}'_1}$, $[\![\mathbf{s}]\!]_{\mathsf{cks}} = [\![\mathbf{s}_0]\!]_{\mathsf{cks}} + [\![\mathbf{s}_1]\!]_{\mathsf{cks}}$, *and* $[\![\mathbf{a}^{(i_1)}]\!]_{\mathsf{cks}} = [\![\mathbf{a}_0]\!]_{\mathsf{cks}} + \alpha_1^{(i_1)} \cdot [\![\mathbf{a}_1]\!]_{\mathsf{cks}} + (\alpha_1^{(i_1)})^2 \cdot ([\![\mathbf{s}_0]\!]_{\mathsf{cks}} - [\![\mathbf{s}_1]\!]_{\mathsf{cks}})$ *(step 9 in Figure 5).*

Finally, assume that we have all witnesses and randomness of all $[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}}$, *for all* $i_1 \in [3]$ *and all* $i_2 \in [3]$, *such that* $[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$. *Then, we can extract* $\mathbb{z}_0, \mathbb{z}_1$ *and* $\mathbf{w}$ *such that*

$$[\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} \text{ and } (\mathbf{p}, [\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}};\ \mathbf{w}) \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$$

where relations $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ *and* $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ *are defined in* (18) *and* (19), *respectively.*

Proof. We split the proof into two steps: (i) extracting all witnesses of $[\![\mathbb{z}_i]\!]_{\mathsf{pp}}$ $\forall i \in \{0,1\}$ and (ii) showing the existence of $\mathbf{w}$ such that $([\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}};\ \mathbf{w}) \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$. These steps are proceeded as follows.

Extracting All Witnesses and Randomness of $[\![\mathbb{z}_i]\!]_{\mathsf{pp}}$ $\forall i \in \{0,1\}$**.** We proceed as follows:

– *Extracting Witnesses and Randomness of* $[\![\mathsf{front}]\!]_{\mathsf{ck}'_2}$ *and* $[\![\mathsf{rear}]\!]_{\mathsf{ck}'_1}$. Since we have all witnesses and randomness of all $[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}}$ for all $i_1 \in [3]$ and all $i_2 \in [3]$, we know (from the statement of this lemma) that $[\![\mathsf{front}]\!]_{\mathsf{ck}'_2} = [\![\mathsf{front}_0]\!]_{\mathsf{ck}'_2}$ and $[\![\mathsf{rear}]\!]_{\mathsf{ck}'_1} = [\![\mathsf{rear}_1]\!]_{\mathsf{ck}'_1}$. Therefore, we also achieve all witnesses and randomness of $[\![\mathsf{front}]\!]_{\mathsf{ck}'_2}$ and $[\![\mathsf{rear}]\!]_{\mathsf{ck}'_1}$ according to Remark 6.
  As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

$$[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} \implies ([\![\mathsf{front}]\!]_{\mathsf{ck}_1}, [\![\mathsf{rear}]\!]_{\mathsf{ck}_2} \in \mathcal{R}_{\mathsf{com}})$$

  according to (18). Hence, we deduce that $[\![\mathsf{front}]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}]\!]_{\mathsf{ck}'_1} \in \mathcal{R}_{\mathsf{com}}$.
– *Extracting Witnesses and Randomness of* $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$ *and* $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$. As there are 3 distinct challenges $\{\alpha_1^{(i_1)}\}_{i_1\in[3]}$, and, for each $i_1 \in [3]$, $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$ and $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$ are folded into $[\![\mathbb{x}^{(i_1)}]\!]_{\mathsf{tck}}$ w.r.t. $\alpha_1^{(i_1)}$. Moreover, we also have witnesses and randomness of $[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}}$, for all $i_1 \in [3]$ and all $i_2 \in [3]$, containing those of $[\![\mathbb{x}^{(i_1)}]\!]_{\mathsf{tck}}$, for all $i_1 \in [3]$. We hence can apply Lemma 1 (recalled in Lemma 10) to extract witnesses and randomness of $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$ and $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$.
  As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

$$[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} \implies [\![\mathbb{x}^{(i_1)}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}.$$

  Hence, by Lemma 1, we deduce that $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}, [\![\mathbb{x}_1]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$.
– *Extracting Witnesses and Randomness of* $[\![\mathsf{y}]\!]_{\mathsf{tck}'}$, $[\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}$ *and* $[\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'}$. For all $i_1 \in [3]$, since there are 3 distinct challenges $\{\alpha_2^{(i_1,i_2)}\}_{i_2\in[3]}$, and, for each $i_2 \in [3]$, $[\![\mathsf{y}'^{(i_1)}]\!]_{\mathsf{tck}'}$ and $[\![\mathsf{y}]\!]_{\mathsf{tck}'}$ are folded into $[\![(\mathbb{x}^\star)^{(i_1,i_2)}]\!]$ w.r.t. $\alpha_2^{(i_1,i_2)}$. Moreover, we also have all witnesses and randomness of all $\{[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}}\}_{i_1\in[3],i_2\in[3]}$ containing those of $\{[\![(\mathbb{x}^\star)^{(i_1,i_2)}]\!]\}_{i_1\in[3],i_2\in[3]}$. For each $i_1 \in [3]$, we hence can apply Lemma 1 (recalled in Lemma 10), we can extract all witnesses and randomness of $[\![\mathsf{y}'^{(i_1)}]\!]_{\mathsf{tck}'}$ and $[\![\mathsf{y}]\!]_{\mathsf{tck}'}$. Moreover, from the statement of this lemma, for all $i_1 \in [3]$, $[\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}$ and $[\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'}$ are folded into $[\![\mathsf{y}'^{(i_1)}]\!]_{\mathsf{tck}'}$ w.r.t. $\alpha_1^{(i_1)}$. Since $\{\alpha_1^{(i_1)}\}_{i_1\in[3]}$ are distinct, we again apply Lemma 1 to extract all witnesses and randomness of $[\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}$ and $[\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'}$.
  As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

$$[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} \implies [\![(\mathbb{x}^\star)^{(i_1,i_2)}]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}.$$

  By Lemma 1, $[\![\mathsf{y}'^{(i_1)}]\!]_{\mathsf{tck}'}$ and $[\![\mathsf{y}]\!]_{\mathsf{tck}'}$ satisfy $[\![\mathsf{y}'^{(i_1)}]\!]_{\mathsf{tck}'}, [\![\mathsf{y}]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}$. Again, by Lemma 1, $[\![\mathbb{x}_0^\star]\!]_{\mathsf{tck}'}, [\![\mathbb{x}_1^\star]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}$ as desired.
– *Extracting Witnesses and Randomnesses of* $[\![\mathbf{a}_0]\!]_{\mathsf{cks}}$, $[\![\mathbf{a}_1]\!]_{\mathsf{cks}}$, $[\![\mathbf{s}_0]\!]_{\mathsf{cks}} - [\![\mathbf{s}_1]\!]_{\mathsf{cks}}$. For each $i_1 \in [3]$,

$$[\![\mathbf{a}^{(i_1)}]\!]_{\mathsf{cks}} = [\![\mathbf{a}_0]\!]_{\mathsf{cks}} + \alpha_1^{(i_1)} \cdot [\![\mathbf{a}_1]\!]_{\mathsf{cks}} + (\alpha_1^{(i_1)})^2 \cdot ([\![\mathbf{s}_0]\!]_{\mathsf{cks}} - [\![\mathbf{s}_1]\!]_{\mathsf{cks}})$$

has $\alpha_1^{(i_1)}$ of degree 3. Moreover, we have distinct $\{\alpha_1^{(i_1)}\}_{i_1\in[3]}$ and all witnesses and randomness of $[\![\mathbb{z}^{(i_1,i_2)}]\!]_{i_1\in[3],i_2\in[3]}$ containing those of $\{[\![\mathbf{a}^{(i_1)}]\!]_{\mathsf{cks}}\}_{i_1\in[3]}$. Therefore, we can extract all witnesses and randomnesses of $[\![\mathbf{a}_0]\!]_{\mathsf{cks}}$, $[\![\mathbf{a}_1]\!]_{\mathsf{cks}}$ and $[\![\mathbf{s}']\!]_{\mathsf{cks}} = [\![\mathbf{s}_0]\!]_{\mathsf{cks}} - [\![\mathbf{s}_1]\!]_{\mathsf{cks}}$ by solving the system of equations w.r.t. all $\{\alpha_1^{(i_1)}\}_{i_1\in[3]}$.
As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

$$\begin{aligned}[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'} &\implies [\![(\mathbf{a})^{(i_1)}]\!]_{\mathsf{cks}} \in \mathcal{R}_{\mathsf{com}} \\ &\implies [\![\mathbf{a}_0]\!]_{\mathsf{cks}}, [\![\mathbf{a}_1]\!]_{\mathsf{cks}}, [\![\mathbf{s}']\!]_{\mathsf{cks}} \in \mathcal{R}_{\mathsf{com}}.\end{aligned}$$

– *Extracting Witnesses and Randomness of* $[\![\mathbf{s}_0]\!]_{\mathsf{cks}}$ *and* $[\![\mathbf{s}_1]\!]_{\mathsf{cks}}$. We have $[\![\mathbf{s}]\!]_{\mathsf{cks}} = [\![\mathbf{s}_0]\!]_{\mathsf{cks}} + [\![\mathbf{s}_1]\!]_{\mathsf{cks}}$ in the problem statement, and $[\![\mathbf{s}']\!]_{\mathsf{cks}} = [\![\mathbf{s}_0]\!]_{\mathsf{cks}} - [\![\mathbf{s}_1]\!]_{\mathsf{cks}}$. Moreover, we also have all witnesses and randomness of $[\![\mathbf{s}]\!]_{\mathsf{cks}}$ and $[\![\mathbf{s}']\!]_{\mathsf{cks}}$. Therefore, by solving equations, we can obtain all witnesses and randomness of $[\![\mathbf{s}_0]\!]_{\mathsf{cks}}$ and $[\![\mathbf{s}_1]\!]_{\mathsf{cks}}$.
As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

$$[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'} \implies [\![\mathbf{s}]\!]_{\mathsf{cks}} = [\![\mathbf{s}_0]\!]_{\mathsf{cks}} + [\![\mathbf{s}_1]\!]_{\mathsf{cks}} \in \mathcal{R}_{\mathsf{com}}.$$

Since $[\![\mathbf{s}']\!]_{\mathsf{cks}} = [\![\mathbf{s}_0]\!]_{\mathsf{cks}} - [\![\mathbf{s}_1]\!]_{\mathsf{cks}} \in \mathcal{R}_{\mathsf{com}}$ as proved above, hence the extracted $[\![\mathbf{s}_0]\!]_{\mathsf{cks}}, [\![\mathbf{s}_1]\!]_{\mathsf{cks}} \in \mathcal{R}_{\mathsf{com}}$.

Thus, we already achieve all witnesses and randomness of $[\![\mathbb{z}_i]\!]_{\mathsf{pp}}$ for all $i \in \{0, 1\}$. By collecting those extracted witnesses and randomness, we achieve

$$[\![\mathbb{z}_i]\!]_{\mathsf{pp}} = ([\![\mathbb{x}_i]\!]_{\mathsf{tck}}, [\![\mathsf{front}_i]\!]_{\mathsf{ck}'_2}, [\![\mathsf{rear}_i]\!]_{\mathsf{ck}'_1}, [\![\mathbb{x}_i^\star]\!]_{\mathsf{tck}'}, [\![\mathbf{s}_i]\!]_{\mathsf{cks}}, [\![\mathbf{a}_i]\!]_{\mathsf{cks}}) \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}.$$

Existence of w s.t. $([\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}}, \mathbf{p};\ \mathbf{w}) \in \mathcal{R}_{\mathsf{gnr\text{-}cond}}^{\mathfrak{S}'}$**.** Notice that, from the statement of this lemma, $[\![\mathbb{y}]\!]_{\mathsf{tck}'} = (1, \mathbf{p}, [\![\mathsf{rear}_0]\!]_{\mathsf{ck}'_1}, [\![\mathsf{front}_1]\!]_{\mathsf{ck}'_2}, [\![\mathbf{w}]\!]_{\mathsf{ck}'_3}, [\![\mathbf{0}^{n'}]\!]_{\mathsf{cke}'})$. Since we have all witnesses and randomness of $[\![\mathbb{y}]\!]_{\mathsf{tck}'}$.

As proved above, $[\![\mathbb{y}]\!]_{\mathsf{tck}'} \in \mathcal{R}_{\mathsf{rr1cs}}^{\mathfrak{S}'}$. By the binding property of $\mathcal{C}$, it implies that $\mathbb{y}.u = 1$, $\mathbb{y}.\mathbf{e} = \mathbf{0}^{n'}$ and

$$\mathbf{A}' \cdot \mathbf{c}' \circ \mathbf{B}' \cdot \mathbf{c}' = \mathbf{C}' \cdot \mathbf{c}'$$

where $\mathbf{c}' = (\mathbf{p} \| \mathsf{rear}_0 \| \mathsf{front}_1 \| \mathbf{w})$. Hence, it holds that the witness $\mathbf{w}$ in $\mathbb{y}$ satisfies $([\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}}, \mathbf{p};\ \mathbf{w}) \in \mathcal{R}_{\mathsf{gnr\text{-}cond}}^{\mathfrak{S}'}$. □

Lemma 13 (Knowledge Soundness of $\mathcal{CF}_{\mathsf{gnr}}$). *$\mathcal{CF}_{\mathsf{gnr}}$ is knowledge-sound if $\mathcal{C}$ is an additively homomorphic and binding commitment scheme, protocol* $\mathsf{CF.Prove}$ *of $\mathcal{CF}_{\mathsf{gnr}}$, for relation $\mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$, is knowledge-sound. Moreover, $\mathcal{CF}_{\mathsf{gnr}}$ has soundness error*

$$\mathcal{O}\left(\frac{1}{|\mathbb{F}|} + \mathsf{serr}_{\mathsf{prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda)\right)$$

where $\mathsf{serr}_{\mathsf{prf}}(\mathsf{pp})$ *is the soundness error of* $\mathsf{CF.Prove}$*, w.r.t. relation $\mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$.*

Proof. The proof is straightforward with Lemma 12 as a building block. Recall from Definition 6 that $\Pi_{\mathcal{CF}_{\mathsf{gnr}}}$ is the combined protocol that sequentially runs $\mathsf{CF.Fold}$ (folding) and then $\mathsf{CF.Prove}$ (proving) where $\mathsf{CF.Fold}$ and $\mathsf{CF.Prove}$ are protocols of $\mathcal{CF}_{\mathsf{gnr}}$.

As from Lemma 12, by rewinding $\mathsf{CF.Fold}$ 9 times following a $(3, 3)$-tree of transcripts. For each of such rewindings, we additionally run $\mathsf{CF.Prove}$. Since $\mathsf{CF.Prove}$ is knowledge-sound, we can run the extractor of $\mathsf{CF.Prove}$ to extract the satisfying witnesses (corresponding to the folded instance) for each rewinding. Finally, having enough transcript from the $(3, 3)$-tree pf acceptance transcripts, we can use Lemma 12 to extract the witnesses corresponding to the to-be-folded instances (before running $\mathsf{CF.Fold}$). Thus, we conclude that $\mathcal{CF}_{\mathsf{gnr}}$ is knowledge-sound.
Soundness Error. As we see from Lemma 12, we need a $(3, 3)$-tree of accepted transcripts in order to successfully extract. Each transcript corresponds to a challenge $(\alpha_1^{(i_1)}, \alpha_2^{(i_1,i_2)})$. Hence, we imply that the soundness error is at most

$$\frac{2}{|\mathbb{F}|} + \left(1 - \frac{2}{|\mathbb{F}|}\right) \cdot \frac{2}{|\mathbb{F}|} = \mathcal{O}\left(\frac{1}{|\mathbb{F}|}\right).$$

Hence, we conclude that the soundness error of $\Pi_{\mathcal{CF}_{\mathsf{gnr}}}$ is

$$\mathcal{O}\left(\frac{1}{|\mathbb{F}|} + \mathsf{serr}_{\mathsf{prf}}(\lambda) + \mathsf{negl}(\lambda)\right)$$

where $\mathsf{serr}_{\mathsf{prf}}(\lambda)$ is the soundness error of $\mathsf{CF.Prove}$ and $\mathsf{negl}(\lambda)$ is some negligible probability for the cheating prover to break the binding property of the underlying commitment scheme. □

HVZK of $\mathcal{CF}_{\mathsf{gnr}}$. HVZK follows the following Lemma 14.

Lemma 14 (HVZK of $\mathcal{CF}_{\mathsf{gnr}}$). *CFS $\mathcal{CF}_{\mathsf{gnr}}$ is HVZK if $\mathcal{C}$ is an additively homomorphic and hiding commitment scheme and* $\mathsf{CF.Prove}$*, for relation $\mathcal{R}^{\mathbb{S},\mathbb{S}'}_{\mathsf{gnr\text{-}inst}}$, is HVZK.*

Proof. Assume that we consider folding $[\![\mathsf{z}_0]\!]_{\mathsf{pp}}$ and $[\![\mathsf{z}_1]\!]_{\mathsf{pp}}$ into $[\![\mathsf{z}]\!]$. Let $\tilde{\mathsf{z}}_0$, $\tilde{\mathsf{z}}_1$ and $\tilde{\mathsf{z}}$ respectively are the commitments (of the form (53)) in $[\![\mathsf{z}_0]\!]_{\mathsf{pp}}$, $[\![\mathsf{z}_1]\!]_{\mathsf{pp}}$ and $[\![\mathsf{z}]\!]$. Let $\hat{\mathsf{z}}_0, \hat{\mathsf{z}}_1$ and $\hat{\mathsf{z}}$ respectively are the randomness (of the form (54)) in $[\![\mathsf{z}_0]\!]_{\mathsf{pp}}$, $[\![\mathsf{z}_1]\!]_{\mathsf{pp}}$ and $[\![\mathsf{z}]\!]$.

Let tr be the transcript

$$\mathsf{tr} \leftarrow \mathsf{View}(\Pi_{\mathcal{CF}_{\mathsf{gnr}}}(\mathsf{pp}, [\![\mathsf{z}_0]\!]_{\mathsf{pp}}, [\![\mathsf{z}_1]\!]_{\mathsf{pp}}, \mathbf{p};\ \mathbf{w}))$$

as defined in (12) w.r.t. CFS $\mathcal{CF}_{\mathsf{gnr}}$. Notice that tr contains the transcript of the execution of $\mathsf{CF.Fold}$, the final instance $\tilde{\mathsf{z}}$, and the transcript of $\mathsf{CF.Prove}$. Hence, we can write

$$\mathsf{tr} = (\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathsf{z}} \| \mathsf{tr}_{\mathsf{prf}})$$

where $\mathsf{tr}_{\mathsf{fold}}$ is the transcript of the execution of $\mathsf{CF.Fold}$ and $\mathsf{tr}_{\mathsf{prf}}$ is the transcript of $\mathsf{CF.Prove}$ when $\tilde{\mathsf{z}}$ is achieved after running $\mathsf{CF.Fold}$.

Assume that $\mathcal{S}_{\mathsf{prove}}$ is the simulator for $\mathsf{CF.Prove}$ since $\mathsf{CF.Prove}$ is a ZKAoK (or ZKAPoK). To show that $\mathcal{CF}_{\mathsf{gnr}}$ satisfies HVZK, we construct a simulator $\mathcal{S}$ which outputs a simulated transcript $\mathsf{tr}^\star$ indistinguishable from tr. Before explicitly constructing $\mathcal{S}$, we make some observations as follows. First, by calling $\mathcal{S}_{\mathsf{prove}}$ on inputs $(\mathsf{pp}, \tilde{\mathsf{z}})$, we obtain transcript $\mathsf{tr}'_{\mathsf{prf}}$. Then, we can form

$$\mathsf{tr}' = \left(\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathsf{z}} \| \mathsf{tr}'_{\mathsf{prf}}\right).$$

By HVZK of $\mathsf{CF.Prove}$, we see that tr and tr' are indistinguishable, i.e., the prefixes or tr and tr' w.r.t. $(\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathsf{z}})$ are identical while the suffixes $\mathsf{tr}_{\mathsf{prf}}$ and $\mathsf{tr}'_{\mathsf{prf}}$ are indistinguishable.

We now show how to construct

$$\mathsf{tr}^\star = (\mathsf{tr}^\star_{\mathsf{fold}} \| \tilde{\mathsf{z}}^\star \| \mathsf{tr}^\star_{\mathsf{prf}}),$$

namely, the simulated transcript by simulating not only $\mathsf{CF.Prove}$, but also $\mathsf{CF.Fold}$. Then, we will show that tr' and $\mathsf{tr}^\star$ are indistinguishable. Consequently, tr and $\mathsf{tr}^\star$ are indistinguishable implying HVZK of $\mathcal{CF}_{\mathsf{gnr}}$.

Constructing $\mathcal{S}$ for $\mathcal{CF}_{\mathsf{gnr}}$. We show a construction of simulator $\mathcal{S}$ for $\mathcal{CF}_{\mathsf{gnr}}$. Notice that, for each folding, prover needs to send $(\tilde{\mathsf{g}}, \tilde{\mathsf{w}}, \tilde{\mathsf{g}}_1)$, namely, commitments in $[\![\mathbf{g}]\!]_{\mathsf{cke}}, [\![\mathbf{w}]\!]_{\mathsf{ck}'_3}$ and $[\![\mathbf{g}_1]\!]_{\mathsf{cke}'}$ as in Figure 5, before receiving challenge α_1, and $\tilde{\mathsf{g}}_2$, namely, commitment in $[\![\mathbf{g}_2]\!]_{\mathsf{cke}'}$ in Figure 5, before receiving α_2. Therefore, the simulator simply commits to zero vectors with randomness sampled appropriately, namely, following the correct distribution of randomness sampling, to obtain those *dummy* commitments $\tilde{\mathsf{g}}, \tilde{\mathsf{w}}, \tilde{\mathsf{g}}_1$ and $\tilde{\mathsf{g}}_2$. By the hiding property of $\mathcal{C}$, these dummy commitments are indistinguishable from the real ones in the real transcripts. Hence, when simulating, simulator $\mathcal{S}$ of $\mathcal{CF}_{\mathsf{gnr}}$ only computes dummy commitments and sends them to the verifier to obtain $(\mathsf{tr}^\star_{\mathsf{fold}} \| \tilde{\mathsf{z}}^\star)$. Then, it calls simulator $\mathcal{S}_{\mathsf{prove}}$, on input $(\mathsf{pp}, \tilde{\mathsf{z}}^\star)$, of $\mathsf{CF.Prove}$ to get the simulated transcript $\mathsf{tr}^\star_{\mathsf{prf}}$. Finally, form the simulated transcript $\mathsf{tr}^\star = (\mathsf{tr}^\star_{\mathsf{fold}} \| \tilde{\mathsf{z}}^\star \| \mathsf{tr}^\star_{\mathsf{prf}})$.

We now analyze how $\mathsf{tr}^\star$ is indistinguishable from tr'. We first notice that $(\mathsf{tr}^\star_{\mathsf{fold}} \| \tilde{\mathsf{z}}^\star)$ and $(\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathsf{z}})$ are indistinguishable according to the hiding property of commitment scheme $\mathcal{C}$. Then, $(\mathsf{pp}, \tilde{\mathsf{z}}^\star)$ is passed to $\mathcal{S}_{\mathsf{prove}}$ for producing simulated proof.

Finally, we show that $\mathsf{tr}'_{\mathsf{prf}}$ is indistinguishable from $\mathsf{tr}^\star_{\mathsf{prf}}$. Indeed, if $\mathcal{S}_{\mathsf{prove}}$ is unable to produce $\mathsf{tr}^\star_{\mathsf{prf}}$ indistinguishable from $\mathsf{tr}'_{\mathsf{prf}}$, then there exists a distinguisher $\mathcal{A}$ that can distinguish between $\mathsf{tr}'_{\mathsf{prf}}$ and $\mathsf{tr}^\star_{\mathsf{prf}}$ with non-negligible probability. We assume that $\mathcal{A}$ output 0 if the input is $\mathsf{tr}'_{\mathsf{prf}}$ and 1 otherwise. Now we can construct a distinguisher $\mathcal{A}'$ to distinguish $(\mathsf{tr}^\star_{\mathsf{fold}} \| \tilde{\mathsf{z}}^\star)$ and $(\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathsf{z}})$ as

follows: $\mathcal{A}'$, on input $\ddot{\mathsf{tr}}$ (which is either $(\mathsf{tr}^{\star}_{\mathsf{fold}}\|\tilde{z}^{\star})$ or $(\mathsf{tr}_{\mathsf{fold}}\|\tilde{z})$), employs $\mathcal{S}_{\mathsf{prove}}$ to produce and forward $\ddot{\mathsf{tr}}_{\mathsf{prf}}$ to $\mathcal{A}$. Then $\mathcal{A}'$ determines $\ddot{\mathsf{tr}}$ as $(\mathsf{tr}_{\mathsf{fold}}\|\tilde{z})$ if $\mathcal{A}$ outputs 0. Otherwise, $\mathcal{A}$ determines $\ddot{\mathsf{tr}}$ as $(\mathsf{tr}^{\star}_{\mathsf{fold}}\|\tilde{z}^{\star})$. One can see that, by using $\mathcal{A}$, this distinguisher $\mathcal{A}'$ can correctly distinguish $(\mathsf{tr}^{\star}_{\mathsf{fold}}\|\tilde{z}^{\star})$ and $(\mathsf{tr}_{\mathsf{fold}}\|\tilde{z})$ with non-negligible probability as well, contradicting the hiding property of $\mathcal{C}$. Hence, $\mathsf{tr}^{\star}_{\mathsf{prf}}$ and $\mathsf{tr}'_{\mathsf{prf}}$ are indistinguishable as well.

Therefore, $\mathsf{tr}^{\star}$ and tr' are indistinguishable implying indistinguishability between $\mathsf{tr}^{\star}$ and tr. Thus, HVZK is guaranteed. □

Remark 15. A variant of Lemma 14 is witness indistinguishability of $\mathcal{CF}_{\mathsf{gnr}}$ which can be formalized that $\mathcal{CF}_{\mathsf{gnr}}$ is (statistically) witness-indistinguishable if $\mathcal{C}$ is an additively homomorphic and (statistically) hiding commitment scheme and $\mathsf{CF.Prove}$, for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$, is (statistically) witness-indistinguishable.

The proof of the above fact is straightforward since the two transcripts corresponding to the two witnesses mainly contain commitments and the final proof which ensures witness indistinguishability by the hiding property of commitments and witness indistinguishability of the final proof.

C.2 Efficiency of $\mathcal{CF}_{\mathsf{gnr}}$ (Extended)

Recall the notations in Definition 2. The efficiency of $\mathcal{CF}_{\mathsf{gnr}}$ is as follows:

- *Size of Public Instance in* $[\![\mathbb{z}]\!]_{\mathsf{pp}}$ *in* (15). $\mathcal{O}(m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{c}(m_i) + \mathsf{c}(n) + m'_{\mathsf{pub}} + \sum_{i\in[3]} \mathsf{c}(m'_i) + \mathsf{c}(n') + \mathsf{c}(s))$ which is obvious from the design in Section 5.1.
- *Communication Cost of* $\mathsf{CF.Fold}$. $\mathcal{O}(\mathsf{c}(n) + \mathsf{c}(n'))$ analyzed as follows. Notice that both parties need to run three rR1CS foldings in which they need to obtain $[\![\mathbf{g}]\!]_{\mathsf{cke}}$, $[\![\mathbf{g}_1]\!]_{\mathsf{cke}'}$ and $[\![\mathbf{g}_2]\!]_{\mathsf{cke}'}$ where $|\mathbf{g}| = n$ and $|\mathbf{g}_1| = |\mathbf{g}_2| = n'$.
- *Prover Time of* $\mathsf{CF.Fold}$. $\mathcal{O}(n + \mathsf{tp}^{\mathsf{c}}(n) + m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tp}^{\mathsf{h}}(m_i) + \mathsf{tp}^{\mathsf{h}}(n) + n' + \mathsf{tp}^{\mathsf{c}}(n') + m'_{\mathsf{pub}} + \sum_{i\in[3]} \mathsf{tp}^{\mathsf{h}}(m'_i) + \mathsf{tp}^{\mathsf{h}}(n'))$ analyzed as follows. As from above, the prover needs to perform three foldings, which can be implied from the analysis of efficiency from Section 3.2.
- *Verifier Time of* $\mathsf{CF.Fold}$. $\mathcal{O}(\mathsf{tv}^{\mathsf{c}}(n) + m_{\mathsf{pub}} + \sum_{i\in[d]} \mathsf{tv}^{\mathsf{h}}(m_i) + \mathsf{tv}^{\mathsf{h}}(n) + \mathsf{tv}^{\mathsf{c}}(n') + m'_{\mathsf{pub}} + \sum_{i\in[3]} \mathsf{tv}^{\mathsf{h}}(m'_i) + \mathsf{tv}^{\mathsf{h}}(n'))$ which, as form above, follows the analysis of efficiency from Section 3.2.
- *Prover and Verifier Time of* $\mathsf{CF.Prove}$. This depends on the employed ZKAoK.

D RAMenPaSTA (Extended)

In Appendix D.1, we provide proof of Lemma 2. In Appendix D.2, we provide the proof of Theorem 2.

D.1 Proof of Lemma 2

We first recall Lemma 2 in the following Lemma 15.

Lemma 15 (Recall of Lemma 2). *Let* $\gamma, \delta, \tau, \omega, \chi, \psi \xleftarrow{\$} \mathbb{F}$ *and* $(\mathsf{mul}_j)_{j\in[T]} \in \mathbb{F}$ *be prepared in advance. Let* $\mathsf{plkiv}'_j = \mathsf{mul}_j \cdot (\chi + \langle (j\|\mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]}\rangle)^{-1}$ *for* $j \in [T]$. *Then,*

- (22) *implies* (26), (27) *and* (28) *with probability* $1 - \mathcal{O}((N+T)/|\mathbb{F}|)$ *due to division by zero.*
- *And* (26), (27) *and* (28) *together imply* (22) *with soundness error at most* $\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$.

Proof (Proof of Lemma 2 (recalled in Lemma 15)). We divide the proof into the following two cases:

The Case "(22) $\Longrightarrow$ (26), (27), (28)": This holds with probability $1 - \mathcal{O}((N+T)/|\mathbb{F}|)$ due to the potential division by zero where, from Section 6.1,

- (23) holds with probability for division by zero bounded by $\mathcal{O}(N/|\mathbb{F}|)$, and
- (24) holds with probability for division by zero bounded by $\mathcal{O}((N+T)/|\mathbb{F}|)$.

Hence, by union bound, the total probability for division by zero is bounded by $\mathcal{O}((N+T)/|\mathbb{F}|)$.
The Case "(26), (27), (28) $\implies$ (22)": Indeed, it is trivial that (26) and (27) imply that

$$\mathsf{C}_{\mathsf{mem}}(i, \mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1}, \mathsf{macs}_i, \mathsf{macs}'_i) = 1 \ \forall i \in [2, N]$$
$$\mathsf{C}^{\gamma,\delta}_{\mathsf{plk}}(\mathsf{plkst}_i, \overline{\mathsf{val}}_i, (\mathsf{pc}_i \| \mathsf{macs}_i), \mathsf{auxplk}_i) = 1 \ \forall i \in [N]$$

with probability 1, meaning that the soundness error of these conditions is 0.

Next, since (27) we have that

$$\begin{cases} \mathsf{miv}_i = (\tau + \langle \mathsf{macs}_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \ \forall i \in [N], \\ \mathsf{miv}'_i = (\tau + \langle \mathsf{macs}'_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \ \forall i \in [N] \end{cases} \tag{55}$$

By substituting (55) into (28), it holds that

$$\sum_{i\in[N]} (\tau + \langle \mathsf{macs}_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} = \sum_{i\in[N]} (\tau + \langle \mathsf{macs}'_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \tag{56}$$

By Lemma (5) (see Appendix B.4), with randomly chosen $\tau, \omega \in \mathbb{F}$, since (56) holds, we have

$$\sum_{i\in[N]} (X + \langle \mathsf{macs}_i, (Y^k)_{k\in[0,3]} \rangle)^{-1} = \sum_{i\in[N]} (X + \langle \mathsf{macs}'_i, (Y^k)_{k\in[0,3]} \rangle)^{-1}$$

in $\mathbb{F}(X, Y)$ with probability at least $1 - \mathcal{O}(N/|\mathbb{F}|)$ as each tuple has constant size equal to 4. By Lemma (7) (see Appendix B.4), this also implies that $(\mathsf{macs}_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}'_i)_{i\in[N]}$ with probability at least $1 - \mathcal{O}(N/|\mathbb{F}|)$. Therefore, the soundness error of the *tuple permutation* condition from (22) is at most $\mathcal{O}(N/|\mathbb{F}|)$.

By (27), we also have that

$$\begin{cases} \mathsf{plkiv}_i = (\chi + \langle (\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1} \ \forall i \in [N], \\ \mathsf{plkiv}'_j = \mathsf{mul}_j \cdot (\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1} \ \forall j \in [T] \end{cases} \tag{57}$$

Again, by substituting (57) into (28), it holds that

$$\sum_{i\in[N]} (\chi + \langle (\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1} = \sum_{j\in[T]} \mathsf{mul}_j \cdot (\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle)^{-1} \tag{58}$$

For some $\mathsf{mul}_1, \ldots, \mathsf{mul}_T \in \mathbb{F}$. Similarly, by Lemmas 7 and 8, we see that $(\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i)_{i\in[N]} \subseteq (j \| \mathsf{plkst}'_j)_{j\in[T]}$ with probability at least $1 - \mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$. Therefore, the soundness error of the *tuple lookup* condition from (22) is at most $\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$.

Finally, the above arguments imply that (22) holds with soundness error at most

$$\mathcal{O}(N/|\mathbb{F}|) + \mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|) = \mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$$

as desired. □

D.2 Proof of Theorem 2

For readability, we recall Theorem 2 in the following Theorem 5.

Theorem 5 (Recall of Theorem 2). *If $\mathcal{C}$ is secure homomorphic commitment scheme and $\Pi_{\mathsf{RP\text{-}prf}}$ is an HVZKAoK for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ then Π_{RP} is an HVZKAoK for relation $\mathcal{R}_{\mathsf{ram}}$ in (20) with completeness error $\mathcal{O}((N+T)/|\mathbb{F}| + \mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp}))$ and soundness error*

$$\mathcal{O}\left(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}| + \mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda)\right)$$

where $\mathsf{cerr}_{\mathsf{prf}}(\mathsf{pp})$ and $\mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp})$ are respectively completeness and soundness error of $\Pi_{\mathsf{RP\text{-}prf}}$ and $\mathsf{negl}(\lambda)$ is the negligible probability for cheating prover to break the binding of the underlying commitment scheme.

Proof (Proof of Theorem 2 (Recalled in Theorem 5)). The proof follows Lemmas 16, 17 and 18. Specifically, Lemmas 16, 18 and 18 are for correctness, knowledge soundness and HVZK of Π_{RP}, respectively. Together, they imply the proof of Theorem 2. □

Correctness of Π_{RP}. Correctness follows Lemma 16.

Lemma 16 (Completeness of Π_{RP}). *If $\mathcal{C}$ is perfectly correct and $\Pi_{\mathsf{RP\text{-}prf}}$, for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$, is complete with completeness error $\mathsf{cerr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp})$, then Π_{RP} is complete with probability $1-\mathcal{O}((N+T)/|\mathbb{F}|+\mathsf{cerr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}))$.*

Proof. Notice that $\mathsf{CF.Fold}$ is correct with correctness error 0. However, Lemma 2 indicates that the proof for both permutation lookup and tuple lookup incurs a total completeness error $\mathcal{O}((N+T)/|\mathbb{F}|)$. Hence, in total, completeness error of Π_{RP} is bounded by $\mathcal{O}((N+T)/|\mathbb{F}|+\mathsf{cerr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}))$. □

Knowledge Soundness of Π_{RP}. Knowledge soundness follows Lemma 17 below. Before discussing the idea for the proof of Lemma 17, we recall the hierarchical structure HS (see Definition 1). We first classify the nodes in HS by depths according to the following Definition 25. Then, we prove knowledge soundness of Π_{RP} by Lemma 17.

Definition 25 (Depths in Hierarchical Structures). *Let $N \in \mathbb{Z}_+$ and HS be defined as in Definition 1. We denote the depth of each node by a function $\mathsf{depth} : \mathsf{HS} \to \mathbb{Z}_+$ as follows:*

- *The root node $(0, N)$ has depth 0, i.e., $\mathsf{depth}(0, N) = 0$.*
- *For any pair of nodes (l, j) and (j, r) satisfying $(l, j), (j, r), (l, r) \in \mathsf{HS}$ (i.e., (l, r) is the direct parent node of (l, j) and (j, r) in HS), we denote by $\mathsf{depth}(l, j) = \mathsf{depth}(j, r) = \mathsf{depth}(l, r) + 1$.*

The height H of HS is the maximum depth of nodes in HS, i.e.,

$$H = \max\left\{\mathsf{depth}(l, r) \,\middle|\, (l, r) \in \mathsf{HS}\right\}.$$

Remark 16. Notice that Π_{RP} in Figure 7 folds N instance-witness pairs, in step 7 following a hierarchical HS. Let H be the height of HS. By classifying the instance-witness pairs by depths as in Definition 25, we see that, for a depth $d \in [H]$, the process that folds all instance-witness pairs at depth-d to the above depths, i.e., $d-1$ to 0, and then prove $[\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$, is a ZKAoK (or ZKPoK), denoted by Π_d, of instance-witness pairs at depth d such that (i) all instance-witness pairs satisfy $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ (see (18)) and, (ii) for each folding at depth d, the condition for such a folding also satisfy $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ (see (19)).

Lemma 17 (Knowledge Soundness of Π_{RP}). *If $\mathcal{C}$ is an additively homomorphic and binding and $\Pi_{\mathsf{RP\text{-}prf}}$, for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$, is knowledge-sound, then Π_{RP} is knowledge-sound. Moreover, Π_{RP} has soundness error*

$$\mathcal{O}\left(\frac{n_{\mathsf{plk}}\cdot(N+T)}{|\mathbb{F}|}+\mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp})+\mathsf{negl}(\lambda)\right)$$

where $\mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp})$ is the soundness error of $\Pi_{\mathsf{RP\text{-}prf}}$ and $\mathsf{negl}(\lambda)$ is the negligible probability for cheating prover to break the binding of the underlying commitment scheme.

Proof. The proof is split into three smaller parts as follows. Firstly, we provide a strategy for the extractor to extract all witnesses corresponding to instance-witness pairs at all nodes of HS. Secondly, from those extracted, we extract components for RAM programs that follow the specification of Lemma 2. Third and eventually, we analyze the soundness error of Π_{RP}.

Rewinding Strategy. According to Remark 16, since for each depth $d \in [0, H]$, there exists a ZKAoK (or ZKPoK) for all relations and folding conditions at depth d, it seems that we can directly apply Lemma 13 for extraction. However, at depth d, there may be more than one folding (involving more than two instance-witness pairs). Therefore, directly applying Lemma 13 may not be applicable. Below, we present another way of extracting w.r.t. HS.

The idea for extraction is as follows. Since Π_{RP} in Figure 7 folds instance-witness pairs $\{[\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}\}_{i\in[N]}$ (see step 7) following the topological order of a hierarchical structure HS (see Definition 1), we

can construct the extractor for Π_{RP} to extract the witnesses corresponding to nodes of HS depth-by-depth from the top to bottom. In particular, let H be the height of HS. We construct the extractors $\{\mathcal{E}_d\}_{d\in[0,H]}$ such that, for $d \in [0,H]$, $\mathcal{E}_d$ extracts the witnesses of instance-witnesses pairs $\{[\![\mathbb{z}_{lr}]\!]_{\mathsf{pp}}\}_{(l,r)\in\mathsf{HS},\mathsf{depth}(l,r)\leq d}$ at depth-d nodes of HS. In other words, $\mathcal{E}_d$ is the extractor for ZKAoK (or ZKPoK) Π_d. The extractions of $\{\mathcal{E}_d\}_{d\in[0,H]}$ are recursively constructed as follows.
(i) Extractor $\mathcal{E}_0$. At depth 0, there is only the root node $(0,N)$. The witness corresponding to $[\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}}$ is extractable by extractor of $\mathsf{CF.Prove}$ (since $\mathsf{CF.Prove}$ is knowledge-sound). Here, $\mathcal{E}_0$ is the extractor of $\mathsf{CF.Prove}$. Recalled that $\mathsf{CF.Prove}$ of Π_{RP} is realized by $\Pi_{\mathsf{RP\text{-}prf}}$ for satisfaction of relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ (see Figure 7).
(ii) Extractor $\mathcal{E}_d$ for $d \in [H]$. We now construct extractor $\mathcal{E}_d$ that employs $\mathcal{E}_{d-1}$ as a sub-rountine. Assume that there are $2 \cdot K_d$ instance-witness pairs

$$\{[\![\mathbb{z}_{lr}]\!]_{\mathsf{pp}}\}_{(l,r)\in\mathsf{HS},\mathsf{depth}(l,r)=d}$$

at depth-d nodes such that we can partition into K_d pairs of these instance-witnesses pairs as follows. For each pair of these instance-witnesses pairs, when running $\mathsf{CF.Fold}$, we obtain a new instance-witness pair at depth $d-1$ of HS. Since Π_{RP} employs $\mathsf{CF.Fold}$ from $\mathcal{CF}_{\mathsf{gnr}}$ as a black box for folding, according to Lemma 12, with a $(3,3)$-tree of acceptance transcripts for each of these pairs of instance-witness pairs, we can extract its corresponding witness. Hence, we construct the extractor $\mathcal{E}_d$ as follows. This extractor first samples i.i.d. K_d $(3,3)$-tree of challenges. For $k \in [K_d]$, we denote by $\mathcal{T}^{(d,k)} = (\alpha_1^{(d,k,i_1)}, \alpha_2^{(d,k,i_1,i_2)})_{i_1,i_2\in[3]}$ to be the k-th $(3,3)$-tree of challenges. Then, we instruct $\mathcal{E}_d$ to rewind 9 as follows. For each $(i_1,i_2) \in [3]\times[3]$, $\mathcal{E}_d$ works as follows:

- For all $k \in [K_d]$, $\mathcal{E}_d$ communicates with prover to fold the k-th pair of instance-witness pairs at depth-d nodes by running $\mathsf{CF.Fold}$ (realized by protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ in Figure 5) with α_1 and α_2 (in $\Pi_{\mathsf{fold\text{-}gnr}}$) respectively are replaced by $\alpha_1^{(d,k,i_1)}$ and $\alpha_2^{(d,k,i_1,i_2)}$. Hence, all folded instance-witness pairs are at depth-$(d-1)$ nodes of HS.
- Run $\mathcal{E}_{d-1}$ to extract witnesses for those instance-witness pairs at depth-$(d-1)$ nodes.

Hence, by such a rewinding strategy, we obtain K_d $(3,3)$-tree of accepted transcripts. By Lemma 12, it is sufficient for extracting witnesses of instance-witness pairs and those for the conditions when folding at depth-d nodes.

Eventually, we see that $\mathcal{E}_H$ extracts all witnesses and conditions for folding following hierarchical structure HS. We are now ready to discuss in detail the extraction of the entire execution trace of RAM programs from RAMenPaSTA.

Extracting the Execution Trace of RAM Program. Since $\Pi_{\mathsf{RP\text{-}prf}}$ is knowledge-sound, it is also a knowledge-sound proof for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ because it is implied by relation $\Pi_{\mathsf{RP\text{-}prf}}$ as specified in (32). By using $\mathcal{E}_H$, we can extract $\mathbb{z}_{lr}$, for all $(l,r) \in \mathsf{HS}$ s.t. $[\![\mathbb{z}_{lr}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$. By $\Pi_{\mathsf{RP\text{-}prf}}$ for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ (see (32)), it also holds that $[\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ since satisfaction of $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ implies satisfaction of $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$. Moreover, according to step 7 in Figure 7, all of the conditions between to-be-folded instance-witness pairs also hold, i.e.,

$$\bigwedge_{\substack{l,j,r \text{ s.t.}\\ (l,j),(j,r)\in\mathsf{HS}}} \Big(([\![\mathbb{z}_{lj}]\!]_{\mathsf{pp}}, [\![\mathbb{z}_{jr}]\!]_{\mathsf{pp}}, (1,j+1);\ \mathbf{w}_k) \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}\Big)$$

implying $([\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}, [\![\mathbb{z}_{i(i+1)}]\!]_{\mathsf{pp}}, (1,i+1);\ \mathbf{w}_{i+1}) \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ for all $i \in [N-1]$.

We also can extract $(\mathsf{mul}_j)_{j\in[T]}$ and randomness $(\hat{\mathsf{m}}_j)_{j\in[T]}$ by the extractor of $\Pi_{\mathsf{RP\text{-}prf}}$ for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ such that

$$(\mathsf{ckm}_j, \tilde{\mathsf{m}}_j;\ \mathsf{mul}_j, \hat{\mathsf{m}}_j) \in \mathcal{R}_{\mathsf{com}}\ j \in [T].$$

(i) *Extracting Components for Single Computation Steps.* Recall that, from Remark 9, we can parse $\mathfrak{S} = (\mathbf{A},\mathbf{B},\mathbf{C})$ and $\mathbf{A},\mathbf{B}$ and $\mathbf{C}$ are public matrices that can be publicly determined from challenges $\gamma,\delta,\tau,\omega,\chi,\psi$. Since $[\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$, it captures the fact that

$$[\![\mathbb{x}_{(i-1)i}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$$

where $[\![\mathbb{x}_{(i-1)i}]\!]_{\mathsf{tck}}$ is a component in $[\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}$ according to the form (15). Hence, for each $i \in [N]$, by following Section 3.2, we can define $u_i = \mathbb{x}_{(i-1)i}.u$, $\mathbf{z}_i = (\mathbb{x}_{(i-1)i}.\mathsf{pub} \| \mathbb{x}_{(i-1)i}.\mathbf{z}_1 \| \dots \| \mathbb{x}_{(i-1)i}.\mathbf{z}_5)$ and $\mathbf{e}_i = \mathbb{x}_{(i-1)i}.\mathbf{e}$ such that

$$\mathbf{A} \cdot \mathbf{z}_i \circ \mathbf{B} \cdot \mathbf{z}_i = u_i \cdot \mathbf{C} \cdot \mathbf{z}_i + \mathbf{e}_i.$$

According to protocol Π_{RP} in Figure 7, it holds that $\mathbb{x}_{(i-1)i}.\mathsf{pub} = 1$ and $\mathbb{x}_{(i-1)i}.\tilde{\mathsf{e}} = \mathsf{C.Commit}_{\mathsf{cke}}(\mathbf{0}^n, 0)$, i.e., $\mathbb{x}_{(i-1)i}.\tilde{\mathsf{e}}$ is commitment in $[\![\mathbb{x}_{(i-1)i}.\mathbf{e}]\!]_{\mathsf{cke}}$, since these public values are computed by verifier. By the binding property of commitment scheme $\mathcal{C}$, it should hold that $\mathbb{x}_{(i-1)i}.\mathbf{e} = \mathbf{0}^n$. Hence, we see that

$$\mathbf{A} \cdot \mathbf{z}'_i \circ \mathbf{B} \cdot \mathbf{z}'_i = \mathbf{C} \cdot \mathbf{z}'_i \tag{59}$$

where $\mathbf{z}'_i = (1 \| \mathbb{x}_{(i-1)i}.\mathbf{z}_1 \| \dots \| \mathbb{x}_{(i-1)i}.\mathbf{z}_5)$. By following (29) in Section 6.2, we parse

$$\begin{aligned}
\mathbb{x}_{(i-1)i}.\mathbf{z}_1 &= (\overline{\mathsf{pc}}_i \| \overline{\mathsf{val}}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i),\\
\mathbb{x}_{(i-1)i}.\mathbf{z}_2 &= (\mathsf{pc}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i),\\
\mathbb{x}_{(i-1)i}.\mathbf{z}_3 &= \mathsf{plkst}_i,\\
\mathbb{x}_{(i-1)i}.\mathbf{z}_4 &= \mathsf{aux}_i, \text{ and}\\
\mathbb{x}_{(i-1)i}.\mathbf{z}_5 &= (\mathsf{miv}_i \| \mathsf{miv}'_i \| \mathsf{plkiv}_i)
\end{aligned}$$

Here, the fact that both $\mathbb{x}_{(i-1)i}.\mathbf{z}_1$ and $\mathbb{x}_{(i-1)i}.\mathbf{z}_2$ contain the same $(\mathsf{macs}_i \| \mathsf{macs}'_i)$ is due to Definition 8. Hence, we have

$$\begin{aligned}
\mathbf{z}'_i = (1 \| \overline{\mathsf{pc}}_i \| \overline{\mathsf{val}}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i \| \mathsf{pc}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i \| \\
\mathsf{plkst}_i \| \mathsf{aux}_i \| \mathsf{miv}_i \| \mathsf{miv}'_i \| \mathsf{plkiv}_i)
\end{aligned} \tag{60}$$

for all $i \in [N]$. By Definition 8, we hence deduce that (27) holds.
Since $\mathbf{A}, \mathbf{B}, \mathbf{C}$ realize the testing of hidden evaluation of hidden circuits by employing PLONK's arithmetization w.r.t. plkst_i, for all $i \in [N]$, by employing challenges γ and δ, according to Appendix B.3, the error probability for this test is at most $\frac{N(n_{\mathsf{plk}} - 4n_{\mathsf{gate}})}{|\mathbb{F}|} \leq \frac{N \cdot n_{\mathsf{plk}}}{|\mathbb{F}|}$, by using union bound over the N computation steps, where n_{gate} is the number of gates in circuit corresponding to plkst_i for $i \in [N]$.

(ii) *Extracting Witnesses for Conditions.* Parse $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$, where $\mathbf{A}', \mathbf{B}', \mathbf{C}'$ are public matrices defined in Definition 9. For each $i \in [2, N]$, recall that our extracted witnesses $\mathbb{z}_{(i-1)i}$ and auxiliary witnesses $\mathbf{w}_i$ satisfy

$$([\![\mathbb{z}_{(i-2)(i-1)}]\!]_{\mathsf{pp}}, [\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}, (1, i);\ \mathbf{w}_i) \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}},$$

it holds that

$$\mathbf{A}' \cdot \mathbf{c}_i \circ \mathbf{B}' \cdot \mathbf{c}_i = \mathbf{C}' \cdot \mathbf{c}_i$$

where $\mathbf{c}_i = ((1, i) \| \mathsf{rear}_{(i-2)(i-1)} \| \mathsf{front}_{(i-1)i} \| \mathbf{w}_i)$. However, since $\widetilde{\mathsf{rear}}_{(i-2)(i-1)} = \mathbb{x}_{(i-2)(i-1)}.\tilde{\mathbf{z}}_2$ and $\widetilde{\mathsf{front}}_{(i-1)i} = \mathbb{x}_{(i-1)i}.\tilde{\mathbf{z}}_1$, where $\widetilde{\mathsf{rear}}_{(i-2)(i-1)}$, $\mathbb{x}_{(i-2)(i-1)}.\tilde{\mathbf{z}}_2$, $\widetilde{\mathsf{front}}_{(i-1)i}$, $\mathbb{x}_{(i-1)i}.\tilde{\mathbf{z}}_1$ are commitments in $[\![\mathsf{rear}_{(i-2)(i-1)}]\!]_{\mathsf{ck}_2}$, $[\![\mathbb{x}_{(i-2)(i-1)}.\mathbf{z}_2]\!]_{\mathsf{ck}_2}$, $[\![\mathsf{front}_{(i-1)i}]\!]_{\mathsf{ck}_1}$, $[\![\mathbb{x}_{(i-1)i}.\mathbf{z}_1]\!]_{\mathsf{ck}_1}$, respectively, according to the settings

$$\begin{aligned}
&[\![\mathsf{front}_{(i-1)i}]\!]_{\mathsf{ck}_1} := [\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1},\\
&[\![\mathsf{rear}_{(i-1)i}]\!]_{\mathsf{ck}_2} := [\![\mathbf{z}_{i2}]\!]_{\mathsf{ck}_2},\\
&[\![\mathbb{x}_{(i-1)i}]\!]_{\mathsf{tck}} := (1, 1, [\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1}, \dots, [\![\mathbf{z}_{i5}]\!]_{\mathsf{ck}_5}, [\![\mathbf{0}^n]\!]_{\mathsf{cke}})
\end{aligned}$$

as in step 6 of Π_{RP}. By the binding property of $\mathcal{C}$, it holds that

$$\mathbf{c}_i = (1 \| (1, i) \| \mathsf{pc}_{i-1} \| \mathsf{macs}_{i-1} \| \mathsf{macs}'_{i-1} \| \overline{\mathsf{pc}}_i \| \overline{\mathsf{val}}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i).$$

By Definition 9, we hence deduce that $\overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}$, $\overline{\mathsf{val}}_i = \mathsf{val}_{i-1}$ and (26) holds.

(iii) *Constraints for Tuple Permutation and Tuple Lookup.* For all $l, j, r \in \mathsf{HS}_{\mathsf{fold}}$ (where $\mathsf{HS}_{\mathsf{fold}}$ is defined in Definition 1), our extracted witnesses satisfy $\mathbf{s}_{lr} = \mathbf{s}_{lj} + \mathbf{s}_{jr}$ according to the proof of Lemma 13, we deduce that

$$\sum_{i\in[N]} \mathbf{s}_{(i-1)i} = \mathbf{s}_{0N}.$$

According to step 6 in Figure 7, since verifier has computed $\tilde{\mathsf{s}}_{(i-1)i} = \mathbb{x}_{(i-1)i}.\tilde{\mathbf{z}}_5$, by the binding property of $\mathcal{C}$, it holds that

$$\mathbf{s}_{(i-1)i} = (\mathsf{miv}_i \| \mathsf{miv}'_i \| \mathsf{plkiv}_i).$$

By parsing $\mathbf{s}_{0N} = (\mathsf{miv} \| \mathsf{miv}' \| \mathsf{plkiv})$, we see that

$$\begin{cases} \sum_{i\in[N]} \mathsf{miv}_i = \mathsf{miv}, \\ \sum_{i\in[N]} \mathsf{miv}'_i = \mathsf{miv}', \\ \sum_{i\in[N]} \mathsf{plkiv}_i = \mathsf{plkiv}. \end{cases}$$

Since $\Pi_{\mathsf{RP\text{-}prf}}$ for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ (see (32)) guarantees that

$$\mathsf{plkiv} = \sum_{j\in[T]} \frac{\mathsf{mul}_j}{\chi + \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle} \text{ and } \mathsf{miv} = \mathsf{miv}',$$

we hence see that

$$\begin{cases} \sum_{i\in[N]} \mathsf{plkiv}_i = \sum_{j\in[T]} \mathsf{mul}_j \cdot \left(\chi + \left\langle (j \| \mathsf{plkst}'_j), \left(\psi^k\right)_{k=0}^{n_{\mathsf{plk}}} \right\rangle\right)^{-1}, \text{ and} \\ \sum_{i\in[N]} \mathsf{miv}_i = \sum_{i\in[N]} \mathsf{miv}'_i. \end{cases}$$

By the fact that $[\![\mathsf{front}_{(i-1)i}]\!]_{\mathsf{ck}_1} = [\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1} = [\![\mathbb{x}_{(i-1)i}.\mathbf{z}_1]\!]_{\mathsf{ck}_1}$ (step (6) in Figure 7) and Definition 8, we know that

$$\begin{cases} \mathsf{miv}_i = (\tau + \langle \mathsf{macs}_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \ \forall i \in [N], \\ \mathsf{miv}'_i = (\tau + \langle \mathsf{macs}'_i, (\omega^k)_{k\in[0,3]} \rangle)^{-1} \ \forall i \in [N] \end{cases}$$

and

$$\mathsf{plkiv}_i = \left(\chi + \left\langle (\mathsf{pc}_{i-1} \| \mathsf{plkst}_i), \left(\psi^k\right)_{k=0}^{n_{\mathsf{plk}}} \right\rangle\right)^{-1}.$$

Notice that the above systems imply that (28) is satisfied.

From (i), (ii) and (iii) above, (27), (26) and (28) are satisfied. By Lemma 2, they all imply that (22) is satisfied with soundness error $\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$. Hence, we have already proved the correct execution of RAM programs. Next, we prove the correct input and output of the RAM program.

Correct Input and Output. Parse

$$\mathsf{front}_{0N} = (\overline{\mathsf{pc}} \| \overline{\mathsf{val}} \| \mathsf{macs} \| \mathsf{macs}') \text{ and } \mathsf{rear}_{0N} = (\mathsf{pc} \| \mathsf{macs} \| \mathsf{macs}').$$

Notice that relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$ (see (32)) enforces $\overline{\mathsf{pc}} = 1, \overline{\mathsf{val}} = \mathsf{val}_{\mathsf{in}}$, $\mathsf{val} = \mathsf{val}_{\mathsf{out}}$ and $[\![\mathsf{val}_{\mathsf{in}}]\!]_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}}$, by the binding property of $\mathcal{C}$. By the knowledge soundness of $\Pi_{\mathsf{RP\text{-}prf}}$ (for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{RP\text{-}prf}}$), we can extract $\mathsf{val}_{\mathsf{in}}$, and we are guaranteed that those for input and output of the N-step RAM program are correct.

Analysis of Soundness Error. Recall that (22) implies the correct execution of N instructions with soundness error $\mathcal{O}(N \cdot n_{\mathsf{plk}}/|\mathbb{F}|)$ (see Section 2.6 and Appendix B.3). By Lemma 2, (27), (26) and (28) imply that (22) is satisfied with soundness error $\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$. Hence, in total, (27), (26) and (28), with guaranteed correct input and output for RAM program, imply satisfaction of $\mathcal{R}_{\mathsf{ram}}$ with total soundness error

$$\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|). \tag{61}$$

However, since we prove (27), (26) and (28) by Π_{RP} with underlying folding scheme, we need to analyze another layer of soundness error. Recall that we split HS into $H+1$ depths (from 0 to H)

according to Definition 25. Here, as discussed above, for $d \in [N]$, there are $2 \cdot K_d$ nodes in depth d, while depth 0 has exactly one root node. As from above, we can extract witnesses of depth-d nodes, for $d \in [N]$, by those from depth-$(d-1)$ nodes in the spirit of K_d $(3,3)$-tree of accepted transcripts, incurring soundness error $\frac{K_d \cdot 4}{\mathbb{F}}$ for all K_d foldings. Define P_d as the soundness error for the validity of those instance-witness pairs at depth-d nodes. Then, we can see that

$$\begin{aligned}
P_0 &= \mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}), \\
P_d &= P_{d-1} + (1 - P_{d-1}) \cdot \frac{K_d \cdot 4}{\mathbb{F}} + \mathsf{negl}(\lambda) \\
&\leq P_{d-1} + \frac{K_d \cdot 4}{\mathbb{F}} + \mathsf{negl}(\lambda) \\
&\leq \mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}) + \sum_{i \in [d]} \frac{K_i \cdot 4}{\mathbb{F}} + \mathsf{negl}(\lambda)
\end{aligned}$$

where $\mathsf{negl}(\lambda)$ can be seen as the ability to break the binding of the underlying commitment scheme. It can be seen that P_H is the soundness error of Π_{RP} which is bounded by

$$\mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}) + \sum_{i \in [H]} \frac{K_i \cdot 4}{\mathbb{F}} + \mathsf{negl}(\lambda) = \mathcal{O}\left(\mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}) + \frac{N}{|\mathbb{F}|} + \mathsf{negl}(\lambda)\right)$$

by the observation that $\sum_{i \in [H]} K_i = \mathcal{O}(N)$.

Combining with soundness error $\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|)$ in (61), we see that the total soundness error, from folding following HS and from those implying (61), is

$$\begin{aligned}
&\mathcal{O}\left(\frac{n_{\mathsf{plk}} \cdot (N+T)}{|\mathbb{F}|}\right) + \left(1 - \mathcal{O}\left(\frac{N \cdot n_{\mathsf{plk}}}{|\mathbb{F}|}\right)\right) \cdot \mathcal{O}\left(\mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}) + \frac{N}{|\mathbb{F}|} + \mathsf{negl}(\lambda)\right) \\
&= \mathcal{O}\left(\frac{n_{\mathsf{plk}} \cdot (N+T)}{|\mathbb{F}|} + \mathsf{serr}_{\mathsf{RP\text{-}prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda)\right).
\end{aligned}$$

We thus conclude the proof. □

HVZK of Π_{RP}. HVZK of Π_{RP} follows Lemma 18.

Lemma 18 (HVZK of Π_{RP}). *If $\mathcal{C}$ is an additively homomorphic and hiding commitment scheme and $\Pi_{\mathsf{RP\text{-}prf}}$, for relation $\mathcal{R}^{\mathbb{G},\mathbb{G}'}_{\mathsf{RP\text{-}prf}}$, is HVZK, then Π_{RP} is HVZK.*

Proof. We prove recursively as follows. According to Remark 16, for any $d \in [0, H]$, the process of (i) folding all instance-witness pairs from depths d to 0 of HS and (ii) proving by $\Pi_{\mathsf{RP\text{-}prf}}$ is a ZKAoK (or ZKPoK) Π_d for those instance-witness pairs and conditions of foldings at depth d. By this observation, we can prove HVZK by induction as follows:

- At depth 0, since Π_0, which is also $\Pi_{\mathsf{RP\text{-}prf}}$, is a ZKAoK (or ZKPoK). Hence, HVZK at depth 0 is guaranteed.
- For $d \in [H]$, assume by inductive hypothesis that Π_{d-1} is HVZK. Then, we can show that Π_d is also HVZK. We can apply a similar proving strategy from the proof of Lemma 14. The difference from the proof of Lemma 14 is that we consider folding K_d pairs of instance-witness pairs at depth-d nodes of HS rather than a single folding as in Lemma 14.

We thus conclude the proof. □