RAMenPaSTA: Parallelizable Scalable Transparent Arguments of Knowledge for RAM Programs

Khai Hanh Tang[1], Minh Pham[2,3], and Chan Nam Ngo[4]

[1] Nanyang Technological University, 50 Nanyang Ave, Singapore 639798, Singapore
[2] Orochi Network LLC, Suite 305, Griffith Corporate Centre, Beachmont, Kingstown, Saint Vincent and the Grenadines
[3] Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam
[4] Independent

Abstract. Incremental Verifiable Computation (IVC) allows a prover to prove to a verifier the correct execution of a sequential computation. Recent works focus on improving the universality and efficiency of IVC Schemes, which can be categorized into Accumulation and Folding-based IVCs with Folding-based ones being more efficient (due to their deferred proof generation until the final step). Unfortunately, both approaches satisfy only heuristic security as they model the Random Oracle (RO) as a circuit in their non-constant depth recursive composition of the base Scheme. Such drawback is two-fold: to connect the consecutive execution step the RO is recursively modeled as a circuit during the folding or the accumulating process, and again in the final SNARK wrapper circuit (a common practice in Folding-based IVCs). As a consequence, cycle of curves are usually necessary for efficient instantiation of such IVC schemes.

We revisit this problem, with a focus on the Folding-based IVCs due to their efficiency, and propose the detachment of RO invocation from the folding circuit. We can instead accumulate such invocations, yielding the so-called Conditional Folding (CF) Scheme to overcome the first drawback. One can consider our CF Scheme a hybrid Folding-Accumulation Scheme with provable security. We provide a non-trivial practical construction for our CF scheme that is natively parallelizable, which offers great efficiency. We rigorously prove the security of our CF scheme (also for the case of folding in parallel; and our scheme can be made non-interactive using Fiat-Shamir). Our CF scheme is generic and does not require trusted setup. It can be adapted to construct the first IVC for RAM programs, i.e. Parallelizable Scalable Transparent Arguments of Knowledge for RAM Programs that we dub RAMenPaSTA, that can be used to build zero-knowledge virtual machines (zkVMs). Both our CF Scheme and RAMenPaSTA can be of independent research interests.

Keywords: Incremental Verifiable Computation · Folding Scheme · ROM · Provable Security · RAM Programs · Transparent Setup · zkVMs

Table of Contents

1 Introduction

Incremental Verifiable Computation (IVC) [Val08] allows a powerful server to continuously execute a sequential and delegated (possibly private) computation on some inputs (with some from the client and also potentially some server-owned private inputs) that results in some output and later cryptographically prove to the weak client that the claimed output is the result of the delegated task.

IVC Applications. IVC is particularly relevant in the context of scaling the blockchain such as Bitcoin or Ethereum by executing the transactions in the off-chain layer Layer-2 (L2) and only use the on-chain layer Layer-1 (L1, i.e. the Bitcoin or Ethereum blockchain itself) as a settlement layer, i.e. to commit only the claimed new state of the transactions applied on the last state.[5]. In this case, the powerful server is called the operator of the L2 and the weak client is every node in L1. Yet, scalable verifiable computation is not the only application of IVC, there are more fundamental applications such as Verifiable Delay Function [BBBF18] or Verifiable Replicated State-Machines such as TinyRAM [BCG$^+$13].[6]

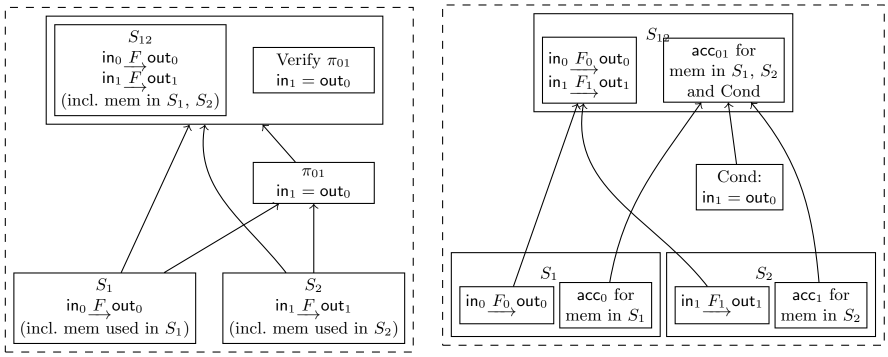
IVC Realizations. IVC can be realized in two different paradigms, the first one is called Proof-Carrying-Data (PCD) [CT10] where mutually distrusted parties can perform pipelined efficiently verifiable private computation. PCD can be achieved via composing Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) [BCI$^+$13, GGPR13] or similar succinct primitive (such as Accumulation Scheme, AS [BCMS20]) in a recursive manner, i.e. at the n-th step of the computation the statement is that there exists a valid statement that the $(n-1)$-th step has been executed correctly and the n-th step is carried out consistently as well (using the outcome from the $(n-1)$-th step and with the operation of the n-th-step). In fact, PCD generalizes IVC as the powerful prover can be distributed as opposed to the central operator of IVC. A more recent approach to realize IVC is to use Folding Scheme (FS) such as Nova [KST22] that avoid using SNARKs (except for the final proof). FS can be considered as a relaxed primitive of SNARK and AS, i.e. instead of accumulating the verification of the (expensive) arguments intermediate steps, FS accumulates the instances and defers the verification until the very end, hence the name Folding Scheme where NP instances are folded to a single one to be verified only once at the end, resulting in significantly lower prover cost. Specifically, a Folding Scheme is defined for an NP relation and it allows two mutually distrusted prover and verifier to compute a single folded instance from an N-sized NP instances. In addition, the prover obtains a folded witness from the witnesses of these instances. FS has gained tremendous traction due to its practical impact in significantly improving the prover cost of IVC and resulted in a long line of published research work [KST22, KS22, KS23, ZGGX23, LXZ$^+$24].

Problematic modeling of RO as a circuit and Conditional Folding Scheme for IVC. All the aforementioned work are proven secure in the Random Oracle Model (ROM) which is problematic due to its heuristic security when instantiated with a cryptographic hash function. This drawback is two fold: (i) for "connecting" the input and output of two consecutive steps while folding a RO invocation is necessary (as shown in the left diagram of Fig. 1), such "connection" is recursively formed along the execution of the program which cause non-constant depth recursion heuristic; and (ii) the final proof is "wrapped" in a SNARK for succinctness. Thus, folding/accumulation-based IVC fails to obtain provable security. Interestingly, this problem has been precedented in the past in [KVV16], as pointed out by [FMNV22, BDD20]. In [KVV16], the authors compute the result of a RO based commitment inside an MPC protocol. However, this means the RO itself must be computed by MPC, which is impossible since ROs cannot be represented by a circuit. This problem is also recently formalized in [CCS22, CGSY23] in which the authors propose an orthogonal fix to the problem via by proposing special RO with additional properties. However, such approach could suffer from inefficiency since these ROs still need to be instantiated with hash functions (SHA-256, Poseidon [GKR$^+$21]) or hardware tokens [CCS22], making the circuits for realizing such ROs become expensive.

While the final wrap is somewhat unavoidable if one aims for succinct verifier, the recursion heuristic in the connections is not the case. Such heuristic can be avoided meaning we can obtain a construction of provable security and can be made non-interactive using the Fiat-Shamir heuristic.

[5] https://ethereum.org/en/developers/docs/scaling/zk-rollups/

[6] TinyRAM just facilitates verification, not a VRSM, but still can be used as one.


S_{12}
$\mathsf{in}_0 \xrightarrow{F} \mathsf{out}_0$
$\mathsf{in}_1 \xrightarrow{F} \mathsf{out}_1$
(incl. mem in S_1, S_2)
Verify π_{01}
$\mathsf{in}_1 = \mathsf{out}_0$
π_{01}
$\mathsf{in}_1 = \mathsf{out}_0$
S_1
$\mathsf{in}_0 \xrightarrow{F} \mathsf{out}_0$
(incl. mem used in S_1)
S_2
$\mathsf{in}_1 \xrightarrow{F} \mathsf{out}_1$
(incl. mem used in S_2)
S_{12}
$\mathsf{in}_0 \xrightarrow{F_0} \mathsf{out}_0$
$\mathsf{in}_1 \xrightarrow{F_1} \mathsf{out}_1$
acc_{01} for mem in S_1, S_2 and Cond
Cond: $\mathsf{in}_1 = \mathsf{out}_0$
S_1
$\mathsf{in}_0 \xrightarrow{F_0} \mathsf{out}_0$
acc_0 for mem in S_1
S_2
$\mathsf{in}_1 \xrightarrow{F_1} \mathsf{out}_1$
acc_1 for mem in S_2

In IVC we want to show that in step S_1, using input in_0 and the state transition function F we obtain the output out_0 and in step S_2, using input in_1 and the state transition function F we obtain the output out_1. It is also necessary to connect S_1 and S_2 by showing $\mathsf{in}_1 = \mathsf{out}_0$.
Folding-based IVC, such as Nova and Nova-ish IVC, has problematic of recursive invocation of RO when verifying π_{01} when connecting S_1 and S_2 for IVC; whereas in Conditional-Folding-based IVC, such as RAMenPaSTA, we accumulate instead and deferred verification of π_{01} till the very end. As such, our CF scheme can be considered as a new paradigm dubbed "Hybrid Folding-Accumulation Scheme".

Fig. 1. Folding Scheme (left) vs Conditional Folding Scheme (right) for IVC.

Observe that the problem is due to the verification of a SNARK proof (e.g. π_{01} in Fig. 1) being encoded as a part of the circuit that requires invoking a query to the RO. We can 'detach' the verification part out of the circuit, as shown in Fig. 1 (right). Instead such proof of the connection can be accumulated and verified at the end and outside of the folding scheme. We call this new scheme Conditional Folding Scheme (CF). One can consider our scheme follow a hybrid folding-accumulate paradigm which is itself of independent research interest.

Non-trivial design and generic construction of CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$ and parallelizable IVC. The proposed fix, seemingly simple (just the detachment of the verification of the connection), yet the construction requires careful design in the forms of instance and witness pairs for folding computation steps. Such careful design allows us to define and construct efficient parallelizable IVC. Concretely, we achieve the following advantages. First, since our construction is generic, with homomorphic commitment schemes and ZKAoKs as building blocks, we can apply any ZKAoK instantiations in our scheme, not solely relying on a specific paradigm, as well as its security assumption. Second, by achieving parallelism, our CF scheme incurs very low proving and verification time asymptotically, with only $\mathcal{O}(|W| \log N)$ time complexity, where N is the number of instance-witness pairs and $|W|$ is the size of each witness.

Our CF Scheme further allows Parallelizable and Verifiable RAM programs (that we dub pvRAM, a.k.a. IVC with associated memory). Specifically, a pvRAM is a proof/argument for RAM programs that enables parallelism natively. As it is a proof, we hence call it verifiable. On one hand, parallelizability is an important property in the context of scaling blockchain as L2 operator is commonly a set of powerful servers and parallelization can significantly bring down the finality time of the block generated by L2. On the other hand, supporting RAM programs (RAM-based machine executions) in a native manner yields efficient verifiable virtual machines such as zkEVMs.[7] As we want our CF scheme to be as flexible in usage as possible we support incomplete binary folding structure (meaning the number of leaves in the folding tree can be less than 2^L where L is the tree depth). Such hierarchy's flexibility is two fold: (1) one do not need 2^L steps of execution to not waste folding effort (no need to pad with dummy steps), and (2) in the case of parallelization, the vCPUs scheduler can easily fit the folding tasks to workers of various different clock cycles; and it also does not require doubling of computing units in case of horizontal scaling of the infrastructure.

RAMenPaSTA and Concrete Instantiations. We observe that one can treat a RAM program as an IVC with an associated memory. We can transform each step of such RAM program into an instance-witness pair that fits the CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ for folding all such instance-witness pairs into a single one that testifies the correctness of the entire computation. Particularly, we employ the

[7] Verifiable execution of the Ethereum Virtual Machine which consists of state storage and memory.

technique for handling memory from [FKL+21] and transform the technique for proving correct execution of RAM programs, with PLONK structure lookup[8] and memory handling techniques from [FKL+21], into the compatible instance-witness pairs forms. Finally, we can fold all such instance-witness pairs, in parallel, into a single pair s.t. its satisfaction to some relation implies a correct computation of all steps of the RAM program.

To this end, we apply the above-mentioned CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ as a building block to construct a generic construction for pvRAM, namely Parallelizable Scalable Transparent Arguments of Knowledge for RAM Programs (RAMenPaSTA). We trade efficiency for *native parallelization* and memory support. Unlike previous non-uniform IVC constructions [KS22, BC23], where the instruction index is set to be public, our construction manages to perfectly hide the index of each instruction in each step by applying the technique for proving the satisfiability of hidden circuit.

Our scheme handles memory consistency check via folding the witnesses and then proves in the end. Hence, the final circuit for proving memory consistency in our scheme only depends on the witness size $|W|$ of a *single* execution trace. This is better compared to previous works [FKL+21, dSGOTV22, YH24], where the size of the circuit for memory consistency check is proportional to the number N of execution steps multiplying the witness size $|W|$. Finally, our scheme incurs only $\mathcal{O}(|W| \log N)$ prover time, dominated by the cost of executing the CF schemes in parallel plus proving the validity of the final instance-witness pair in the last step.

We show in Table 1 the comparison between our work and previous state-of-the-art results in constructing (ZK) proofs/arguments for correct machine executions. Here, our construction is generic and achieves native parallelization. See Sec. 1.2 for detailed related work.

Table 1. Comparison with previous works for proving correct machine execution regarding: no recursion heuristic (no recur. heur.), memory support (mem. supp.), folding (fold.), genericity (generic), no trusted setup (no trust. setup), and hidden program counter or instruction (hidden pc/instr.). (See more in Sec. 1.2.) SuperNova [KS22] employs Merkle tree for committing and proving memory consistency. Such approach has been pointed out to be impractical as in [ZSZG23]. Moreover, RAMenPaSTA supports HVZK for program counter in RAM program, while SuperNova and ProtoStar [BC23] do not. Franseze et al. [FKL+21] and Guilhem et al. [dSGOTV22] provided generic ZK constructions for RAM programs. However, they are not known to be natively parallelizable and cannot defer proving by folding. Moreover, we also do not know whether they hide program counter. To the best of our knowledge, we are the first to support oblivious executions of RAM-based program in the sense that the program counter and its associated instruction are hidden.

Protocol	**No**	**Mem.**	**Fold.**	**Generic**	**No**	**Hidden**
	Recur.	**Supp.**			**Trust.**	**PC/Instr.**
	Heur.				**Setup**	
Sublonk [CGG+23]	✓	-	-	-	-	✓
SuperNova [KS22]	-	✓(Merkle)	✓	✓	✓	-
ProtoStar [BC23]	-	-	✓	✓	✓	-
KiloNova [ZGGX23]	-	-	✓	-	✓	-
MUXProof [DXNT23]	✓	✓	-	-	-	✓
Franseze et al. [FKL+21]	✓	✓	-	✓	-	-
Guilhem et al. [dSGOTV22]	✓	✓	-	✓	✓	-
Ours (Parallel)	✓	✓	✓	✓	✓	✓

1.1 Our Contributions

We summarize our main contributions of this paper as follows:

Conditional Folding Scheme. We propose Contitional Folding (CF) Scheme, a new primitive acting as a building block for constructing pvRAM that avoids recursion heuristic. A CF scheme is an interactive protocol between a prover and verifier. Informally, for two NP relations $\mathcal{R}$ and $\mathcal{R}_{\mathsf{cond}}$, the protocol allows prover to fold two instance-witness pairs into a single pair and to convince verifier in a way that, the folded pair satisfies $\mathcal{R}$ if and only if each of the original pairs also satisfies $\mathcal{R}$, and these two pairs are related, i.e., satisfying $\mathcal{R}_{\mathsf{cond}}$. Concrete syntax and security properties of CF Scheme are presented in Sec. 4.

[8] The notion of Proof of Proving Hidden-Circuit Satisfiability from PLONK is employed by [CGG+23]. Here we only require Folding Plonk Structure.

Generic CF Scheme Construction. We propose a generic construction of CF scheme (Sec. 5) with security proof (Appdx. B). We design a CF scheme where both relations $\mathcal{R}$ and $\mathcal{R}_{\mathsf{cond}}$ are chosen to capture relaxed R1CS (rR1CS) constraints, introduced in [KST22]. Our construction is *both* generic and parallelizable by folding all the instance-witness pair following a binary-tree-like structure.

Security Proof for Generic CF Scheme Construction. We are required to analyze the knowledge soundness property for our CF scheme construction when folding the instance-witness pairs in parallel (recalling that we also support incomplete binary folding structure). The security analysis is non-trivial and becomes one of the main technical contributions in this work.[9]

RAMenPaSTA: Generic, Parallizable and Zero-Knowledge. We propose a generic construction of pvRAM, which employs CF scheme as a sub-protocol, named RAMenPaSTA. Unlike many previous works [BGH19, COS20, BCMS20, BCL^{+}21, KS22, BC23], by leveraging CF scheme, RAMenPaSTA manages to avoid the heuristic recursion issue. Our construction is generic, since we only require CF scheme as the sole building block.

1.2 Related Work

RAMenPaSTA is closely related to the problem of IVC construction and Argument of Correct RAM programs as RAMenPaSTA is IVC with associated memory itself.

Memory Consistency Check. Memory consistency check is one of the required steps in proving the correctness of RAM program, and several solutions for handling memory have been proposed. Braun et al. [BFR^{+}13] proposed a solution to memory using Merkle tree, which has been pointed out to be impractical [WSR^{+}15, ZSZG23]. Another solution is to use the *sorting technique* which is based on routing networks [BCG^{+}13, BCTV14, WSR^{+}15, BBC^{+}17]. While the sorting approach outperforms its Merkle tree approach counterpart [ZSZG23], the sorting-based constructions above still incur at least $\mathcal{O}(N \log N)$ prover cost due to the cost of routing network.

Nevertheless, the sorting technique is a promising approach and thus, there have been several attempts to reduce the cost. Franzese et al. [FKL^{+}21] proposed a construction based on sorting technique for proving memory consistency. Instead of sorting network, they employ polynomial equality check for permutation argument. Their work is further optimized by [dSGOTV22, YH24] with additional properties such as linear prover and verifier costs in public coin setting. All these constructions are generic and offer better efficiency than the previous works above which remove the $\mathcal{O}(\log N)$ overhead, as analyzed in [FKL^{+}21, Sec. 1.1]. However, they still suffer from several efficiency problems. More specifically, in all these constructions, to perform the memory consistency check, they require invoking a circuit whose size is proportional to the number of execution steps. For instance, in [FKL^{+}21, Sec. 3.2], to verify memory consistency, they need to construct an arithmetic circuit of length $\Theta(N)$, which used to check whether two arrays of lengths N are permutation of each other. In addition, the circuit size of [YH24] also depends on the size of the memory, which is not desirable because in several applications memory can be extremely large, e.g., 256-bit addressable memory in EVM [W^{+}14].

Recently, [ZSZG23] proposed another method, called *address cycle method* to reduce the online communication cost between the prover and the verifier in the sorting approach. They also proposed two instantiations for memories with full address space (the space of the address is $\mathbb{F}$) and linear size address space (the address space is $[M]$, where M is the memory size), respectively. However, for memories with full address space (which is our consideration), their protocol inevitably incurs $\mathcal{O}(S \log^2 S)$ prover cost due to multi-point evaluation, where S is the number of distinct addresses in the execution trace. This cost is not desirable because in the worst case, S could be as large as N. In addition, since their protocol do not consider the zero-knowledge property, it is hard to concretely analyze the total complexity of their protocol against those achieving zero-knowledge in term of N.

[9] At the first glance, the extraction for tree-like folding process might seems easy. However, traditional extraction techniques [AC20, AFR23] consider the tree structure as a line (only one path from top to bottom) and then extract with trees of challenges (multi-round special soundess), meaning the instance-witness pairs at some level ℓ only affect the pairs at level $\ell - 1$. Differently we have incomplete binary structure (pairs at level ℓ might affect pairs at some level below $\ell - 1$) therefore we cannot have this consideration. We overcome this obstacle with a new technique that make use of $L + 1$ extractors and an additional extractor that we call $\mathcal{E}_{\mathsf{prf}}$. Such technique could also be of independent research interest.

IVC from Accumulation/Folding Scheme. Since IVC allows us to prove the correctness of incremental computation, they are one of the main components in proving the correct execution of RAM programs, demonstrated by [BCTV14]. Due to these applications and the development of zkVMs, the progress of realizing IVC has rapidly developed. Historically, IVC was constructed using recursive SNARK [BCCT13, BCTV14] where in each step the prover need to employs a SNARK to prove the correctness of the current step and recursively verifies the SNARK produced in the previous step. Such approach has been pointed out to be impractical [CCDW20]. Recently, a new approach called *accumulation/folding* has been employed to construct IVC/PCD with reduced complexity overhead. Starting with [BGH19], this approach has been studied intensively by many subsequent works. At a high level, these constructions employ an accumulation scheme [BCMS20, BCL+21, CCS22, CCG+23, BC23] or folding schemes [KST22, KS22, KS23, LGZX23, ZGGX23] with the idea of accumulation the instance-witness pairs of a NP relation into a single instance-witness pair of the same relation. IVC from accumulation schemes are similar to those from SNARK, except that it recursively verifies the previous proof from the accumulation scheme verifier instead of SNARK. IVC from folding scheme instead fold instances (where each instance also includes a folding verifier) recursively into a single one. Then, checking the validity of final step implies that all the previous steps are valid. Among these constructions, [KS22, BC23, ZGGX23] manage to realize non-uniform IVC, i.e, a generalization of IVC where the number of possible instructions in each step is more than one, and function/instruction in the next computation step is determined from the input of the current step.

IVCs from folding schemes are more efficient than the traditional SNARK-based IVCs since one only need to prove the correctness of the whole execution in the final step with a single folded instance-witness pair, which greatly reduces prover cost. Unfortunately, the majority the accumulating/folding-based constructions above share a common drawback: except [CCS22, CCG+23], all these constructions require modeling the random oracle as an arithmetic circuit, which is problematic since random oracles are ideal functionality and thus cannot be represented by any circuit. In addition, the random oracle (or *low degree random oracle* in the case of [CCS22] and *arithmetic random oracle* in the case of [CCG+23]) has to be instantiated via some cryptographic hash functions, making the realized circuit become expensive due to the number of constraints of hash functions. Moreover, when applying non-uniform IVC to prove the correctness of RAM program, [KS22, BC23, ZGGX23] do not keep program counter (or the structure corresponding to the current instruction in the case of [ZGGX23]) private, for instance, SuperNova [KS22] must at least reveal the program counter of the current computation step.

Other Works. Several other works [IOS23, DXNT23, CGG+23] also proposed different solutions for proving the correctness of machine execution that avoid the IVC approach and thus, manage to avoid the recursion heuristic problem above. However, these constructions have efficiency issue as they both incur heavy cost $\mathcal{O}(N\,|W| \cdot \log(N\,|W|))$ on prover side, where $|W|$ is the witness size. In addition, since the constructions in [DXNT23] and [CGG+23] are built upon Marlin [CHM+20] and PLONK [GWC19] respectively, the security of the protocol must rely on a trusted setup because Marlin and PLONK require trusted setups for public parameters.

1.3 Paper Organization

The paper is organized in the following structure:

- Sec. 1 is the introduction of the paper.
- In Sec. 2, we present the necessary backgrounds including commitment scheme, interactive protocol for folding relaxed R1CS instance-witness pairs and hierarchical structures. Extended preliminaries can be found in Appdx. A.
- In Sec. 3, we describe the technical overview of our result.
- In Sec. 4, we formally define the syntax and security properties of conditional folding schemes (CF schemes).
- In Sec. 5, we propose a generic construction $\mathcal{CF}_{\mathsf{gnr}}$ of CF schemes that supports parallelizable incremental computations with memory, namely, pvRAM. Its security proof can be found in Appdx. B.
- In Sec. 6, we propose a generic pvRAM construction Π_{pvr} employing the CF scheme described in Sec. 5. Its security proof can be found in Appdx. C.
- In Sec. 7, we discuss potential instantiations of Π_{pvr} described in Sec. 6.

2 Preliminaries

Notations. We denote by $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{Z}_+$ to be the sets of integers, natural numbers $\{0, 1, 2, \dots\}$ and positive integers, respectively. For $a, b \in \mathbb{Z}$ s.t. $a \leq b$, we write $[a, b]$ and $[a]$ to indicate $\{a, a+1, \dots, b\}$ and $\{1, \dots, a\}$, respectively. Let $\mathbb{F}$ be a finite field. 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 a interactive proof/argument for prover's knowledge of some witness wit with respect to 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. Moreover, we usually call "pair", "iff","s.t." to indicate "instance-witness pair", "if and only if" and "such that", respectively throughout this paper.

2.1 Commitment Scheme

We recall the syntax of commitment scheme, denoted by $\mathcal{C}$, in Def. 1. $\mathcal{C}$ should satisfy two security properties: binding and hiding. Additionally, we require $\mathcal{C}$ in our construction to be additively homomorphic (see Appdx. A.6).

Definition 1 (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 $b \in \{0, 1\}$.*

Define relation $\mathcal{R}_{\mathsf{com}}$ s.t., for $\hat{\mathsf{c}}$ sampled from R_{ck}, it holds that $(\mathsf{ck}, \tilde{\mathsf{c}};\ \mathbf{c}, \hat{\mathsf{c}}) \in \mathcal{R}_{\mathsf{com}} \iff \tilde{\mathsf{c}} = \mathsf{C.Commit}(\mathsf{ck}, \mathbf{c}, \hat{\mathsf{c}})$.

Remark 1. For ease of writing and developing ideas in this result, for a vector $\mathbf{c}$ over $\mathbb{F}$, we usually write $[\![\mathbf{c}]\!]_{\mathsf{ck}}$ to indicate the tuple $(\mathsf{ck}, \tilde{\mathsf{c}};\ \mathbf{c}, \hat{\mathsf{c}})$ containing the commitment $\tilde{\mathsf{c}}$ and randomness $\hat{\mathsf{c}}$. Here, writing $[\![\mathbf{c}]\!]_{\mathsf{ck}}$ does not imply that $(\mathsf{ck}, \tilde{\mathsf{c}};\ \mathbf{c}, \hat{\mathsf{c}}) \in \mathcal{R}_{\mathsf{com}}$. However, when writing $[\![\mathbf{c}]\!]_{\mathsf{ck}} \in \mathcal{R}_{\mathsf{com}}$, we equivalently understand $(\mathsf{ck}, \tilde{\mathsf{c}};\ \mathbf{c}, \hat{\mathsf{c}}) \in \mathcal{R}_{\mathsf{com}}$. In executing protocol between prover and verifier, it is understood that verifier only knows $\mathsf{ck}, \tilde{\mathsf{c}}$ while prover knows $\mathsf{ck}, \tilde{\mathsf{c}}, \mathbf{c}$ and $\hat{\mathsf{c}}$. Hence, for a vector $\mathbf{c}$ held by prover, we define protocol in (1) for prover to commit to $\mathbf{c}$ and send commitment to verifier so that both parties achieve $[\![\mathbf{c}]\!]_{\mathsf{ck}}$.

$$\mathit{\Pi}_{\mathsf{com}}(\mathsf{ck};\ \mathbf{c}) \to [\![\mathbf{c}]\!]_{\mathsf{ck}} \tag{1}$$

Remark 2. Following Rmk. 1, in some place, 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 verifier knows a, c and commitments to $\mathbf{b}, \mathbf{d}$ while 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 begin of Sec. 2.

Remark 3 (Homomorphism). In this result, the commitment scheme is assumed to be homomorphic. For two commitment tuples $(\mathsf{ck}, \tilde{\mathsf{c}}_i;\ \mathbf{c}_i, \hat{\mathsf{c}}_i)$ $\forall i \in \{0, 1\}$, for any $\alpha \in \mathbb{F}$, we can achieve the commitments to $\mathbf{c}_2 = \alpha \cdot \mathbf{c}_0$ and $\mathbf{c}_3 = \mathbf{c}_0 + \mathbf{c}_1$ by computing the tuples $(\mathsf{ck}, \tilde{\mathsf{c}}_2;\ \mathbf{c}_2, \hat{\mathsf{c}}_2) = (\mathsf{ck}, \alpha \cdot \tilde{\mathsf{c}}_0;\ \mathbf{c}_0 + \mathbf{c}_1, \alpha \cdot \hat{\mathsf{c}}_0)$ and $(\mathsf{ck}, \tilde{\mathsf{c}}_3;\ \mathbf{c}_3, \hat{\mathsf{c}}_3) = (\mathsf{ck}, \tilde{\mathsf{c}}_0 + \tilde{\mathsf{c}}_1;\ \mathbf{c}_0 + \mathbf{c}_1, \hat{\mathsf{c}}_0 + \hat{\mathsf{c}}_1)$, respectively. For shortening purpose, we can write $[\![\mathbf{c}_2]\!] := \alpha \cdot [\![\mathbf{c}_0]\!]$ and $[\![\mathbf{c}_3]\!] := [\![\mathbf{c}_0]\!] + [\![\mathbf{c}_1]\!]$ which can be understood that, in executing protocols with prover and verifier, verifier computes $\hat{\mathsf{c}}_2$ and $\hat{\mathsf{c}}_3$ while prover computes $\tilde{\mathsf{c}}_2, \mathbf{c}_2, \hat{\mathsf{c}}_2, \tilde{\mathsf{c}}_3, \mathbf{c}_3$ and $\hat{\mathsf{c}}_3$.

Remark 4. Sometimes, we need to compute the commitment for *public vectors*, e.g., the zero vector $\mathbf{0}^k \in \mathbb{F}^k$ for some $k \in \mathbb{N}$. Hence, for the rest of the paper, we always commitment these public vectors with randomness 0. For example, in the case of the zero vector $\mathbf{0}$ we have $[\![\mathbf{0}]\!]_{\mathsf{ck}} = (\mathsf{ck}, \mathsf{C.Commit}(\mathsf{ck}, \mathbf{0}, 0);\ \mathbf{0}, 0)$ for any ck. In this case, the commitments to these public vectors can be publicly determined by verifier without having to interact with prover.

2.2 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 Appdx. A.7), as an interactive protocol, from [KST22].

rR1CS. Let $\mathbb{F}$ be a finite field. Let $c, m_1, \ldots, m_c, n \in \mathcal{Z}_+$, $m = 1 + \sum_{i=1}^{c} m_i$. Assume that $\mathbb{x} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, \mathbb{x}.\mathbf{z}_1, \ldots, \mathbb{x}.\mathbf{z}_c, \mathbb{x}.\mathbf{e})$. Let $\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_c, \mathsf{cke})$ be a tuple of commitment keys. Then, we write

$$\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, \llbracket \mathbb{x}.\mathbf{z}_1 \rrbracket_{\mathsf{ck}_1}, \ldots, \llbracket \mathbb{x}.\mathbf{z}_c \rrbracket_{\mathsf{ck}_c}, \llbracket \mathbb{x}.\mathbf{e} \rrbracket_{\mathsf{cke}}) \tag{2}$$

to indicate instance-witness pair after committing to $\mathbb{x}.\mathbf{z}_1, \ldots, \mathbb{x}.\mathbf{z}_c$ and $\mathbb{x}.\mathbf{e}$ by $\mathsf{ck}_1, \ldots, \mathsf{ck}_c$ and cke, respectively. See Rmk. 2 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, $\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}$ is a valid rR1CS pair if it satisfies the relation

$$\mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}} = \left\{ \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}} \;\middle|\; \begin{array}{l} \mathbb{x}.u \in \mathbb{F} \wedge \mathbb{x}.\mathsf{pub} \in \mathbb{F} \wedge (\mathbb{x}.\mathbf{z}_i \in \mathbb{F}^{m_i} \ \forall i \in [c]) \wedge \mathbb{x}.\mathbf{e} \in \mathbb{F}^n \\ \wedge \llbracket \mathbb{x}.\mathbf{z}_i \rrbracket_{\mathsf{ck}_i} \in \mathcal{R}_{\mathsf{com}} \ \forall i \in [c] \wedge \llbracket \mathbb{x}.\mathbf{e} \rrbracket_{\mathsf{cke}} \in \mathcal{R}_{\mathsf{com}} \\ \wedge \mathbf{A} \cdot \mathbf{z}' \circ \mathbf{B} \cdot \mathbf{z}' = \mathbb{x}.u \cdot \mathbf{C} \cdot \mathbf{z}' + \mathbb{x}.\mathbf{e} \end{array} \right\} \tag{3}$$

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

Folding Two rR1CS Instance-Witness Pairs. We recall the technique in [KST22] for folding two pairs into a single satisfying pair with respect to $\mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ in (3). Let $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}$, $\llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}$ be two pairs satisfying $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}, \llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$. We show how to fold $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}$ and $\llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}$ to obtain the new pair $\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$. With random α, the idea is to compute $\mathbb{x}.u = \mathbb{x}_0.u + \alpha \cdot \mathbb{x}_1.u$, $\mathbb{x}.\mathsf{pub} := \mathbb{x}_0.\mathsf{pub} + \alpha \cdot \mathbb{x}_1.\mathsf{pub}$, $\mathbb{x}.\mathbf{z}_i = \mathbb{x}_0.\mathbf{z}_i + \alpha \cdot \mathbb{x}_1.\mathbf{z}_i \ \forall i \in [c]$ and $\mathbb{x}.\mathbf{e} := \mathbb{x}_0.\mathbf{e} + \alpha \cdot \mathbb{x}_1.\mathbf{e}$. Let $\mathbf{z}'_i := (\mathbb{x}_i.\mathsf{pub} \| \mathbb{x}.\mathbf{z}_1 \| \ldots \| \mathbb{x}_i.\mathbf{z}_c) \ \forall i \in \{0,1\}$ and $\mathbf{z}' := (\mathbb{x}.\mathsf{pub} \| \mathbb{x}.\mathbf{z}_1 \| \ldots \| \mathbb{x}.\mathbf{z}_c)$. Define

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

Then, one can check that

$$\mathbf{A} \cdot \mathbf{z}' \circ \mathbf{B} \cdot \mathbf{z}' = \mathbb{x}.u \cdot \mathbf{C} \cdot \mathbf{z}' + \mathbb{x}.\mathbf{e} \tag{4}$$

where $\mathbb{x}.\mathbf{e} = \mathbb{x}_0.\mathbf{e} + \alpha \cdot \mathsf{garb}(\mathfrak{S}, \llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}, \llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}) + \alpha^2 \cdot \mathbb{x}_1.\mathbf{e}$. From (4), we obtain a new pair $\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}$ satisfying $\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$. Hence, we present protocol Π_{rr1cs} (in Cstr. 1) for folding two pairs with extractability discussed in Lm. 1.

Construction 1 (Protocol Π_{rr1cs}.). We first parse

$$\begin{aligned} \llbracket \mathbb{x}_i \rrbracket_{\mathsf{tck}} &= (\mathbb{x}_i.u,\ \mathbb{x}_i.\mathsf{pub},\ \llbracket \mathbb{x}_i.\mathbf{z}_1 \rrbracket_{\mathsf{ck}_1},\ \ldots,\ \llbracket \mathbb{x}_i.\mathbf{z}_c \rrbracket_{\mathsf{ck}_c},\ \llbracket \mathbb{x}_i.\mathbf{e} \rrbracket_{\mathsf{cke}}) \ \forall i \in \{0,1\}, \\ \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}} &= (\mathbb{x}.u,\ \mathbb{x}.\mathsf{pub},\ \llbracket \mathbb{x}.\mathbf{z}_1 \rrbracket_{\mathsf{ck}_1},\ \ldots,\ \llbracket \mathbb{x}.\mathbf{z}_c \rrbracket_{\mathsf{ck}_c},\ \llbracket \mathbb{x}.\mathbf{e} \rrbracket_{\mathsf{cke}}\ ) \end{aligned}$$

to follow the form (2). We recall protocol Π_{rr1cs} for folding two satisfying pairs of rR1CS into a new satisfying pair. Let $\mathcal{C}$ be a homomorphic commitment scheme (Sec. 2.1). Protocol Π_{rr1cs} works as follows.

$\underline{\Pi_{\mathsf{rr1cs}}(\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_c, \mathsf{cke}), \mathfrak{S}, \llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}, \llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}) \to \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}}$

1. Prover: $\mathbf{g} \leftarrow \mathsf{garb}(\mathfrak{S}, \llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}, \llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}})$.
2. Both parties run $\llbracket \mathbf{g} \rrbracket_{\mathsf{cke}} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke};\ \mathbf{g})$ where Π_{com} is defined in (1).
3. Verifier: $\alpha \xleftarrow{\$} \mathbb{F}$ and send α to prover.
4. This step folds $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}$ and $\llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}$ with α. Specifically, both parties compute:

$$\begin{aligned} &\mathbb{x}.u := \mathbb{x}_0.u + \alpha \cdot \mathbb{x}_1.u, && \llbracket \mathbb{x}.\mathbf{z}_i \rrbracket_{\mathsf{ck}_i} := \llbracket \mathbb{x}_0.\mathbf{z}_i \rrbracket_{\mathsf{ck}_i} + \alpha \cdot \llbracket \mathbb{x}_1.\mathbf{z}_i \rrbracket_{\mathsf{ck}_i} \ \forall i \in [c], \\ &\mathbb{x}.\mathsf{pub} := \mathbb{x}_0.\mathsf{pub} + \alpha \cdot \mathbb{x}_1.\mathsf{pub}, && \llbracket \mathbb{x}.\mathbf{e} \rrbracket_{\mathsf{cke}} := \llbracket \mathbb{x}_0.\mathbf{e} \rrbracket_{\mathsf{cke}} + \alpha \cdot \llbracket \mathbf{g} \rrbracket_{\mathsf{cke}} + \alpha^2 \cdot \llbracket \mathbb{x}_1.\mathbf{e} \rrbracket_{\mathsf{cke}}. \end{aligned}$$

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

The proof of Lemma 1 will be presented in Appdx. A.9.

2.3 Hierarchical Structures

As our result focuses on folding an N-step computation in parallel which can be done by following a tree-like structure, in this section, we define the family of hierarchical structures that helps formalize the parallel folding process above. Intuitively, suppose we have two tree-like structures, representing the folding process from steps $l+1$ to j and from steps $j+1$ to r. Then we can combine these two to obtain a tree-like structure that represents the folding process from step $l+1$ to r. We formally define the family hierarchical structures in Def. 2.

Definition 2 (Hierarchical Structures). *For any $l, r \in \mathbb{Z}_+$, $l < r$, a family $\mathcal{HS}_{lr}$ is a set of hierarchical structures s.t. each $\mathsf{HS} \in \mathcal{HS}_{lr}$ satisfies:*

- *If $l+1 = r$, then HS contains only (l, r).*
- *If $l+1 < r$, then there exists uniquely $j \in [l+1, r-1]$ s.t. there exist $\mathsf{HS}_0 \in \mathcal{HS}_{lj}$, $\mathsf{HS}_1 \in \mathcal{HS}_{jr}$ satisfying $\mathsf{HS} = \mathsf{HS}_0 \cup \mathsf{HS}_1 \cup \{(l, r)\}$.*

3 Technical Overview

We discuss technical overview of our result. This is split intro three parts including a definition of CF scheme, generic constructions of CF scheme and pvRAM, and security proof of these constructions in Sec.3.1, 3.2 and 3.3, respectively.

3.1 Defining CF Schemes

The Challenges. Let $\mathcal{R}$ be some NP relation. We would like to define the CF scheme that fold two instance-witness pairs $(Z_0;\ W_0), (Z_1;\ W_1) \in \mathcal{R}$ into $(Z;\ W) \in \mathcal{R}$ s.t. the folding process is successful iff certain conditions between $(Z_0;\ W_0)$ and $(Z_1;\ W_1)$ are met. This condition is captured by relation $\mathcal{R}_{\mathsf{cond}}$ informally defined as follows. The pairs $(Z_0;\ W_0)$ and $(Z_1;\ W_1)$ together satisfy this condition if there exists an auxiliary witness W' satisfying $(Z_0, Z_1;\ W_0, W_1, W') \in \mathcal{R}_{\mathsf{cond}}$. Regarding auxiliary witness W', for example, when comparing $a, b \in \mathbb{Z}$, we may need to decompose a and b into binary. Those binary forms are considered to be included in W'. In other words, a CF scheme is defined s.t. the membership of folded pair holds, i.e., $(Z;\ W) \in \mathcal{R}$, iff the memberships of $(Z_0;\ W_0)$ and $(Z_1;\ W_1)$ in $\mathcal{R}$ hold and there exists some auxiliary witness W' satisfying $(Z_0, Z_1;\ W_0, W_1, W') \in \mathcal{R}_{\mathsf{cond}}$.

Previous works [BCMS20, BCL$^+$21, BC23, KST22, KS22, KS23, LXZ$^+$24, ZGGX23], especially Nova [KST22], employ cycle of curves and recursion heuristic. The usage of cycle of curve is due to the fact that Nova's IVC witness contains both group (elements over a group $\mathbb{G}$) and scalar elements (elements over a field $\mathbb{F}$ s.t. $|\mathbb{F}| = |\mathbb{G}|$), whose base fields are different. To ensure all the components of the witness are represented and computed in the same field, Nova must perform the folding steps in a cycle of curve. Since Nova needs its instance to be short and fixed, it has to realize verifier's work, having random oracle call realized by a hash function, as a circuit. This method is called recursion heuristic, which incurs a violation to random oracle model (ROM) since random oracle (RO) is an idealized object and cannot be realized by any circuit with the following drawbacks. (i) Recursion heuristic is not a standard method for making a proof although many constructions employed this strategy. (ii) As RO is heuristically instantiated by a hash (e.g, SHA-256, Poseidon [GKR$^+$21]), realizing a hash function as a circuit is actually expensive, in terms of efficiency, for prover. (iii) Since they apply recursion heuristic to prove IVC sequentially, they suffer the issue of linear-depth recursion.

We now consider CF scheme. Our proposed CF scheme allows us to avoid those mentioned issues. Moreover, it also enables a generic construction of RAM program, as a non-uniform IVC with an associated memory, that is natively parallelizable and can be instantiated from different paradigms and assumptions. These positivities will be clarified later in this section.

We now consider specifying syntax and defining security of a CF scheme. This is a challenging task. In fact, as discussed above, verifier only knows Z_0, Z_1 and Z, impossible for verifier to check whether there exists W s.t. $(Z;\ W) \in \mathcal{R}$. Therefore, to ensure this, prover and verifier need to conduct a proof/argument for $(Z;\ W) \in \mathcal{R}$ since prover knows W. However, as we apply this CF scheme for folding not only two, but N instance-witness pairs, we hence cannot proceed a proof/argument after each folding since this way incurs a huge overhead of communication. Inspired by Nova, we instead fold all pairs into a single pair for proceeding the proof/argument checking the membership in $\mathcal{R}$. In Nova [KST22], they sequentially fold an N-step computation and informally said that the process can be adapted to fold in parallel. Paranova [ZV23] also attempted modifying Nova to make it parallel. However, detail specification and security proof of Nova and Paranova for parallel folding are not carefully considered. Moreover, it is not known whether their approaches are generic.

In our consideration, the folding process can be done by folding all those instance-witness pairs following a (not necessarily complete) binary tree, called *hierarchical structure* HS (see Sec. 2.3). In this way, we enable the parallelism of this folding process which results in a reduced time of execution. To fold N instance-witness pairs into a single pair, we assume that each of these N pairs corresponds to a leaf of HS. By following a topological order of HS, we can obtain a folded pair at the root of HS. This pair is hence considered the folded pair of those N pairs at the leaf nodes. Finally, prover and verifier proceed an (interactive) argument for the validity of the final pair, namely, belonging to $\mathcal{R}$.

Definition of CF Schemes - Syntax and Security. With the above folding strategy, we can define the syntax of a CF scheme $\mathcal{CF}$, for two parties, namely, prover and verifier, to contain an algorithm $\mathsf{CF.Setup}$, for returning public parameters, and two protocols, namely, $\mathsf{CF.Fold}$ for folding two instance-witness pairs into a new pair and $\mathsf{CF.Prove}$ for proving the satisfaction, in $\mathcal{R}$, of a pair.

We discuss correctness and security of $\mathcal{CF}$. Correctness ensures that, if both parties honestly follow protocols $\mathsf{CF.Fold}$ and $\mathsf{CF.Prove}$, $\mathsf{CF.Prove}$ always returns 1 indicating the satisfaction of the respective instance-witness pair. Security of $\mathcal{CF}$ includes soundness and honest-verifier zero-knowledge (HVZK). Regarding soundness, for an N-step computation represented by instance-witness pairs $\{(Z_i;\ W_i)\}_{i\in[N]}$ folded into a single pair $(Z;\ W)$, if there exists some $i \in [N]$ s.t. $(Z_i;\ W_i) \notin \mathcal{R}$ or, in case that $i \in [2, N]$, condition between $(Z_{i-1};\ W_{i-1})$ and $(Z_i;\ W_i)$ is not satisfied, i.e., no W' s.t. $(Z_{i-1}, Z_i;\ W_{i-1}, W_i, W') \in \mathcal{R}_{\mathsf{cond}}$, then prover has negligible probability to convince verifier when executing $\mathsf{CF.Prove}$. In our definition of CF schemes in Sec. 4, we define a stronger version of soundness, namely, knowledge soundness, in a way that there exists some extractor to extract back the witnesses $(W_i)_{i\in[N]}$ and auxiliary witnesses, supporting the conditions between those pairs, satisfying all required constraints w.r.t. $\mathcal{R}$ and $\mathcal{R}_{\mathsf{cond}}$. Regarding HVZK, by following traditional definitions from ZK proofs/arguments, we capture the fact that no information about witnesses is compromised beyond the validity of the prover's statement.

3.2 Generic Constructions of CF Scheme and pvRAM

Recall that a pvRAM is a parallelizable proof/argument for RAM programs. We use the model of RAM program from [FKL+21] that contains

- an instruction set $\mathcal{F}$ containing T instructions denoted by $\mathcal{F} = \{F'_j\}_{j\in[T]}$,
- a register containing c_{reg} addresses for some small constant c_{reg},
- a memory $\mathsf{mem} = (\mathsf{mem}_i)_{i\in[M]}$ of M addresses (cells),
- a program counter for determining the next instruction, among $\mathcal{F}$.

When executing an N-step RAM program, for the step i, we need inputs including a program counter pc_{i-1}, register reg_{i-1} and an input value val_{i-1}, where subscript "$i-1$" is to indicate the current state of the program counter and register before executing step i (or after step $i-1$). Initially we assume that $\mathsf{pc}_0 = 1$, reg_0 is set to be some initial input to RAM program, and val_0 is some initial dummy value. The program counter pc_{i-1} determines instruction $F_i := F'_{\mathsf{pc}_{i-1}}$ from $\mathcal{F}$, which executes on input $(\mathsf{reg}_{i-1}, \mathsf{val}_{i-1})$ and returns output $(\mathsf{pc}_i, \mathsf{reg}_i, \ell_i, \mathsf{val}_i, \mathsf{mop}_i)$ where $\ell_i \in [M]$ is an address in memory and mop_i is a memory access operation which is either READ or WRITE. Here, $\mathsf{pc}_i, \mathsf{reg}_i$ and val_i are used for the next step while ℓ_i, val_i and mop_i determine the action for modifying the memory. If $\mathsf{mop}_i = \text{WRITE}$, set $\mathsf{mem}_{\ell_i} := \mathsf{val}_i$. Otherwise, set $\mathsf{val}_i = \mathsf{mem}_{\ell_i}$. The output of the computation is written to register at the end, namely, reg_N. See Appdx. A.1 for a detailed description of RAM programs.

By viewing RAM program as an execution involving N sub-executions, where the i-th sub-execution receives $(\mathsf{pc}_{i-1}, \mathsf{reg}_{i-1})$ and returns $(\mathsf{pc}_i, \mathsf{reg}_i, \ell_i, \mathsf{val}_i, \mathsf{mop}_i)$, we can construct a (ZK) pvRAM for proving the execution by realizing CF scheme discussed above. Specifically, we construct an instance-witness pair for each step and then fold all these pairs into a *single* pair by following a hierarchical structure. Finally, we proceed a ZK proof/argument, namely, $\mathsf{CF.Prove}$, for the validity of the final pair. Knowledge soundness and ZK of $\mathsf{CF.Prove}$ implies the validity of the entire execution according to security of CF scheme.

We now discuss our brief design of instance-witness pairs to capture the N steps of RAM program. We then show obstacles and how we deal with them.

Obstacles for Designing Instance-Witness Pairs. An instance to a step of computation is a tuple containing public inputs and commitments to the secrets (components need to be protected by ZK property) while a witness contains secrets and randomness of the commitments. However, we face some obstacles:

1. Since we would like to enable ZK of the computation, $(\mathsf{pc}_i)_{i\in[0,N]}$ should not be compromised. Therefore, $F_i = F'_{\mathsf{pc}_{i-1}}\ \forall i \in [N]$ must be private, requiring us to deal with the proof of hidden function executions. Moreover, we need to ensure that the selection of $F_i = F'_{\mathsf{pc}_{i-1}}$ from $\{F'_j\}_{j\in[T]}$ is correct.
2. Memory and register are two types of storage in RAM program. Since register is of constant size, it is viewed as a direct input to each instruction. However, memory is extremely huge. We can apply the memory check technique in [FKL+21]. Adapting this technique into the scope of CF scheme is not a direct approach.
3. The connection between consecutive instance-witness pairs must be ensured, e.g., memory accesses and outputs after step i and inputs before step $i+1$ must be consistent. This connection can be captured by relation $\mathcal{R}_{\mathsf{cond}}$ of our notion of CF scheme. However, realizing this is a non-trivial task.

We now elaborate our solution to the above obstacles.

Universally Realizing Instructions by PLONK Structures. There are two approaches universally realizing instructions, in the forms of arithmetic circuits, by using either R1CS or PLONK's arithmetization. However, R1CS matrices usually take quadratic size on the number of circuit gates. There are some optimizations [Set20, CHM+20, DXNT23] to reduce the size, based on *sumcheck arguments* [LFKN90, BCR+19], to be linear in the number of gates. However, these optimizations seem to be not applicable to our approach since we would like a generic approach to reach other assumptions and proof techniques, e.g., MPC-in-the-head, dual-mode NIWIs, not relying solely on sumcheck. On the other hand, PLONK's arithmetization [GWC19] has two main components, i.e., (i) selectors and (ii) a permutation, for capturing correct gate computations and correct wire connections. A formal description of PLONK's arithmetization is discussed in Appdx. A.3. To universally ensure the correct computation of a function F' in the instruction set $\mathcal{F}$, we realize by using a public circuit to check that corresponding selectors of F' and the secrets are consistent. This public circuit requires two additional inputs, namely, γ and δ, for guaranteeing copy constraints (connections among wires). Then, we use a grand product argument, with γ, δ, to universally ensure that the permutation argument is satisfied (see (38)). Hence, we encode each instruction F'_j in $\mathcal{F}$ to have a PLONK structure plkst'_j in the form of a fixed-length vector over $\mathbb{F}$ containing corresponding selectors and permutation.

Correctly Selecting Instruction with Program Counter. As we need to choose $F_i = F'_{\mathsf{pc}_{i-1}}$, with PLONK structure denoted by plkst_i (for F_i), according to pc_i, for each $i \in [N]$, we need to show that $\mathsf{plkst}_i = \mathsf{plkst}_{\mathsf{pc}_{i-1}}$ is correctly picked from $\{\mathsf{plkst}'_j\}_{j\in[T]}$. This can be captured by showing that

$$\{(\mathsf{pc}_{i-1}\|\mathsf{plkst}_i)\} \subseteq \{(j\|\mathsf{plkst}'_j)\}_{j\in[T]}. \tag{5}$$

We can encode each vector in $\{(\mathsf{pc}_{i-1}\|\mathsf{plkst}_i)\}_{i\in[N]}$ and $\{(j\|\mathsf{plkst}'_j)\}_{j\in[T]}$ by hashing each $(\mathsf{pc}_{i-1}\|\mathsf{plkst}_i)$ and $(j\|\mathsf{plkst}'_j)$ into $\mathsf{plkcp}_i \in \mathbb{F}$ and $\mathsf{plkcp}'_j \in \mathbb{F}$, respectively. Then, we apply lemma for lookup problem from [Hab22] (recalled in Appdx. A.4) to show that $\{\mathsf{plkcp}_i\}_{i\in[N]} \subseteq \{\mathsf{plkcp}'_j\}_{j\in[T]}$ implying (5) with overwhelming probability. In brief, to show that $\{\mathsf{plkcp}_i\}_{i\in[N]} \subseteq \{\mathsf{plkcp}'_j\}_{j\in[T]}$, we can compute $\{\mathsf{plkiv}_i\}_{i\in[N]}, \{\mathsf{plkiv}'_j\}_{j\in[T]}$ from $\{\mathsf{plkcp}_i\}_{i\in[N]}$ and $\{\mathsf{plkcp}'_j\}_{j\in[T]}$ and alternatively show that $\sum_{i\in[N]} \mathsf{plkiv}_i = \sum_{j\in[T]} \mathsf{mul}_j \cdot \mathsf{plkiv}'_j$ where $\{\mathsf{mul}_j\}_{j\in[T]}$ are additional inputs to support lookup argument. Transforming into checking this grand sum, i.e., $\sum_{i\in[N]} \mathsf{plkiv}_i = \sum_{j\in[T]} \mathsf{mul}_j \cdot \mathsf{plkiv}'_j$, is highly compatible with our proposed CF schemes since we can delegate additions to folded pairs.

Attaching Memory Accesses to Each Instance-Witness Pair. Random-access memory can be extremely large, e.g., 256-bit addressable memory in EVM [W$^+$14]. We hence cannot prove by viewing entire memory as input to an instruction in each step. However, each instruction in machines (or architectures), e.g., EVM [W$^+$14] or RISC-V [WLP$^+$14], may access to a limited number of addresses in the memory. There exist solutions for memory handling including Verkle trees [Kus19] and memory checking techniques from [FKL$^+$21]. However, Verkle trees seem do not suitable with our folding scheme. In this result, we follow the memory checking technique from [FKL$^+$21] (recalled in Appdx. A.2). Technically, in [FKL$^+$21], the i-th access to the memory, can be captured by a constant-length vector, denoted by $\mathsf{macs}_i = (\ell_i, \mathsf{time}_i, \mathsf{val}_i, \mathsf{mop}_i) \in \mathbb{F}^4$ in this paper. To show $(\mathsf{macs}_i)_{i\in[N]}$ is formed correctly, they show the existence of $(\mathsf{macs}'_i)_{i\in[N]} = (\ell'_i, \mathsf{time}'_i, \mathsf{val}'_i, \mathsf{mop}'_i)$ s.t.

(i) $1 \leq \ell_i \leq M \wedge \mathsf{mop}_i \in \{0,1\}\ \forall i \in [N]$;
(ii) $(\mathsf{macs}'_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}_i)_{i\in[N]}$; and
(iii) $\forall i \in [2,N]$: $\mathsf{time}_{i-1} < \mathsf{time}_i$, $(\ell'_{i-1} < \ell'_i) \vee ((\ell'_{i-1} = \ell'_i) \wedge (\mathsf{time}'_{i-1} < \mathsf{time}'_i))$, $(\mathsf{macs}_{i-1} = 0) \vee (i-1 > 1)$, $(\ell'_{i-1} = \ell'_i) \vee (\mathsf{mop}'_i = 0)$ and $(\ell'_{i-1} \neq \ell'_i) \vee (\mathsf{val}'_{i-1} = \mathsf{val}'_i) \vee (\mathsf{mop}'_i = 0)$.

Then, they show that there exists constraints between consecutive memory accesses, namely, $(\mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1})$ and $(\mathsf{macs}_i, \mathsf{macs}'_i)$, as in step (iii) above, hold for all $i \in [N]$. Proving permutations can be done by applying [Hab22] (see Appdx. A.4). Specifically, we hash $\mathsf{macs}_i, \mathsf{macs}'_i \in \mathbb{F}^4$ into $\mathsf{mcp}_i, \mathsf{mcp}'_i \in \mathbb{F}$, respectively, and apply [Hab22] to compute their respective inverses $\mathsf{miv}_i, \mathsf{miv}'_i$. Then, we can check permutation by alternatively check that $\sum_{i\in[N]} \mathsf{macs}_i = \sum_{i\in[N]} \mathsf{macs}'_i$, which is highly compatible with our proposed CF schemes since we can delegate additions to folded pairs.
Realizing Conditions by Accumulating. From above discussions, we now see that a computation step i involves to the components

$$\overline{\mathsf{pc}}_i, \overline{\mathsf{reg}}_i, \overline{\mathsf{macs}}_i, \mathsf{plkst}_i, \mathsf{pc}_i, \mathsf{reg}_i, \mathsf{macs}_i, \mathsf{macs}'_i, \mathsf{plkiv}_i, \mathsf{miv}_i, \mathsf{miv}'_i. \tag{6}$$

where $\overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}, \overline{\mathsf{reg}}_i = \mathsf{reg}_{i-1}, \overline{\mathsf{macs}}_i = \mathsf{macs}_{i-1}$. Assume that we can pack these components into a tuple $\mathbb{z}_{(i-1)i}$, viewed as witness of a step. Here, the index "$(i-1)i$" is to indicate a computation receiving as inputs the output from step $i-1$ and returning outputs after step i. In general we write $\mathbb{z}_{ij}$, for $i < j$, to represent the witness for a computation from step $i+1$ to the end of step j.

To make $\mathbb{z}_{ij}$ into an instance-witness pair, we can commit to those components in $\mathbb{z}_{ij}$ by using a homomorphic commitment scheme $\mathcal{C}$. By exploiting homomorphism property as in Rmk. 3, for the same commitment key, any actions on components of $\mathbb{z}_{ij}$ can be done homomorphically in their corresponding commitments and randomness. Therefore, in this Sec. 3.2, we simply discuss manipulation on $\mathbb{z}_{ij}$. Manipulations in commitments and randomness will be understood implicitly and will be presented in detail in Sec. 5 and 6.

Since we consider an N-step computation, we require that the output of the i-th step is equal to the input of $i+1$-th step, which can be captured by enforcing $(\mathsf{pc}_i, \mathsf{reg}_i, \mathsf{macs}_i) = (\overline{\mathsf{pc}}_{i+1}, \overline{\mathsf{reg}}_{i+1}, \overline{\mathsf{macs}}_{i+1})$, for each $i \in [N-1]$. Hence, when applying folding technique from Sec. 2.2 to fold $\mathbb{z}_{(i-1)i}$ and $\mathbb{z}_{i(i+1)}$ into the pair $\mathbb{z}_{(i-1)(i+1)}$, we are unable to capture this connection. Moreover, if we continue to fold, e.g., $\mathbb{z}_{(i-1)(i+1)}$ and $\mathbb{z}_{(i+1)(i+2)}$ into $\mathbb{z}_{(i-1)(i+2)}$, the components needed in $\mathbb{z}_{(i-1)(i+1)}$ are not preserved to ensure the connection with $\mathbb{z}_{(i+1)(i+2)}$.

Additionally, we need to guarantee $\sum_{i\in[N]} \mathsf{plkiv}_i = \sum_{j\in[T]} \mathsf{mul}_j \cdot \mathsf{plkiv}'_j$ and $\sum_{i\in[N]} \mathsf{miv}_i = \sum_{i\in[N]} \mathsf{miv}'_i$ for plonk structure lookup and memory accesses.

Hence, from the above issues, we need a strategy to fold while simultaneously ensuring conditions of folding hold. To this end, we do as follows:

Capturing Conditions. Define R1CS structure $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$ that captures the condition when folding. Specifically, for $i \in [N-1]$, by defining $\mathbf{w}'_i$ to contain necessary components involving conditions for folding pairs i and $i+1$, we can guarantee that the condition hold iff $\mathbf{A}' \cdot \mathbf{w}'_i \circ \mathbf{B}' \cdot \mathbf{w}'_i = \mathbf{C}' \cdot \mathbf{w}'_i$.

Designing Witnesses. Design $\mathbb{z}_{(i-1)i}$, for $i \in [N]$, to contain

$$\mathbb{x}_{(i-1)i}, \mathbf{i}_{(i-1)i}, \mathbf{o}_{(i-1)i}, \mathbb{x}^{\star}_{(i-1)i}, \mathbf{s}_{(i-1)i}, \mathbf{a}_{(i-1)i} \tag{7}$$

as follows:

– $\mathbb{x}_{(i-1)i}$ contains those above mentioned components in (6).

- $\mathbf{i}_{(i-1)i}$ contains $\overline{\mathsf{pc}}_i, \overline{\mathsf{reg}}_i, \overline{\mathsf{macs}}_i$, macs'_i as inputs to step i and $\mathbf{o}_{(i-1)i}$ contains pc_i, reg_i, macs_i, macs'_i as outputs of step i. Here, both $\mathbf{i}_{(i-1)i}$ and $\mathbf{o}_{(i-1)i}$ contain macs'_i because of conditions between macs'_{i-1} and macs'_i (resp., macs'_i and macs'_{i+1}) via $\mathbf{o}_{(i-2)(i-1)}$ and $\mathbf{i}_{(i-1)i}$ (resp., $\mathbf{o}_{(i-1)i}$ and $\mathbf{i}_{i(i+1)}$).
- $\mathbf{s}_{(i-1)i}$ and $\mathbf{a}_{(i-1)i}$ serve as additional components for checking the grand sums $\sum_{i\in[N]} \mathsf{miv}_i = \sum_{i\in[N]} \mathsf{miv}'_i$ and $\sum_{i\in[N]} \mathsf{plkiv}_i = \sum_{j\in[T]} \mathsf{mul}_j \cdot \mathsf{plkiv}'_j$. In particular, for each $i \in [N]$, we assume that $\mathbf{s}_{(i-1)i}$ contains miv_i. Then, later in our design, we will fold $\mathbf{s}_{ij}$ and $\mathbf{s}_{jk}$ into $\mathbf{s}_{ik}$ by computing $\mathbf{s}_{ik} := \mathbf{s}_{ij} + \mathbf{s}_{jk}$. Hence, one sees $\mathbf{s}_{0N}$ contains $\sum_{i\in[N]} \mathsf{miv}_i$. However, to ensure extraction, we additionally design later that $\mathbf{a}_{ik} := \mathbf{a}_{ij} + \alpha_2 \cdot \mathbf{a}_{jk} + \alpha_2^2 \cdot (\mathbf{s}_{ij} - \mathbf{s}_{jk})$ so that we can extract back $\mathbf{a}_{ij}, \mathbf{a}_{jk}, \mathbf{s}'_{ik} = \mathbf{s}_{ij} - \mathbf{s}_{jk}$ by rewinding with 3 different choices of α_2. With $\mathbf{s}'_{ik}$ and $\mathbf{s}_{ik}$, we can solve equations to compute back $\mathbf{s}_{ij}$ and $\mathbf{s}_{jk}$. Detail of this way of extraction is discussed in Lm. 11 in Appdx. B.1.

Here, we notice that $\mathbb{x}_{(i-1)i}$ contains both $\mathbf{i}_{(i-1)i}$ and $\mathbf{o}_{(i-1)i}$. However, this only holds true with witnesses representing the computation steps. In general, if we fold those N pairs following some hierarchical structure $\mathsf{HS} \in \mathcal{HS}_{0N}$, specified in Sec. 2.3, then the fact that $\mathbb{x}_{ij}$ contains both $\mathbf{i}_{ij}$ and $\mathbf{o}_{ij}$ only holds with pairs corresponding to leaf nodes of HS_{0N}, namely, $i + 1 = j$.

Folding Witnesses. We sketch the strategy for folding two witnesses, e.g., folding $\mathbb{z}_{ij}$ and $\mathbb{z}_{jk}$ into $\mathbb{z}_{ik}$ for $0 \le i < j < k \le N$ as follows.

- Apply Nova folding from Sec. 2.2 to obtain $\mathbb{x}_{ik}$ by folding, with respect to $\mathfrak{S}$, from $\mathbb{x}_{ij}$ and $\mathbb{x}_{jk}$.
- Accumulate the condition by first computing a vector $\mathbb{x}'$ containing both $\mathbf{o}_{ij}$ and $\mathbf{i}_{jk}$ in $\mathbb{z}_{ij}$ and $\mathbb{z}_{jk}$, respectively, and applying Nova folding twice to fold $\mathbb{x}^\star_{ij}$, $\mathbb{x}^\star_{jk}$ and $\mathbb{x}'$ into $\mathbb{x}^\star_{ik}$.
- Set $\mathbf{i}_{ik} := \mathbf{i}_{ij}$, namely, input of $\mathbb{z}_{ij}$, and $\mathbf{o}_{ik} := \mathbf{o}_{jk}$, namely, output of $\mathbb{z}_{jk}$.
- Compute $\mathbf{s}_{ik} := \mathbf{s}_{ij} + \mathbf{s}_{jk}$ and $\mathbf{a}_{ik} := \mathbf{a}_{ij} + \alpha_2 \cdot \mathbf{a}_{jk} + \alpha_2^2 \cdot (\mathbf{s}_{ij} - \mathbf{s}_{jk})$ where α_2 is chosen by verifier.

This folding process then suggests us to split the construction into two parts:

- We present in Sec. 5 a generic CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ without taking care of components of RAM programs, i.e., without those in (6).
- Then, we construct RAMenPaSTA (protocol Π_{pvr}) in Sec. 6 as a generic construction of pvRAM that employs $\mathcal{CF}_{\mathsf{gnr}}$ as a building block. Moreover, instantiations of Π_{pvr} are deferred to Appdx. 7.

3.3 Security Proof of Generic CF Scheme

The security of generic CF scheme discussed above includes correctness, knowledge soundness and zero-knowledge (ZK). Completeness is straightforward. ZK of our generic CF scheme follows ZK of $\mathsf{CF.Prove}$ and hiding property of employed homomorphic commitment scheme.

For knowledge soundness, the extraction is non-trivial because one needs to deal with extraction for all nodes in a hierarchical structure. Technically, we divide the hierarchical structure HS, not necessarily of a complete binary tree, that prover and verifier follow in generic CF scheme into multiple levels (see Def. 21) from 0 (bottom) to L (top). State-of-the-art special-sound extraction techniques [AC20, AFR23] consider this tree structure as a line (only one path from top to bottom) and then extract with trees of challenges. Moreover, since HS is not necessarily complete, extracting witnesses is much harder. In fact, for complete binary-tree structure, when extracting pairs in layer $\ell \in [L]$, these pairs only affect level $\ell - 1$. However, for non-complete binary tree, these pairs may affect multiple levels below layer ℓ. To this end, we construct $L + 1$ extractors $\{\mathcal{E}_\ell\}_{\ell\in[0,L]}$ and an extractor $\mathcal{E}_{\mathsf{prf}}$. Here, $\mathcal{E}_\ell$, for $\ell < L$, uses $\mathcal{E}_{\ell+1}$ as a subroutine to extract witnesses at level ℓ and these witnesses are unable to be extracted by $\mathcal{E}_{\ell+1}$. $\mathcal{E}_L$ extracts witnesses by call $\mathcal{E}_{\mathsf{prf}}$ of protocol $\mathsf{CF.Prove}$ of the CF scheme. This extraction technique is presented in Lm. 12 in Appdx. B.1.

4 Conditional Folding Scheme

In this section, we present the definition of conditional folding (CF) scheme $\mathcal{CF}$ for NP relation $\mathcal{R}$ with respect to condition relation $\mathcal{R}_{\mathsf{cond}}$. Informally speaking, such a CF scheme allows to fold the instance-witness pairs satisfying relation $\mathcal{R}$ into a new instance-witness pair satisfying relation $\mathcal{R}$ if both to-be-folded instance-witness pairs satisfy $\mathcal{R}_{\mathsf{cond}}$. In Sec. 4.1, we define the syntax of a conditional folding scheme. Then, in Sec. 4.2, we define its correctness and security properties including knowledge soundness and (honest-verifier) zero-knowledge. See Sec. 3.1 for an overview of syntax and security definitions of CF schemes.

4.1 Syntax

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

$\mathsf{CF.Setup}(1^\lambda) \to \mathsf{pp}$**:** *Run public parameter generator, on input a security parameter 1^λ, it returns a public parameter pp.*

$\mathsf{CF.Fold}(\mathsf{pp}, Z_0, Z_1;\ W_0, W_1, W') \to (Z;\ W)$**:** *This is an interactive protocol between prover, holding $(Z_0, W_0), (Z_1, W_1) \in \mathcal{Z} \times \mathcal{W}$ and auxiliary $W \in \mathcal{W}_{\mathsf{aux}}$ supporting the condition for folding, and verifier, holding instances $Z_0, Z_1 \in \mathcal{Z}$. In the end, prover and verifier receive the folded pair (Z, W) and folded instance $Z \in \mathcal{Z}$, respectively.*

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

4.2 Security Requirements

Before defining security requirements of CF scheme, we first introduce the notion of transcript of a CF scheme. For any $N \in \mathbb{Z}_+$, assume that $\mathsf{HS} \in \mathcal{HS}_{0N}$ is a hierarchical structure as defined in Def. 2. We denote by

$$b \leftarrow \Pi_{\mathcal{CF}}(\mathsf{HS}, \mathsf{pp}, \{Z_{(i-1)i}\}_{i\in[N]};\ \mathsf{sec}) \tag{8}$$

the protocol between prover and verifier, both holding instances $\{Z_{(i-1)i}\}_{i\in[N]}$. In addition, prover holds secret information sec allowing them to compute witnesses during folding and proving. Both parties follow the structure HS to obtain final instance-witness pair $(Z_{0N};\ W_{0N})$ and prove its membership in $\mathcal{R}$. It then returns a bit b indicating whether verifier accepts. Specifically, $\Pi_{\mathcal{CF}}$ runs as follows:

1. For any $(l, j), (j, r) \in \mathsf{HS}$, if prover and verifier already achieved the pairs $((Z_{lj}, W_{lj}), (Z_{jr}, W_{jr})) \in (\mathcal{Z} \times \mathcal{W})^2$ and $(Z_{lj}, Z_{jr}) \in \mathcal{Z}^2$, respectively, prover can compute $W'_j \in \mathcal{W}_{\mathsf{aux}}$, from sec, satisfying $(\mathsf{pp}, Z_{lj}, Z_{jr},\ W_{lj}, W_{jr}, W'_j) \in \mathcal{R}_{\mathsf{cond}}$ and both parties run
$$(Z_{lr};\ W_{lr}) \leftarrow \mathsf{CF.Fold}(\mathsf{pp}, Z_{lj}, Z_{jr};\ W_{lj}, W_{jr}, W'_j)$$
to obtain Z_{lr} for verifier and (Z_{lr}, W_{lr}) for prover.
2. After obtaining $(Z_{0N};\ W_{0N})$, prover and verifier together execute the protocol
$$b \leftarrow \mathsf{CF.Prove}(\mathsf{pp}, Z_{0N};\ W_{0N}).$$

Defining Transcript. Define the public transcript

$$\mathsf{tr} \leftarrow \mathsf{View}(\Pi_{\mathcal{CF}}(\mathsf{HS}, \mathsf{pp}, (Z_{(i-1)i})_{i\in[N]};\ \mathsf{sec})) \tag{9}$$

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

We now define the security properties including perfect correctness, knowledge soundness and honest-verifier zero-knowledge (HVZK) in Def. 4, 5 and 6, respectively, for a CF scheme $\mathcal{CF}[\mathcal{R}, \mathcal{R}_{\mathsf{cond}}]$, with syntax defined in Def. 3:

```
$\mathsf{pp} \leftarrow \mathsf{CF.Setup}(1^\lambda)$.
$((Z_{(i-1)i})_{i\in[N]}, \mathsf{sec}) \leftarrow \mathcal{A}(\mathsf{pp})$.
$\big((W_{(i-1)i})_{i\in[N]}, (W'_j)_{j\in[N-1]}\big) \leftarrow \mathcal{E}^{\mathcal{A}}\big(\mathsf{pp}, (Z_{(i-1)i})_{i\in[N]}\big)$.
$b_0 := \neg \bigwedge_{i\in[N]} \big((\mathsf{pp}, Z_{(i-1)i};\ W_{(i-1)i}) \in \mathcal{R}\big)$.
$b_1 := \neg \bigwedge_{i\in[N-1]} \big((\mathsf{pp}, Z_{(i-1)i}, Z_{i(i+1)};\ W_{(i-1)i}, W_{i(i+1)}, W'_i) \in \mathcal{R}_{\mathsf{cond}}\big)$.
Return $b_0 \vee b_1$.
```

Fig. 2. Experiment $\mathbf{Exp}^{\textsf{cf-knwlg-sound}}_{\mathcal{A}}(\lambda)$.

Definition 4 (Perfect Correctness of $\mathcal{CF}$). *Let* $\mathsf{pp} \leftarrow \mathsf{CF.Setup}(1^\lambda)$*. For any* $\{(\mathsf{pp}, Z_i;\ W_i)\}_{i\in\{0,1\}} \subseteq \mathcal{R}$ *and any* $W' \in \mathcal{W}_{\mathsf{aux}}$ *s.t.* $(\mathsf{pp}, Z_0, Z_1;\ W_0, W_1, W') \in \mathcal{R}_{\mathsf{cond}}$*, it holds that*

$$\Pr\big[(\mathsf{pp}, Z;\ W) \in \mathcal{R} \big| (Z;\ W) \leftarrow \mathsf{CF.Fold}(\mathsf{pp}, Z_0, Z_1;\ W_0, W_1, W')\big] = 1.$$

Moreover, for any $(\mathsf{pp}, Z;\ W) \in \mathcal{R}$*,* $\Pr\big[b = 1 \big| b \leftarrow \mathsf{CF.Prove}(\mathsf{pp}, Z;\ W)\big] = 1$.

We now briefly discuss the intuition for modeling knowledge soundness property in Def. 5. To model this property, 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 $Z_{(i-1)i}$ for all $i \in [N]$. It should hold that extracted witnesses contain $(W_{(i-1)i})_{i=1}^N$ corresponding to $(Z_{(i-1)i})_{i=1}^N$ and, moreover, capture all conditions between $W_{(i-1)i}$ and $W_{i(i+1)}$ for all $i \in [N-1]$.

Definition 5 (Knowledge Soundness of $\mathcal{CF}$). *$\mathcal{CF}$ is said to satisfy knowledge soundness if for any positive integer N, any (PPT) algorithm $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$, with rewindable oracle access to $\mathcal{A}$, it holds that* $\Pr[\mathbf{Exp}^{\textsf{cf-knwlg-sound}}_{\mathcal{A}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ *where* $\mathbf{Exp}^{\textsf{cf-knwlg-sound}}_{\mathcal{A}}(\lambda)$ *is described in Fig. 2. A CF scheme $\mathcal{CF}$ is statistically (resp., computationally) knowledge-sound if $\mathcal{A}$ is computationally unbounded (resp., bounded). $\mathcal{CF}$ has soundness error ϵ if $\mathcal{A}$ can break knowledge soundness with probability at most ϵ.*

Remark 5. In Fig. 2, we write $\mathcal{E}^{\mathcal{A}}(\mathsf{pp}, (Z_{(i-1)i})_{i\in[N]})$ to indicate that $\mathcal{E}$ (having oracle access to $\mathcal{A}$) and $\mathcal{A}$, playing the roles of verifier and prover, respectively, to run $\Pi_{\mathcal{CF}}$ on common inputs $(\mathsf{pp}, (Z_{(i-1)i})_{i\in[N]})$. Moreover, $\mathcal{E}$ has an ability to rewind $\mathcal{A}$ to any previous state of $\mathcal{A}$ and run other instances of $\Pi_{\mathcal{CF}}$ with different randomness. Finally, if all instances of $\Pi_{\mathcal{CF}}$ return 1 after interacting with $\mathcal{A}$ with rewinding, $\mathcal{E}$ can return the valid witnesses $(W_{(i-1)i})_{i\in[N]}, (W'_j)_{j\in[N-1]}$ s.t. $(Z_i, W_i)_{i\in[N]}$ forms a correct N-step computation with each witness W'_j, for all $j \in [N-1]$ captures the condition between steps j and $j+1$.

Definition 6 (HVZK of $\mathcal{CF}$). *$\mathcal{CF}$ is HVZK if there exists a PPT simulator $\mathcal{S}$ s.t., for any distinguisher $\mathcal{A}$, any valid* $(\mathsf{HS}, \mathsf{pp}, \{Z_{(i-1)i}\}_{i\in[N]}, \mathsf{sec})$ *s.t., with* sec *induce the corresponding witnesses corresponding to* $\{Z_{(i-1)i}\}_{i\in[N]}$ *for a correct N-step computation, it holds that*

$$\begin{aligned}&\big|\Pr\big[\mathcal{A}(\mathsf{tr}) = 1 \big| \mathsf{pp} \leftarrow \mathsf{CF.Setup}(1^\lambda), \mathsf{tr} \leftarrow \mathsf{View}(\Pi_{\mathcal{CF}}(\mathsf{HS}, \mathsf{pp}, (Z_{(i-1)i})_{i\in[N]};\ \mathsf{sec}))\big] \\ &\quad - \Pr\big[\mathcal{A}(\mathsf{tr}) = 1 \big| \mathsf{pp} \leftarrow \mathsf{CF.Setup}(1^\lambda), \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{HS}, \mathsf{pp}, (Z_{(i-1)i})_{i\in[N]})\big]\big| \leq \mathsf{negl}(\lambda)\end{aligned}$$

where sec *is some secret of prover. $\mathcal{CF}$ is statistically (resp., computationally) HVZK if $\mathcal{A}$ is computationally unbounded (resp., bounded).*

5 A Generic CF Scheme

In this section, we propose a generic CF scheme $\mathcal{CF}_{\mathsf{gnr}}$, following technical overview sketched in Sec. 3.2, that supports constructing pvRAM, to be discussed in Sec. 6. In Sec. 5.1, we design the form of instance-witness pairs such that each pair contains sufficient information regarding the trace of one computation step or conditions between consecutive computation steps. In Sec. 5.2, we define the relations for the pairs defined in Sec. 5.1. These include $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ for constraints inside a single pair and $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ for conditions required for folding. Then, in Sec. 5.3, we construct protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ for folding two instance-witness pairs specified in Sec. 5.1 with respect to relations defined in Sec. 5.2.

5.1 Designing Forms of Instances and Witnesses for Folding Computation Steps

Let $m_1, \ldots, m_5, n, m'_1, m'_2, m'_3, n' \in \mathbb{Z}_+$. Let $m = 1 + \sum_{i=1}^{5} m_i$ and $m' = 1 + \sum_{i=1}^{3} m'_i$. For an N-step computation, discussed in Sec. 3.2, let $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C}) \in (\mathbb{F}^{n \times m})^3$ and $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}') \in (\mathbb{F}^{n' \times m'})^3$ s.t.

- $\mathfrak{S}$ contains R1CS matrices for constraining the witness of a satisfying computation step; and
- $\mathfrak{S}'$ has R1CS matrices for capturing conditions between consecutive steps.

Let $\mathcal{C}$ be a homomorphic commitment scheme defined in Sec. 2.1. We define instance-witness pair $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}$ (see Rmk. 2 for use of notation), where pp and $\mathbb{z}$ are of the forms (10) and (11), respectively.

$$\begin{array}{lcllllllllr}
\mathsf{pp} & = & (& & \mathsf{tck}, & & & \mathsf{tck}' & & &), \quad (10) \\
\mathbb{z} & = & (\mathsf{lt}, & \mathsf{rt}, & \mathbb{x}, & \mathbf{i}, & \mathbf{o}, & \mathbb{x}^\star, & \mathbf{s}, & \mathbf{a} &), \quad (11) \\
\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} & = & (\mathsf{lt}, & \mathsf{rt}, & \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}, & \llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, & \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2}, & \llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}, & \llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5}, & \llbracket \mathbf{a} \rrbracket_{\mathsf{ck}_5} &) \quad (12)
\end{array}$$

where $\mathsf{ck}_1, \ldots, \mathsf{ck}_5, \mathsf{cke}, \mathsf{ck}'_1 = \mathsf{ck}_2, \mathsf{ck}'_2 = \mathsf{ck}_1, \mathsf{ck}'_3, \mathsf{cke}'$ are commitment keys for committing messages over $\mathbb{F}$ of lengths $m_1, \ldots, m_5$, n, m'_1, m'_2, m'_3, n', respectively, $\mathsf{lt}, \mathsf{rt} \in \mathbb{N}$ are indices; $\mathsf{tck} = (\mathsf{ck}_1, \ldots, \mathsf{ck}_5, \mathsf{cke})$ and $\mathsf{tck}' = (\mathsf{ck}'_1, \ldots, \mathsf{ck}'_3, \mathsf{cke}')$; $\mathbf{i}$, $\mathbf{o}$, $\mathbf{s}$, $\mathbf{a}$ are vectors over $\mathbb{F}$;

$$\begin{array}{lclllllll}
\mathbb{x} & = & (\mathbb{x}.u, & \mathbb{x}.\mathsf{pub}, & \mathbb{x}.\mathbf{z}_1, & \ldots, & \mathbb{x}.\mathbf{z}_5, & \mathbb{x}.\mathbf{e} &), \\
\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}} & = & (& & \llbracket \mathbb{x}.\mathbf{z}_1 \rrbracket_{\mathsf{ck}_1}, & \ldots, & \llbracket \mathbb{x}.\mathbf{z}_5 \rrbracket_{\mathsf{ck}_5}, & \llbracket \mathbb{x}.\mathbf{e} \rrbracket_{\mathsf{cke}} &), \quad (13) \\
\mathbb{x}^\star & = & (\mathbb{x}^\star.u, & \mathbb{x}^\star.\mathsf{pub}, & \mathbb{x}^\star.\mathbf{z}_1, & \ldots, & \mathbb{x}^\star.\mathbf{z}_3, & \mathbb{x}^\star.\mathbf{e} &), \\
\llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'} & = & (& & \llbracket \mathbb{x}^\star.\mathbf{z}_1 \rrbracket_{\mathsf{ck}'_1}, & \ldots, & \llbracket \mathbb{x}^\star.\mathbf{z}_3 \rrbracket_{\mathsf{ck}'_3}, & \llbracket \mathbb{x}^\star.\mathbf{e} \rrbracket_{\mathsf{cke}'} &). \quad (14)
\end{array}$$

Here, the pairs $\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}$, $\llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}$ satisfy $\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}, \llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ (see (3)).

5.2 Defining Relations

Notice that, since we are constructing CF scheme $\mathcal{CF}_{\mathsf{gnr}}$, for an instance-witness pair $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}$ of the form (12), we now define the relations supporting our construction of CF scheme $\mathcal{CF}_{\mathsf{gnr}}$.

Let pp of the form (10), $\mathfrak{S}, \mathfrak{S}'$ be fixed in advance. Recall the discussion of N-step computation in Sec. 3 and Appdx. A.1. We define the relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$, in (16), and $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$, in (17), respectively, with respect to the following intuition.

- $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ captures the constraints for each pair in a single computation step.
- $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ captures the condition between the two to-be-folded pairs in order to perform the folding from CF scheme $\mathcal{CF}_{\mathsf{gnr}}$.

Before defining $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$, we need to specify relation $\mathcal{R}_{\mathsf{gnr\text{-}com}}$, for public parameter pp of the form (10), to capture the relationship between components in $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}$:

$$\mathcal{R}_{\mathsf{gnr\text{-}com}} = \left\{ \llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} \middle| \llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} \text{ of form (12)} \wedge (\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a} \rrbracket_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}) \right\}. \tag{15}$$

Next, we define relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ to be

$$\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} = \left\{ \llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} \middle| \begin{array}{l} \mathsf{lt}, \mathsf{rt} \in \mathbb{N} \wedge (\mathsf{lt} < \mathsf{rt}) \wedge \llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}com}} \\ \wedge \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}} \wedge \llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}} \end{array} \right\}. \tag{16}$$

We now define the condition relation $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ with associated set $\mathcal{W}_{\mathsf{pvr\text{-}aux}}$ defined to be the set containing all auxiliary witnesses in the forms of fixed-length vectors over $\mathbb{F}$ s.t. we can fold $\llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}$ and $\llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}}$ with auxiliary witness $\mathbf{w} \in \mathcal{W}_{\mathsf{pvr\text{-}aux}}$. Parse

$$\mathbb{z}_i = (\mathsf{lt}_i, \mathsf{rt}_i, \llbracket \mathbb{x}_i \rrbracket_{\mathsf{tck}}, \llbracket \mathbf{i}_i \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o}_i \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbb{x}^\star_i \rrbracket_{\mathsf{tck}'}, \llbracket \mathbf{s}_i \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a}_i \rrbracket_{\mathsf{ck}_5})$$

as in (12) for all $i \in \{0, 1\}$. Relation $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ is defined to be

$$\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}} = \left\{ (\llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}, \llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}};\ \mathbf{w}) \middle| \begin{array}{l} \mathbf{w} \in \mathcal{W}_{\mathsf{pvr\text{-}aux}} \wedge \mathsf{rt}_0 = \mathsf{lt}_1 \\ \wedge \mathbf{A}' \cdot \mathbf{c} \circ \mathbf{B}' \cdot \mathbf{c} = \mathbf{C}' \cdot \mathbf{c} \end{array} \right\} \tag{17}$$

where $\mathbf{c} = (1\|\mathbf{o}_0\|\mathbf{i}_1\|\mathbf{w})$. Relation $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ specifies the condition in the form of a computation of R1CS equation with respect to matrices $\mathbf{A}', \mathbf{B}'$ and $\mathbf{C}'$ in $\mathfrak{S}'$. Specifically, it specifies the conditions between consecutive computation steps,i.e, $\llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}$ and $\llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}}$ by enforcing certain constraints between $\mathbf{o}_0$ and $\mathbf{i}_1$. Since $\llbracket \mathbb{z}_i \rrbracket_{\mathsf{pp}}$, for $i \in \{0, 1\}$, represents the computation from the steps $\mathsf{lt}_i + 1$ to rt_i, by folding $\llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}$ and $\llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}}$, we enforce $\mathsf{rt}_0 = \mathsf{lt}_1$ in order to obtain folded $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}$ that represents consecutive computation steps from $\mathsf{lt}_0 + 1$ to rt_1.

5.3 Protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ and Construction of CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$

We first describe protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ for folding $\llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}$ and $\llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}}$ into $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}$. Then, we describe the CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ employing $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ as a building block.

Overview of $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$. We describe in high level the protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ for folding $\llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}$ and $\llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}}$, where, $\forall i \in \{0, 1\}$, by following (12) to parse

$$\llbracket \mathbb{z}_i \rrbracket_{\mathsf{pp}} = (\mathsf{lt}_i, \mathsf{rt}_i, \llbracket \mathbb{x}_i \rrbracket_{\mathsf{tck}}, \llbracket \mathbf{i}_i \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o}_i \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbb{x}^\star_i \rrbracket_{\mathsf{tck}'}, \llbracket \mathbf{s}_i \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a}_i \rrbracket_{\mathsf{ck}_5}),$$

into $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}$ as follows:

- Assume $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} = (\mathsf{lt}, \mathsf{rt}, \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}, \llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}, \llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a} \rrbracket_{\mathsf{ck}_5})$ as in (12).
- It is assumed that $\mathsf{rt}_0 = \mathsf{lt}_1$ according to $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ in (17). Hence, the public indices lt and rt of folded instance $\mathbb{z}$ are set to be lt_0 and rt_1, respectively, to represent the computation from steps $(\mathsf{lt}_0 + 1)$ to rt_1. Similarly, set input $\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1} := \llbracket \mathbf{i}_0 \rrbracket_{\mathsf{ck}_1}$ and output $\llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2} := \llbracket \mathbf{o}_1 \rrbracket_{\mathsf{ck}_2}$.
- Obtain folded rR1CS instance-witness pair by folding $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}$ and $\llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}$ into $\llbracket \mathbb{x} \rrbracket$ with respect to a challenge $\alpha_1 \in \mathbb{F}$. This can be done by running protocol $\mathit{\Pi}_{\mathsf{rr1cs}}$ defined in Cstr. 1 with respect to tck and $\mathfrak{S}$.
- Obtain the accumulated relaxed R1CS instance-witness pair $\llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}$ for condition by first forming the rR1CS instance-witness pair $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'}$ where $\mathbb{y}$ contains $\mathbf{o}_0, \mathbf{i}_1$ and some auxiliary witness $\mathbf{w} \in \mathcal{W}_{\mathsf{pvr\text{-}aux}}$ s.t. $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}$. Then, fold the three pairs $\llbracket \mathbb{x}^\star_0 \rrbracket_{\mathsf{tck}'}$ $\llbracket \mathbb{x}^\star_1 \rrbracket_{\mathsf{tck}'}$ and $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'}$, into $\llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}$ by first running protocol $\mathit{\Pi}_{\mathsf{rr1cs}}$, described in Cstr. 1, with challenge $\alpha_1 \in \mathbb{F}$ to fold $\llbracket \mathbb{x}^\star_0 \rrbracket_{\mathsf{tck}'}$ and $\llbracket \mathbb{x}^\star_1 \rrbracket_{\mathsf{tck}'}$ into $\llbracket \mathbb{y}' \rrbracket_{\mathsf{tck}'}$. Then, fold $\llbracket \mathbb{y}' \rrbracket_{\mathsf{tck}'}$ and $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'}$ into $\llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}$ by again running protocol $\mathit{\Pi}_{\mathsf{rr1cs}}$ with challenge $\alpha_2 \in \mathbb{F}$.
- Finally, compute $\llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5} := \llbracket \mathbf{s}_0 \rrbracket_{\mathsf{ck}_5} + \llbracket \mathbf{s}_1 \rrbracket_{\mathsf{ck}_5}$ and $\llbracket \mathbf{a} \rrbracket_{\mathsf{ck}_5} := \llbracket \mathbf{a}_0 \rrbracket_{\mathsf{ck}_5} + \alpha_1 \cdot \llbracket \mathbf{a}_1 \rrbracket_{\mathsf{ck}_5} + \alpha_1^2 \cdot (\llbracket \mathbf{s}_0 \rrbracket_{\mathsf{ck}_5} - \llbracket \mathbf{s}_1 \rrbracket_{\mathsf{ck}_5})$.

Description of protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$. Assuming pp, $\mathfrak{S} \in (\mathbb{F}^{n \times m})^3$, $\mathfrak{S}' \in (\mathbb{F}^{n' \times m'})^3$ are publicly determined by prover and verifier. We present the protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ in Cstr. 2 for folding $\llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}$ and $\llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}}$, with supporting secret vector $\mathbf{w} \in \mathcal{W}_{\mathsf{pvr\text{-}aux}}$ for condition, into $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}$.

Construction 2 (Protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$). We first parse

$$\begin{aligned}
\llbracket \mathbb{z}_i \rrbracket_{\mathsf{pp}} &= (\mathsf{lt}_i, \mathsf{rt}_i, \llbracket \mathbb{x}_i \rrbracket_{\mathsf{tck}}, \llbracket \mathbf{i}_i \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o}_i \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbb{x}^\star_i \rrbracket_{\mathsf{tck}'}, \llbracket \mathbf{s}_i \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a}_i \rrbracket_{\mathsf{ck}_5}) \ \forall i \in \{0, 1\},\\
\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} &= (\mathsf{lt}, \mathsf{rt}, \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}, \llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}, \llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a} \rrbracket_{\mathsf{ck}_5}).
\end{aligned}$$

We formally describe protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ as follows:

$\underline{\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}(\mathsf{pp}, \mathfrak{S}, \mathfrak{S}', \llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}, \llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}};\ \mathbf{w}) \to \llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}}$

1. Prover:
   (a) $\mathbf{g} \leftarrow \mathsf{garb}(\mathfrak{S}, \llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}, \llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}})$ (see Sec. 2.2 for use of garb).
   (b) $\mathbf{g}_1 \leftarrow \mathsf{garb}(\mathfrak{S}', \llbracket \mathbb{x}^\star_0 \rrbracket_{\mathsf{tck}'}, \llbracket \mathbb{x}^\star_1 \rrbracket_{\mathsf{tck}'})$.
2. Both parties run $\llbracket \mathbf{g} \rrbracket_{\mathsf{cke}} \leftarrow \mathit{\Pi}_{\mathsf{com}}(\mathsf{cke};\ \mathbf{g})$, $\llbracket \mathbf{w} \rrbracket_{\mathsf{ck}'_3} \leftarrow \mathit{\Pi}_{\mathsf{com}}(\mathsf{ck}'_3;\ \mathbf{w})$ and $\llbracket \mathbf{g}_1 \rrbracket_{\mathsf{cke}'} \leftarrow \mathit{\Pi}_{\mathsf{com}}(\mathsf{cke}';\ \mathbf{g}_1)$ where $\mathit{\Pi}_{\mathsf{com}}$ is defined in (1). Form the pair for condition
$$\begin{aligned}
\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'} \quad &= \quad (\mathbb{y}.u,\quad \mathbb{y}.\mathsf{pub},\quad \llbracket \mathbb{y}.\mathbf{z}_1 \rrbracket_{\mathsf{ck}'_1},\quad \llbracket \mathbb{y}.\mathbf{z}_2 \rrbracket_{\mathsf{ck}'_2},\quad \llbracket \mathbb{y}.\mathbf{z}_3 \rrbracket_{\mathsf{ck}'_3},\quad \llbracket \mathbb{y}.\mathbf{e} \rrbracket_{\mathsf{cke}'}\ )\\
&:= \quad (1,\qquad 1,\qquad \llbracket \mathbf{o}_0 \rrbracket_{\mathsf{ck}_2},\qquad \llbracket \mathbf{i}_1 \rrbracket_{\mathsf{ck}_1},\qquad \llbracket \mathbf{w} \rrbracket_{\mathsf{ck}'_3},\qquad \llbracket \mathbf{0}^{n'} \rrbracket_{\mathsf{cke}'}\ ).
\end{aligned}$$
   Notice that $\mathsf{ck}'_1 = \mathsf{ck}_2$ and $\mathsf{ck}'_2 = \mathsf{ck}_1$ as designed in Sec. 5.1 and the commitment $\llbracket \mathbf{0}^{n'} \rrbracket_{\mathsf{cke}'}$ can be determined by verifier alone, as in Rmk. 4.
3. Verifier: Abort if $\mathsf{rt}_0 \neq \mathsf{lt}_1$. Otherwise, $\alpha_1 \xleftarrow{\$} \mathbb{F}$ and send α_1 to prover.

4. Both parties run step 4 in $\mathit{\Pi}_{\mathsf{rr1cs}}$ (see Cstr. 1) to
   - fold $[\![\mathbb{x}_0]\!]_{\mathsf{tck}}$, $[\![\mathbb{x}_1]\!]_{\mathsf{tck}}$ into $[\![\mathbb{x}]\!]_{\mathsf{tck}}$ with α_1, and
   - fold $[\![\mathbb{x}^\star_0]\!]_{\mathsf{tck}'}$, $[\![\mathbb{x}^\star_1]\!]_{\mathsf{tck}'}$ into $[\![\mathbb{y}']\!]_{\mathsf{tck}'}$ with α_1.
5. Prover: Compute $\mathbf{g}_2 \leftarrow \mathsf{garb}(\mathfrak{S}', [\![\mathbb{y}']\!]_{\mathsf{tck}'}, [\![\mathbb{y}]\!]_{\mathsf{tck}'})$.
6. Both parties run $[\![\mathbf{g}_2]\!]_{\mathsf{cke}'} \leftarrow \mathit{\Pi}_{\mathsf{com}}(\mathsf{cke}';\ \mathbf{g}_2)$.
7. Verifier: $\alpha_2 \xleftarrow{\$} \mathbb{F}$ and send α_2 to prover.
8. Both parties run step 4 in $\mathit{\Pi}_{\mathsf{rr1cs}}$ to fold $[\![\mathbb{y}']\!]_{\mathsf{tck}'}$, $[\![\mathbb{y}]\!]_{\mathsf{tck}'}$ into $[\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}$ with α_2.
9. Finally, both parties compute the following

$$\begin{aligned}[\![\mathbf{i}]\!]_{\mathsf{ck}_1} &:= [\![\mathbf{i}_0]\!]_{\mathsf{ck}_1}, \quad & [\![\mathbf{s}]\!]_{\mathsf{ck}_5} &:= [\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} + [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5},\\ [\![\mathbf{o}]\!]_{\mathsf{ck}_2} &:= [\![\mathbf{o}_1]\!]_{\mathsf{ck}_2}, \quad & [\![\mathbf{a}]\!]_{\mathsf{ck}_5} &:= [\![\mathbf{a}_0]\!]_{\mathsf{ck}_5} + \alpha_1 \cdot [\![\mathbf{a}_1]\!]_{\mathsf{ck}_5} + \alpha_1^2 \cdot ([\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} - [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5}).\end{aligned}$$

CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$. The construction of CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ can be constructed from this protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ 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 now construct the CF scheme $\mathcal{CF}_{\mathsf{gnr}}[\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}, \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}] = (\mathsf{CF.Setup}, \mathsf{CF.Fold}, \mathsf{CF.Prove})$ for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ with respect to condition relation $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$, based on protocol $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ above. Its description is as follows:

Construction 3 (CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$). $\mathcal{CF}_{\mathsf{gnr}}$ is constructed as follows:

$\mathsf{CF.Setup}(1^\lambda) \to \mathsf{pp}$**:** This algorithm works as follows:
1. Sample $\mathsf{ck}_1, \ldots, \mathsf{ck}_5, \mathsf{cke}, \mathsf{ck}'_3, \mathsf{cke}'$ from $\mathsf{C.Setup}$. Set $\mathsf{ck}'_1 := \mathsf{ck}_2$, $\mathsf{ck}'_2 := \mathsf{ck}_1$.
2. Form tuples $\mathsf{tck} := (\mathsf{ck}_1, \ldots, \mathsf{ck}_5, \mathsf{cke})$ and $\mathsf{tck}' := (\mathsf{ck}'_1, \mathsf{ck}'_2, \mathsf{ck}'_3, \mathsf{cke}')$.
3. Return $\mathsf{pp} := (\mathsf{tck}, \mathsf{tck}')$.

$\mathsf{CF.Fold}(\mathsf{pp}, [\![\mathbb{z}_0]\!]_{\mathsf{tck}}, [\![\mathbb{z}_1]\!]_{\mathsf{tck}};\ \mathbf{w}) \to [\![\mathbb{z}]\!]_{\mathsf{tck}}$**:** Both parties run $\mathit{\Pi}_{\mathsf{fold\text{-}gnr}}$ (see Cstr. 2) as follows: $[\![\mathbb{z}]\!]_{\mathsf{tck}} \leftarrow \mathit{\Pi}_{\mathsf{fold\text{-}gnr}}(\mathsf{pp}, \mathfrak{S}, \mathfrak{S}', [\![\mathbb{z}_1]\!]_{\mathsf{tck}};\ \mathbf{w})$.

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

This CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ has an associated protocol

$$b \leftarrow \mathit{\Pi}_{\mathcal{CF}_{\mathsf{gnr}}}(\mathsf{HS}, \mathsf{pp}, \{[\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}\}_{i \in [N]};\ \mathsf{sec}) \tag{18}$$

that is described similarly to $\mathit{\Pi}_{\mathcal{CF}}$ in (8) where $\mathsf{HS} \in \mathcal{HS}_{0N}$.

Remark 6. In Sec. 6.3, $\mathsf{CF.Prove}$ will be implemented by $\mathit{\Pi}_{\mathsf{pvr\text{-}prf}}$ for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ while implies satisfaction of relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$.

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

Proof (Sketch). Completeness is straightforward. HVZK is implied from hiding property of $\mathcal{C}$ and HVZK from the employed protocol realizing $\mathsf{CF.Prove}$. Knowledge soundness follows the security proof sketched in Sec. 3.3. The full security proof is presented in Appdx. B.1. □

Efficiency. Let $\mathsf{c}(k)$ be the size of commitments to messages in $\mathbb{F}^k$ and p be the communication cost of $\mathsf{CF.Prove}$. The total communication cost of $\mathit{\Pi}_{\mathcal{CF}_{\mathsf{gnr}}}$ is $\mathcal{O}\left(N \cdot (\mathsf{c}(n) + \mathsf{c}(m'_3) + \mathsf{c}(n')) + \mathsf{p}\right)$.

Note that our $\mathcal{CF}_{\mathsf{gnr}}$ can be made to fold in parallel, i.e, following a binary tree of depth $\mathcal{O}(\log N)$. This incurs proving time $\mathcal{O}(|W| \log N)$ for prover when executing $\mathsf{CF.Fold}$ where $|W|$ is the size of witness in an instance-witness pair. Here, we assume that witness size is at least linear in instance size since instances in our scheme are a collection of commitments. Finally, let tp denote the prover time of $\mathsf{CF.Prove}$. Then the total prover time, when minimizing folding time by parallelism, is $\mathcal{O}(|W| \log N + \mathsf{tp})$.

The same argument can be applied for verifier cost. Let tv denote the verifier time when executing $\mathsf{CF.Fold}$. Then the total verifier cost, when minimizing folding time by parallelism, is $\mathcal{O}(|W| \log N + \mathsf{tv})$.

6 RAMenPaSTA: pvRAM from CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$

In this section, we describe RAMenPaSTA in Cstr. 4, a generic construction of pvRAM by employing the CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ from Sec. 5. Intuitively, observe that one can treat a RAM program [FKL+21] as an IVC with an associated memory, an instruction set $\mathcal{F} = \{F'_j\}_{j\in[T]}$ of T instructions and a program counter pc for determining the next instruction. We show how to transform each computation step of such RAM program into an instance-witness pair of a relation that allows one to employ CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ in Cstr. 2 for folding all such instance-witness pairs into a single one that testifies the correctness of the entire computation.

In Sec. 6.1, we describe a method to transform witnesses regarding each computation step of an execution of a RAM program into the forms of instance-witness pairs suitable for applying the CF scheme in $\mathcal{CF}_{\mathsf{gnr}}$. Then, in Section 6.2, we show partition the components introduced in Sec. 6.1 to fit the design of $\mathcal{CF}_{\mathsf{gnr}}$ in Sec. 5. Finally, in Sec. 6.3, we describe a generic construction built upon $\mathcal{CF}_{\mathsf{gnr}}$.

6.1 Handling Computation Steps

We present the technique for handling the components of RAM programs that, later in Sec. 6.2, we can partition these components and adapt to CF scheme $\mathcal{CF}_{\mathsf{gnr}}$. Let $N, T \in \mathbb{Z}_+$. To construct pvRAM for an N-step RAM program, we recall the components mentioned in Sec. 3.2 for handling (i) correct execution of each instruction, (ii) correctly selecting instruction by program counter and (iii) memory consistency (Sec. A.2). The RAM program has an instruction set $\mathcal{F} = \{F'_j\}_{j\in[T]}$ representable by the PLONK structures $\{\mathsf{plkst}'_j\}_{j\in[T]}$, initial values $\overline{\mathsf{pc}}_0 := 1$. For step $i \in [N]$, from (6), the components for its execution include

$$\overline{\mathsf{pc}}_i, \overline{\mathsf{reg}}_i, \overline{\mathsf{macs}}_i, \mathsf{plkst}_i, \mathsf{pc}_i, \mathsf{reg}_i, \mathsf{macs}_i, \mathsf{macs}'_i, \mathsf{plkiv}_i, \mathsf{miv}_i, \mathsf{miv}'_i \tag{19}$$

where $\overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}, \overline{\mathsf{reg}}_i = \mathsf{reg}_{i-1}, \overline{\mathsf{macs}}_i = \mathsf{macs}_{i-1}$. Here, $\overline{\mathsf{pc}}_i$ and $\overline{\mathsf{macs}}_i$ are output program counter and memory access from step $i-1$. plkst_i is the PLONK structure determined by selecting $\mathsf{plkst}_i := \mathsf{plkst}'_{\overline{\mathsf{pc}}_i}$ from the set $\{\mathsf{plkst}'_j\}_{j\in[T]}$. The outputs of step i include the program counter pc_i and memory access macs_i indicating the access to memory after step i. The sequence $(\mathsf{macs}'_j)_{j\in[N]}$ is a permutation of $(\mathsf{macs}_j)_{j\in[N]}$ for supporting proving memory consistency. Finally, $(\mathsf{plkiv}_j)_{j\in[N]}, (\mathsf{miv}_j)_{j\in[N]}$ and $(\mathsf{miv}'_j)_{j\in[N]}$ are to support proving PLONK structure lookup and permutation in memory consistency analyzed below.

Universally Realizing Instructions by PLONK Structures. As discussed in Sec. 3.2 (and Appdx. A.3), we universally realize each instruction by PLONK structures. Then, to verify computations of instructions work correctly, we apply system (38) in Appdx. A.3 with additional challenges, namely, $\gamma, \delta \xleftarrow{\$} \mathbb{F}$ for checking copy constraints.

PLONK Structure Lookup. For each $i \in [N]$, to guarantee that plkst_i is correctly determined from $\overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}$, namely, $\mathsf{plkst}_i = \mathsf{plkst}'_{\overline{\mathsf{pc}}_i}$, we establish the set $\{(j\|\mathsf{plkst}'_j)\}_{j\in[T]}$ and show that $(\overline{\mathsf{pc}}_i\|\mathsf{plkst}_i) \in \{(j\|\mathsf{plkst}'_j)\}_{j\in[T]}$. Hence, we need to apply a tuple lookup argument to guarantee that $\{(\overline{\mathsf{pc}}_i\|\mathsf{plkst}_i)\}_{i\in[N]} \subseteq \{(j\|\mathsf{plkst}'_j)\}_{j\in[T]}$. By applying [Hab22], adapted to polynomial version (see Appdx. A.4), we see that $\{(\overline{\mathsf{pc}}_i\|\mathsf{plkst}_i)\}_{i\in[N]} \subseteq \{(j\|\mathsf{plkst}'_j)\}_{j\in[T]}$ iff

$$\sum_{i=1}^{N} \frac{1}{X + \left\langle (\overline{\mathsf{pc}}_i\|\mathsf{plkst}_i), (Y^k)_{k=[0,n_{\mathsf{plk}}]} \right\rangle} = \sum_{j=1}^{T} \frac{\mathsf{mul}_j}{X + \left\langle (j\|\mathsf{plkst}'_j), (Y^k)_{k\in[0,n_{\mathsf{plk}}]} \right\rangle} \tag{20}$$

where each mul_j is determined by prover by counting multiplicity of $(j\|\mathsf{plkst}'_j)$ in $\{(\overline{\mathsf{pc}}_i, \mathsf{plkst}_i)\}_{i\in[N]}$. By sampling (χ, ψ) uniformly from $\mathbb{F}\times\mathbb{F}$ and setting $(X, Y) := (\chi, \psi)$, we see that, if $\{(\overline{\mathsf{pc}}_i\|\mathsf{plkst}_i)\}_{i\in[N]} \not\subseteq \{(j\|\mathsf{plkst}'_j)\}_{j\in[T]}$, then (20) holds with probability at most $\mathsf{err}_{\mathsf{lookup}}(\mathbb{F}, n_{\mathsf{plk}}+1, N, T) = \mathcal{O}(n_{\mathsf{plk}}(N+T)/|\mathbb{F}|)$, analyzed in Appdx. A.4. However, in (20), we may face division by zero for some bad choice of (χ, ψ). To avoid this, sample ψ, set $\boldsymbol{\psi} = \left(\psi^k\right)_{k\in[0,n_{\mathsf{plk}}]}$ and

$$\mathsf{plkcp}_i := \langle(\overline{\mathsf{pc}}_i\|\mathsf{plkst}_i), \boldsymbol{\psi}\rangle \; \forall i \in [N], \quad \mathsf{plkcp}'_j := \left\langle (j\|\mathsf{plkst}'_j), \boldsymbol{\psi}\right\rangle \; \forall j \in [T]. \tag{21}$$

One sees that (20) is reduced to show that $\sum_{i=1}^{N} \frac{1}{X+\mathsf{plkcp}_i} = \sum_{j=1}^{T} \frac{\mathsf{mul}_j}{X+\mathsf{plkcp}'_j}$, equivalent to proving $\{-\mathsf{plkcp}_i\}_{i\in[N]} \subseteq \{-\mathsf{plkcp}'_j\}_{j\in[T]}$. Hence, by sampling $\chi \xleftarrow{\$} \mathbb{F}\setminus(\{-\mathsf{plkcp}_i\}_{i\in[N]} \cup \{-\mathsf{plkcp}'_j\}_{j\in[T]})$, we can avoid division by zero. However, in executing protocol, verifier determines ψ and only knows $\{-\mathsf{plkcp}'_j\}_{j\in[T]}$. Moreover, since prover tries to prove $\{-\mathsf{plkcp}_i\}_{i\in[N]} \subseteq \{-\mathsf{plkcp}'_j\}_{j\in[T]}$, it suffices for verifier to sample $\chi \xleftarrow{\$} \mathbb{F} \setminus \{-\mathsf{plkcp}'_j\}_{j\in[T]}$ avoiding division by zero. By setting

$$\mathsf{plkiv}_i = (\chi + \mathsf{plkcp}_i)^{-1}\ \forall i \in [N], \quad \mathsf{plkiv}'_j = \mathsf{mul}_j \cdot (\chi + \mathsf{plkcp}'_j)^{-1}\ \forall j \in [T], \tag{22}$$

testing $\{(\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i)\}_{i\in[N]} \subseteq \{(j\|\mathsf{plkst}'_j)\}_{j\in[T]}$ can be deduced to checking that

$$\sum_{i=1}^{N} \mathsf{plkiv}_i = \sum_{j=1}^{T} \mathsf{plkiv}'_j \tag{23}$$

holds with error probability at most $\mathsf{err}_{\mathsf{lookup}}(\mathbb{F}, n_{\mathsf{plk}}+1, N, T)$.

Finally, as we introduced $\mathsf{plkcp}_i\ \forall i \in [N]$, we also need to make sure that $\mathsf{plkiv}_i \cdot (\chi + \mathsf{plkcp}_i) = 1$ when specify constraints.

Memory Access Permutation. Recall that, $\forall i \in [N]$, $\mathsf{macs}_i, \mathsf{macs}'_i \in \mathbb{F}^4$. We sketch the technique for proving that $(\mathsf{macs}_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}'_i)_{i\in[N]}$. This can be done by extending [Hab22] to show that

$$\sum_{i=1}^{N} \frac{1}{X + \langle \mathsf{macs}_i, (Y^k)_{k=0}^3 \rangle} = \sum_{i=1}^{N} \frac{1}{X + \langle \mathsf{macs}'_i, (Y^k)_{k=0}^3 \rangle}. \tag{24}$$

See Appdx. A.4 for the use of (24). By sampling (τ, ω) uniformly from $\mathbb{F} \times \mathbb{F}$ and setting $(X, Y) := (\tau, \omega)$, if $(\mathsf{macs}'_i)_{i\in[N]}$ is not a permutation of $(\mathsf{macs}_i)_{i\in[N]}$, then (24) holds with probability at most $\mathsf{err}_{\mathsf{perm}}(\mathbb{F}, 4, N) = \mathcal{O}(N/|\mathbb{F}|)$. As of the case of PLONK structure lookup, we may face the issue of division by zero. However, in this case, both $(\mathsf{macs}_i)_{i\in[N]}$ and $(\mathsf{macs}'_i)_{i\in[N]}$ are unknown to verifier. Hence, verifier cannot sample challenges avoiding division by zero. To this end, let $\omega \xleftarrow{\$} \mathbb{F}$ be determined first. Then, set

$$\mathsf{mcp}_i := \langle \mathsf{macs}_i, \boldsymbol{\omega} \rangle\ ,\ \mathsf{mcp}'_i := \langle \mathsf{macs}'_i, \boldsymbol{\omega} \rangle\ \ \forall i \in [N]. \tag{25}$$

If we sample $\tau \xleftarrow{\$} \mathbb{F}$, with probability at most $2N/|\mathbb{F}|$, $-\tau \in \{\mathsf{mcp}_i\}_{i\in[N]} \cup \{\mathsf{mcp}'_i\}_{i\in[N]}$ incurring division by zero. We call τ in this case to be *bad* τ. This restrains prover to prove and hence compromises HVZK property. To avoid this, we need to devise a way to allow prover to continue proving even bad τ occurs. Observe that $b = a^{-1} \iff b \in \mathbb{F}^\star \wedge a \cdot b \in \{0,1\}\ \forall a \in \mathbb{F}^\star$ where $\mathbb{F}^\star = \mathbb{F} \setminus \{0\}$. Therefore, by enforcing prover to firstly sample $\mathsf{iv} \xleftarrow{\$} \mathbb{F}^\star$ and then setting

$$\mathsf{miv}_i := \mathsf{iv} \text{ if } \tau + \mathsf{mcp}_i = 0 \text{ and } \mathsf{miv}_i := (\tau + \mathsf{mcp}_i)^{-1}, \text{otherwise}, \tag{26}$$

testing $\sum_{i=1}^{N} \mathsf{miv}_i = \sum_{i=1}^{N} \mathsf{miv}'_i$ probabilistically deduces that $(\mathsf{macs}_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}'_i)_{i\in[N]}$ without making prover stop in the middle when bad τ occurs. Let us precisely explain this fact. Obviously, it is straightforward to see that, for any $i \in [N]$ satisfying $\mathsf{mcp}_i \neq -\tau$, one has the equivalence

$$\mathsf{miv}_i = (\tau + \mathsf{mcp}_i)^{-1} \iff \mathsf{miv}_i \neq 0 \wedge (\tau + \mathsf{mcp}_i) \cdot \mathsf{miv}_i \in \{0,1\}. \tag{27}$$

Showing RHS of (27) implies that miv_i is exactly the inverse of $\tau + \mathsf{mcp}_i$. If $\mathsf{mcp}_i = -\tau$, showing RHS of (27) only implies that miv_i is some value in $\mathbb{F}^\star$. A similar process applies to computing $\{\mathsf{miv}'_i\}_{i\in[N]}$ from τ and $\{\mathsf{mcp}'_i\}_{i\in[N]}$. One sees that, in case that bad τ occurs, prover can still continue to prove even soundness does not hold anymore. This helps protect HVZK of protocol. Moreover, as discussed previously, bad τ occurs with negligible probability, namely, $\mathcal{O}(N/|\mathbb{F}|)$. Therefore, bad τ only affects to soundness error of the protocol.

Sampling Global Challenges. Regarding the use of challenges χ, ψ, τ and ω, we define the the distributions $\mathrm{CH}_{\mathsf{cp}}$ and $\mathrm{CH}_{\mathsf{test}}$ s.t.

$$\begin{aligned} (\psi, \omega) \leftarrow \mathrm{CH}_{\mathsf{cp}} &\iff (\psi, \omega) \xleftarrow{\$} \mathbb{F} \times \mathbb{F}, \text{ and} \\ (\chi, \tau) \leftarrow \mathrm{CH}^{\psi}_{\mathsf{test}} &\iff (\chi, \tau) \xleftarrow{\$} \left(\mathbb{F} \setminus \{-\mathsf{plkcp}'_j\}_{j\in[T]}\right) \times \mathbb{F} \end{aligned} \tag{28}$$

where $\{-\mathsf{plkcp}'_j\}_{j\in[T]}$ can be computed according to (21) when ψ is known.

Putting Everything Together. With global challenges γ, δ, χ, ψ, τ and ω, pre-computed values $(\mathsf{mul}_i)_{i\in[T]} \in \mathbb{F}$ and the witness defined in (6), whose components are described in detail above, we now partition these components in a way that we can devise a CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ for folding the computation steps of the RAM program.

Global challenges. Set $\mathsf{gchal} = (\gamma, \delta, \chi, \psi, \tau, \omega)$ to contain global challenges.

Constraints for single computation steps. For each $i \in [N]$, one must satisfy

- correct computation of F_i involving $\overline{\mathsf{reg}}_i$,$\overline{\mathsf{macs}}_i$,$\mathsf{aux}_i$,$\mathsf{plkst}_i$,$\mathsf{pc}_i$,$\mathsf{reg}_i$,$\mathsf{macs}_i$,$\gamma$,$\delta$;
- constraints in macs_i hold, e.g., access index is in allowed range $[M]$;
- the correct computation of plkiv_i from $(\overline{\mathsf{pc}}_i, \mathsf{plkst}_i)$, χ and ψ, as in (22); and
- the correct computation of miv_i and miv'_i from $\mathsf{macs}_i, \mathsf{macs}'_i$, τ and ω, i.e., satisfying the RHS of (27).

Therefore, the constraints for a single computation step involve $\overline{\mathsf{pc}}_i$, $\overline{\mathsf{reg}}_i$, $\overline{\mathsf{macs}}_i$, aux_i, plkst_i, pc_i, reg_i, macs_i, plkiv_i, miv_i and miv'_i for all $i \in [N]$ and global challenges γ, δ, χ, ψ, τ and ω.

Constraints for pairs of consecutive computation steps. These constraints involve the consistency checking of memory in (34) with respect to the tuples $(\mathsf{macs}_{i-1}, \mathsf{macs}'_{i-1})$ and $(\mathsf{macs}_i, \mathsf{macs}'_i)$ for all $i \in [2, N]$. Moreover, we guarantee $(\mathsf{pc}_{i-1}, \mathsf{reg}_{i-1}, \mathsf{macs}_{i-1}) = (\overline{\mathsf{pc}}_i, \overline{\mathsf{reg}}_i, \overline{\mathsf{macs}}_i)$ for all $i \in [2, N]$.

Constraints for all computation steps. These constraints involve checking that

- the correct use of plkst_i with respect to $\overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}$ by checking the lookup $\{(\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i)\}_{i\in[N]} \subseteq \{(j \| \mathsf{plkst}'_j)\}_{j\in[T]}$ as in (23); and
- $(\mathsf{macs}'_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}_i)_{i\in[N]}$, as in discussion above.

6.2 Partitioning Components and Adapting to CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$

In this section, we first discuss how to put those components Sec. 6.1 into instance-witness pairs suitable for applying CF scheme in Sec. 5.3 as a building block. Then, we specify the necessary relations folding, as required in Sec. 5.2.

For step $i \in [N]$, recall the components discussed above including:

$$\overline{\mathsf{pc}}_i, \overline{\mathsf{reg}}_i, \overline{\mathsf{macs}}_i, \mathsf{aux}_i, \mathsf{plkst}_i, \mathsf{pc}_i, \mathsf{reg}_i, \mathsf{macs}_i, \mathsf{macs}'_i, \mathsf{plkiv}_i, \mathsf{miv}_i, \mathsf{miv}'_i.$$

Hence, we partition these components into vectors $\mathbf{z}_{i1}, \ldots, \mathbf{z}_{i5}$ over $\mathbb{F}$ s.t.

$$\begin{aligned} &\mathbf{z}_{i1} = (\overline{\mathsf{pc}}_i \| \overline{\mathsf{reg}}_i \| \overline{\mathsf{macs}}_i \| \mathsf{macs}'_i), \quad \mathbf{z}_{i2} = (\mathsf{pc}_i \| \mathsf{reg}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i), \\ &\mathbf{z}_{i3} = \mathsf{plkst}_i, \quad \mathbf{z}_{i4} = \mathsf{aux}_i, \quad \mathbf{z}_{i5} = (\mathsf{plkiv}_i \| \mathsf{miv}_i \| \mathsf{miv}'_i) \end{aligned} \tag{29}$$

for all $i \in [N]$. Here, $\mathbf{z}_{i1}$ and $\mathbf{z}_{i2}$ represent the input and output of the computation, respectively. Moreover, as macs'_i supports the consistency of memory, we put macs'_i into $\mathbf{z}_{i2}$. The vector $\mathbf{z}_{i3}$ represents the PLONK structure for the corresponding instruction while $\mathbf{z}_{i4}$ is the auxiliary witness supporting the computation from $\mathbf{z}_{i1}$ to $\mathbf{z}_{i2}$ with respect to PLONK structure $\mathbf{z}_{i3}$. Finally, $\mathbf{z}_{i5}$ is to support tuple lookup and permutation testings. Notice that, $\mathbf{z}_{i5}$ cannot be determined directly from $\mathbf{z}_{i1}, \ldots, \mathbf{z}_{i4}$. As clarified in Section 6.1, we need additional challenges χ, ψ, τ, ω in gchal determined in advance s.t. relationship between $\mathbf{z}_{i5}$ and $\mathbf{z}_{i1}, \ldots, \mathbf{z}_{i4}$ is captured.

Putting into Instance-Witness Pairs for CF Scheme. Let HS_{0N} be a hierarchical structure. Since our final goal is to construct pvRAM for an N-step RAM program, by employing generic CF scheme in Sec. 5.3 as a subroutine and by using indices, e.g., subscript "$(i-1)i$", corresponding to leaf nodes of HS_{0N}, we need to appropriately design

$$\mathbb{z}_{(i-1)i} = (\mathsf{lt}_{(i-1)i}, \mathsf{rt}_{(i-1)i}, \mathbb{x}_{(i-1)i}, \mathbf{i}_{(i-1)i}, \mathbf{o}_{(i-1)i}, \mathbb{x}^\star_{(i-1)i}, \mathbf{s}_{(i-1)i}, \mathbf{a}_{(i-1)i}),$$

of the form (11), with those components in (29) in the following way:

$$\begin{aligned} &\mathsf{lt}_{(i-1)i} := i-1, \mathsf{rt}_{(i-1)i} := i, \ \mathbf{i}_{(i-1)i} := \mathbf{z}_{i1}, \ \mathbf{o}_{(i-1)i} := \mathbf{z}_{i2}, \\ &\mathbf{s}_{(i-1)i} := \mathbf{z}_{i5}, \mathbf{a}_{(i-1)i} := \mathbf{0}^3, \\ &\mathbb{x}_{(i-1)i} = (\mathbb{x}_{(i-1)i}.u, \ \mathbb{x}_{(i-1)i}.\mathsf{pub}, \ \mathbb{x}_{(i-1)i}.\mathbf{z}_1, \ \ldots, \ \mathbb{x}_{(i-1)i}.\mathbf{z}_5, \ \mathbb{x}_{(i-1)i}.\mathbf{e}) \\ &\qquad\quad := (1, \ 1, \ \mathbf{z}_{i1}, \ \ldots, \ \mathbf{z}_{i5}, \ \mathbf{0}^n) \end{aligned}$$

while the design of $\mathbb{x}^\star_{(i-1)i}$ is set to be any random tuple such that combination of components satisfies rR1CS specified by matrices $\mathbf{A}', \mathbf{B}', \mathbf{C}'$ defined later. This acts as a accumulator for conditions when folding. We defer the detailed setting of this $\mathbb{x}^\star_{(i-1)i}$ to Cstr. 4 in Sec. 6.3. Hence, with $\mathsf{pp} = (\mathsf{tck}, \mathsf{tck}')$ defined as in (10), we can establish $[\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}}$ of the form (12).

Determining Matrices for rR1CS. To specify necessary relations, e.g., $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ and $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ in (16) and (17), respectively, for folding, we need to specify the tuple of matrices $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C})$, after global challenges in gchal are determined, for capturing constraints for single computation steps, discussed in Sec. 6.1. In particular, for each $i \in [N]$, by defining $\mathbf{z}_i = (1\|\mathbf{z}_{i1}\| \ldots \|\mathbf{z}_{i5})$, we would like to capture that $\mathbf{A} \cdot \mathbf{z}_i \circ \mathbf{B} \cdot \mathbf{z}_i = \mathbf{C} \cdot \mathbf{z}_i$ iff all constraints in computation step i hold. Similarly, based on constraints for pairs of consecutive computation steps, discussed in Sec. 6.1, we can specify tuple $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$ capturing conditions between $\mathbf{o}_{(i-1)i}$ and $\mathbf{i}_{i(i+1)}$ for all $i \in [N-1]$. See Appdx. C.1 for more concrete constraints that $\mathfrak{S}$ and $\mathfrak{S}'$ capture.

With $\mathfrak{S}$ and $\mathfrak{S}'$ determined and all $\mathbb{z}_{(i-1)i}$, for all $i \in [N]$, specified above, in this section, namely, Sec. 6.2, we can directly use relations $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ and $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ in (16) and (17), respectively.

Realizing CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$ for pvRAM. As we discussed all instance-witness pairs, tuples of matrices $\mathfrak{S}, \mathfrak{S}'$ and relations $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ and $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$ above, we can realize CF scheme in Cstr. 3 for pvRAM.

We are ready to discuss the complete construction of pvRAM in Sec. 6.3.

6.3 Description of Generic Construction of pvRAM

In this section, we present our construction of pvRAM by protocol Π_{pvr} constructed in Cstr. 4. The problem statement is as follow. Let pp be the tuple of commitment keys, described in (10). Let $\mathcal{F} = \{F'_j\}_{j\in[T]}$ be an instruction set of T instructions describable by the PLONK structures $\mathcal{F}_{\mathsf{struct}} = \{\mathsf{plkst}'_j\}_{j\in[T]}$. Given an output register $\mathsf{reg}_{\mathsf{out}}$ and commitment tuple $[\![\mathsf{reg}_{\mathsf{in}}]\!]_{\mathsf{cki}}$ to secret input register $\mathsf{reg}_{\mathsf{in}}$ with commitment key cki, prover proceeds a interactive argument with verifier for the statement that $\mathsf{reg}_{\mathsf{out}}$ is the output of an N-step execution of the RAM program with instruction set $\mathcal{F}$ on committed input register $\mathsf{reg}_{\mathsf{in}}$. See Sec. 3.2 and Appdx. A.1 for description of RAM program. This problem statement can be formalized by relation $\mathcal{R}_{\mathsf{ram}}$ in (30) below.

$$\mathcal{R}_{\mathsf{ram}} = \left\{(\mathsf{cki}, \mathcal{F}_{\mathsf{struct}}, [\![\mathsf{reg}_{\mathsf{in}}]\!]_{\mathsf{cki}}, \mathsf{reg}_{\mathsf{out}}) \middle| [\![\mathsf{reg}_{\mathsf{in}}]\!]_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}} \wedge \mathsf{RAM}(\mathsf{reg}_{\mathsf{in}}) = \mathsf{reg}_{\mathsf{out}}\right\} \tag{30}$$

where $\mathcal{F}_{\mathsf{struct}} = \{\mathsf{plkst}'_j\}_{j\in[T]}$ and RAM is an N-step RAM program and with instruction set $\mathcal{F}$ s.t. RAM takes as input $\mathsf{reg}_{\mathsf{in}}$ and returns $\mathsf{reg}_{\mathsf{out}}$.

We discuss the intuition regarding the construction of Π_{pvr}. Notice that, since prover's input is the execution trace of RAM , prover can compute $(\mathbf{z}_{ij})_{i\in[N],j\in[3]}$, of the form (29). Let c_{reg} be the register length. Parsing $\mathbf{z}_{11} = (\overline{\mathsf{pc}}_1\|\overline{\mathsf{reg}}_1\|\overline{\mathsf{macs}}_1)$ and $\mathbf{z}_{N2} = (\mathsf{pc}_N\|\mathsf{reg}_N\|\mathsf{macs}_N\|\mathsf{macs}'_N)$, let $\mathsf{reg}_{\mathsf{in}} = \overline{\mathsf{reg}}_1, \mathsf{reg}_{\mathsf{out}} = \mathsf{reg}_N \in \mathbb{F}^{c_{\mathsf{reg}}}$ to be the input and output of RAM as the input to the first and output to the last instructions, respectively. At this point, prover has no global challenges, namely, $\gamma, \delta, \chi, \psi, \tau$ and ω. Prover hence cannot compute $\mathbf{z}_{i4}$ and $\mathbf{z}_{i5}$ for all $i \in [N]$.

On the other hand, for ensuring the correct lookup of PLONK structures, prover can compute $(\mathsf{mul}_i)_{i\in[T]}$, as in (20), for supporting the lookup argument. Hence, with pp (see (10)) determined in advance, we first force prover and verifier to run Π_{com} (see Rmk. 1) to obtain $\{[\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1}, [\![\mathbf{z}_{i2}]\!]_{\mathsf{ck}_2}, [\![\mathbf{z}_{i3}]\!]_{\mathsf{ck}_3}\}_{i\in[N]}$, $([\![\mathsf{mul}_j]\!]_{\mathsf{ckm}_j})_{j\in[T]}$ where $\mathsf{ck}_1, \mathsf{ck}_2$ and ck_3 belong to pp and $(\mathsf{ckm}_j)_{j\in[T]}$ are commitment keys employed for committing to multiplicities $\{\mathsf{ckm}_j\}_{j\in[T]}$.

Then, verifier samples global challenge $\mathsf{gchal} = (\gamma, \delta, \chi, \psi, \tau, \omega)$ from distribution $\mathrm{CH}_{\mathsf{global}}$, defined according to the equivalence

$$\mathsf{gchal} \leftarrow \mathrm{CH}_{\mathsf{global}} \iff (\gamma, \delta \xleftarrow{\$} \mathbb{F}) \wedge ((\psi, \omega) \leftarrow \mathrm{CH}_{\mathsf{cp}}) \wedge ((\chi, \tau) \leftarrow \mathrm{CH}^{\psi}_{\mathsf{test}}) \tag{31}$$

where $\mathrm{CH}_{\mathsf{cp}}$ and $\mathrm{CH}^{\psi}_{\mathsf{test}}$ are defined in (28). Verifier then sends gchal to prover.

With gchal, prover can compute $(\mathbf{z}_{i4}, \mathbf{z}_{i5})_{i\in[N]}$ of the form (29). Prover and verifier then run Π_{com} to obtain $\{[\![\mathbf{z}_{i4}]\!]_{\mathsf{ck}_4}, [\![\mathbf{z}_{i5}]\!]_{\mathsf{ck}_5}\}_{i\in[N]}$ where ck_4 and ck_5 belong to pp. Moreover, both parties can independently determine $(\mathsf{plkiv}'_j)_{j\in[T]}$, as in (21) and (22), since $(\mathsf{plkst}'_j)_{j\in[T]}$ is public and ψ, χ in gchal are determined. Prover now has sufficient information to construct $([\![\mathbb{z}_{(i-1)i}]\!]_{\mathsf{pp}})_{i\in[N]}$ discussed in Sec. 6.2 above since they have $\{[\![\mathbf{z}_{i1}]\!]_{\mathsf{ck}_1}, \ldots, [\![\mathbf{z}_{i5}]\!]_{\mathsf{ck}_5}\}_{i\in[N]}$.

Finally, both parties start folding $(\llbracket \mathbb{z}_{(i-1)i} \rrbracket_{\mathsf{pp}})_{i\in[N]}$ by running protocol $\mathsf{CF.Fold}$ in $\mathcal{CF}_{\mathsf{gnr}}$ following a common hierarchical structure $\mathsf{HS} \in \mathcal{HS}_{0N}$ in Def. 2 to obtain a final folded tuple $\llbracket \mathbb{z}_{0N} \rrbracket_{\mathsf{pp}}$. Finally, prover conducts a proof/argument $\Pi_{\mathsf{pvr\text{-}prf}}$ for the statement that

$$(\mathsf{pp}, \mathsf{cki}, (\mathsf{ckm}_j)_{j\in[T]}, \llbracket \mathsf{reg}_{\mathsf{in}} \rrbracket_{\mathsf{cki}}, \mathsf{reg}_{\mathsf{out}}, (\llbracket \mathsf{mul}_j \rrbracket_{\mathsf{ckm}_j})_{j\in[T]}, \llbracket \mathbb{z}_{0N} \rrbracket_{\mathsf{pp}}) \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$$

defined to be

$$\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}} = \left\{ \begin{array}{c|l} \begin{array}{l} (\mathsf{pp}, \mathsf{cki}, (\mathsf{ckm}_j)_{j\in[T]}, \\ \llbracket \mathsf{reg}_{\mathsf{in}} \rrbracket_{\mathsf{cki}}, \mathsf{reg}_{\mathsf{out}}, \\ (\llbracket \mathsf{mul}_j \rrbracket_{\mathsf{ckm}_j})_{j\in[T]}, \\ \llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}) \end{array} & \begin{array}{l} \llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} \wedge \llbracket \mathsf{reg}_{\mathsf{in}} \rrbracket_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}} \\ \wedge \overline{\mathsf{pc}} = 1 \wedge \overline{\mathsf{reg}} = \mathsf{reg}_{\mathsf{in}} \wedge \mathsf{reg} = \mathsf{reg}_{\mathsf{out}} \\ \wedge (\llbracket \mathsf{mul}_j \rrbracket_{\mathsf{ckm}_j} \in \mathcal{R}_{\mathsf{com}} \ \forall j \in [T]) \\ \wedge \mathsf{plkiv} = \sum_{j\in[T]} \frac{\mathsf{mul}_j}{\chi + \mathsf{plkcp}'_j} \wedge \mathsf{miv} = \mathsf{miv}' \\ \wedge \mathsf{macs}^\star = \mathsf{macs}' \end{array} \end{array} \right\} \tag{32}$$

where

- $\mathbb{z} = (\mathsf{lt}, \mathsf{rt}, \mathbb{x}, \mathbf{i}, \mathbf{o}, \mathbb{x}^\star, \mathbf{s}, \mathbf{a})$, of the form (11), and $\mathbf{s} = (\mathsf{plkiv} \| \mathsf{miv} \| \mathsf{miv}')$ in $\mathbb{z}$;
- $\mathbf{i} = (\overline{\mathsf{pc}} \| \overline{\mathsf{reg}} \| \overline{\mathsf{macs}} \| \mathsf{macs}^\star)$ and $\mathbf{o} = (\mathsf{pc} \| \mathsf{reg} \| \mathsf{macs} \| \mathsf{macs}')$ in $\mathbb{z}$;
- $\mathsf{reg}_{\mathsf{in}}$ and $\mathsf{reg}_{\mathsf{out}}$ are input and output registers of RAM program;
- $\mathsf{plkcp}'_j = \langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k\in[0,n_{\mathsf{plk}}]} \rangle \ \forall j \in [T]$ (see (21)) from public $\{\mathsf{plkst}'_j\}_{j\in[T]}$.

We are now ready to construct protocol Π_{pvr} in the following Cstr. 4.

Construction 4 (RAMenPaSTA as Protocol Π_{pvr}). With the parameters $m_1, \ldots, m_5$, n, $m'_1, \ldots, m'_5$, n' and

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

satisfying $\mathsf{ck}'_1 = \mathsf{ck}_2$ and $\mathsf{ck}'_2 = \mathsf{ck}_1$, determined as in Sec. 5.1, protocol Π_{pvr} runs as follows:

$\underline{\Pi_{\mathsf{pvr}} \left(\mathsf{pp}, \mathsf{cki}, (\mathsf{ckm}_j)_{j\in[T]}, \mathcal{F}_{\mathsf{struct}}, \llbracket \mathsf{reg}_{\mathsf{in}} \rrbracket_{\mathsf{cki}}, \mathsf{reg}_{\mathsf{out}};\ (\mathbf{z}_{ij})_{i\in[N], j\in[3]}, (\mathsf{mul}_j)_{j\in[T]} \right) \to \{0,1\}}$

1. Both parties run protocols
$$\begin{aligned} &\llbracket \mathbf{z}_{ij} \rrbracket_{\mathsf{ck}_j} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ck}_j;\ \mathbf{z}_{ij}) \ \forall i \in [N], \forall j \in [3], \\ &\llbracket \mathbf{0}^n \rrbracket_{\mathsf{cke}} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke};\ \mathbf{0}^n), \quad \llbracket \mathbf{0}^{n'} \rrbracket_{\mathsf{cke}'} \leftarrow \Pi_{\mathsf{com}}(\mathsf{cke}';\ \mathbf{0}^{n'}), \\ &\llbracket \mathbf{0}^3 \rrbracket_{\mathsf{ck}_5} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ck}_5, \mathbf{0}^3), \quad \llbracket \mathsf{mul}_j \rrbracket_{\mathsf{ckm}_j} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ckm}_j;\ \mathsf{mul}_j) \ \forall j \in [T] \end{aligned}$$
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 Rmk. 4.
2. Verifier: $\mathsf{gchal} \leftarrow \mathrm{CH}_{\mathsf{global}}$ and send gchal to prover.
3. Each party determines $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C}), \mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$ from gchal.
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 gchal.
   - Find arbitrary $\{\mathbf{z}'_i\}_{i\in[3]}$ s.t. $\mathbf{z}'_i \in \mathbb{F}^{m'_i}$ and $\mathbf{A}' \cdot \mathbf{z}' \circ \mathbf{B}' \cdot \mathbf{z}' = \mathbf{C} \cdot \mathbf{z}'$ where $\mathbf{z}' = (1 \| \mathbf{z}'_1 \| \mathbf{z}'_2 \| \mathbf{z}'_3)$.
5. Both parties run $\llbracket \mathbf{z}'_j \rrbracket_{\mathsf{ck}'_j} \leftarrow \Pi_{\mathsf{com}}(\mathsf{ck}'_j;\ \mathbf{z}'_j) \ \forall\ j \in [3]$.
6. $\forall i \in [N]$, both parties form tuple $\llbracket \mathbb{z}_{(i-1)i} \rrbracket_{\mathsf{pp}}$, of the form (12), by setting
$$\begin{aligned} &\mathsf{lt}_{(i-1)i} := i - 1,\ \mathsf{rt}_{(i-1)i} := i, \\ &\llbracket \mathbf{i}_{(i-1)i} \rrbracket_{\mathsf{ck}_1} := \llbracket \mathbf{z}_{i1} \rrbracket_{\mathsf{ck}_1},\ \llbracket \mathbf{o}_{(i-1)i} \rrbracket_{\mathsf{ck}_2} := \llbracket \mathbf{z}_{i2} \rrbracket_{\mathsf{ck}_2}, \\ &\llbracket \mathbb{x}_{(i-1)i} \rrbracket_{\mathsf{tck}} := (1, 1, \llbracket \mathbf{z}_{i1} \rrbracket_{\mathsf{ck}_1}, \ldots, \llbracket \mathbf{z}_{i5} \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{0}^n \rrbracket_{\mathsf{cke}}), \\ &\llbracket \mathbb{x}^\star_{(i-1)i} \rrbracket_{\mathsf{tck}'} := (1, 1, \llbracket \mathbf{z}'_{i1} \rrbracket_{\mathsf{ck}'_1}, \ldots, \llbracket \mathbf{z}'_{i3} \rrbracket_{\mathsf{ck}'_3}, \llbracket \mathbf{0}^{n'} \rrbracket_{\mathsf{cke}'}), \\ &\llbracket \mathbf{s}_{(i-1)i} \rrbracket_{\mathsf{ck}_5} := \llbracket \mathbf{z}_{i5} \rrbracket_{\mathsf{ck}_5}, \quad \llbracket \mathbf{a}_{(i-1)i} \rrbracket_{\mathsf{ck}_5} := \llbracket \mathbf{0}^3 \rrbracket_{\mathsf{ck}_5} \end{aligned}$$
where $\llbracket \mathbb{x}_{(i-1)i} \rrbracket_{\mathsf{tck}}$ and $\llbracket \mathbb{x}^\star_{(i-1)i} \rrbracket_{\mathsf{tck}'}$ are of the forms (13) and (14), respectively.
7. Both parties run $\Pi_{\mathcal{CF}_{\mathsf{gnr}}}(\mathsf{HS}, \mathsf{pp}, \{\llbracket \mathbb{z}_{(i-1)i} \rrbracket_{\mathsf{pp}}\}_{i\in[N]};\ \mathsf{sec}) \to \{0,1\}$ defined in (18) to obtain $\llbracket \mathbb{z}_{0N} \rrbracket_{\mathsf{pp}}$ and run $\mathsf{CF.Prove}$ as a ZKAoK $\Pi_{\mathsf{pvr\text{-}prf}}$ for showing
$$(\mathsf{pp}, \mathsf{cki}, (\mathsf{ckm}_j)_{j\in[T]}, \llbracket \mathsf{reg}_{\mathsf{in}} \rrbracket_{\mathsf{cki}}, \mathsf{reg}_{\mathsf{out}}, (\llbracket \mathsf{mul}_j \rrbracket_{\mathsf{ckm}_j})_{j\in[T]}, \llbracket \mathbb{z}_{0N} \rrbracket_{\mathsf{pp}}) \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}.$$

Theorem 2 (Security of Π_{pvr}). *If $\mathcal{C}$ is secure homomorphic commitment scheme and $\Pi_{\mathsf{pvr\text{-}prf}}$ is an HVZKAoK/HVZKPoK for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ then Π_{pvr} is an HVZKAoK/HVZKPoK for relation $\mathcal{R}_{\mathsf{ram}}$ in (30) with soundness error*

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

where $\mathsf{serr}_{\mathsf{pvr\text{-}prf}}(\mathsf{pp})$ is soundness error of $\Pi_{\mathsf{pvr\text{-}prf}}$.

The security of Π_{pvr} follows from the security of the underlying commitment scheme, CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ and HVZKAoK/HVZKPoK. For knowledge soundness, we can extract the witnesses of Π_{pvr} from the extractor of $\mathcal{CF}_{\mathsf{gnr}}$ according to Thm 1 and analyze the extracted components matching required constraints of RAM programs. The full proof of Thm 2 will be presented in Appdx. C.2.
Efficiency. Let $\mathsf{c}(k)$ be the size of commitments to messages in $\mathbb{F}^k$ and p be the communication cost of $\Pi_{\mathsf{pvr\text{-}prf}}$. The total communication cost of Π_{pvr} is

$$\mathcal{O}\left(N \cdot ((\mathsf{c}(m_i) + \cdots + \mathsf{c}(m_5)) + \mathsf{c}(n) + \mathsf{c}(m_3') + \mathsf{c}(n')) + T \cdot \mathsf{c}(1) + \mathsf{p}\right).$$

Regarding prover time, we assume that prover already sorted memory accesses, in witness, to satisfy condition (33). It can be seen that prover time is $\mathcal{O}(T \cdot \mathsf{tp}_1 + N \cdot \mathsf{tp}_2 + \mathsf{tp})$ where tp_1 is the time for computing and committing to each mul_j in $\{\mathsf{mul}_j\}_{j\in[T]}$, tp_2 is the time for each folding in $\Pi_{\mathcal{CF}_{\mathsf{gnr}}}$ and tp is the proving time of $\mathsf{CF.Prove}$ in $\mathcal{CF}_{\mathsf{gnr}}$. Moreover, since each mul_j can be computed and committed independently and the foldings are following a hierarchical structure $\mathsf{HS} \in \mathcal{HS}_{0N}$, if HS is of the form of depth $\mathcal{O}(\log N)$, then, when parallelizing into $\max\{N, T\}$ prover threads, prover time can be optimized to $\mathcal{O}(\log T \cdot \mathsf{tp}_1 + \log N \cdot \mathsf{tp}_2 + \mathsf{tp})$, which is sub-linear in computation time.

A similar argument applies to verifier time $\mathcal{O}(T \cdot \mathsf{tv}_1 + N \cdot \mathsf{tv}_2 + \mathsf{tv})$ where tv_1 and tv_2 are time for verifier's tasks related to committing each mul_j in $\{\mathsf{mul}_j\}_{j\in[T]}$ and folding, respectively, while tv is verifier time when verifying in $\mathsf{CF.Prove}$. When minimizing time by parallelism, verifier's time achieves $\mathcal{O}(\log T \cdot \mathsf{tv}_1 + \log N \cdot \mathsf{tv}_2 + \mathsf{tv})$.
Instantiations. We also discuss possible instantiations of our RAMenPaSTA in Sec. 7 including those from Σ-protocol theory [AC20], dual-mode NIWIs [GS08] and MPC-in-the-Head paradigm [IKOS07, IKOS09].

7 Instantiations

We provide potential instantiations of pvRAM presented in Cstr. 4 including instantiations from compressed Σ-protocol theory (Sec. 7.1), dual-mode NIWIs (Sec. 7.2) and MPC-in-the-Head paradigm (Sec. 7.3).

7.1 Instantiation From Compressed Σ-Protocol Theory

Recall that, in [AC20], they construct a succinct ZKAoK for circuit satisfiability in [AC20, Sec. 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 Pedersen commitment scheme [Ped92] (recalled in Appdx. A.6) as a building block. Moreover, 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 Appdx. A.12) for circuit satisfiability meets required properties in Thm. 2. Therefore, it is expected that RAMenPaSTA in Cstr. 4 can be instantiated by Pedersen commitment scheme to achieve a sub-linear[10] 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 $w \in \mathbb{N}$. 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

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

employ a grand commitment key $\mathsf{ck} = (g_1, \dots, g_{w'}) \in \mathbb{G}^{w'}$ for some $w' \geq w$ such that each entry of ck is sampled independently and uniformly from $\mathbb{G}$. Nevertheless, commitments in $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ may be computed from the same keys, e.g., $\mathsf{ck}'_1 = \mathsf{ck}_2$ as specified in Sec. 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 $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ into L split-relations of $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$, for some constant $L \in \mathbb{N}$, such that

- with respect to the same statement and witness, $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ is satisfied if and only if all split-relations are satisfied, and
- any two commitments involving constraints of a split-relation do not have the same commitment keys.

Then, we devise a sufficiently long commitment key $\mathsf{ck} \in \mathbb{G}^{w'}$, for some $w' \in \mathbb{N}$, 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, verifier can compute commitments with respect to those split-relations and grand commitment key ck by simply manipulating on 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 checking multiplication of two single field elements. Therefore, 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 $(a_1^{(i)}, b_1^{(i)}, \dots, a_\mu^{(i)}, b_\mu^{(i)}, a_{\mu+1}^{(i)})$ where $a_1^{(i)}, \dots, a_{\mu+1}^{(i)}$ are prover's messages while $b_1^{(i)}, \dots, b_\mu^{(i)} \in \mathbb{F}$ are verifier's challenges. Notice that, to simplify the proof of knowledge soundness and reduce the communication cost, we can enforce $b_i^{(1)} = \cdots = b_i^{(L)}$, 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 $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ in (32) as follows:

$$\begin{cases} [\![\mathbb{z}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}, [\![\mathsf{reg_{in}}]\!]_{\mathsf{cki}} \in \mathcal{R}_{\mathsf{com}}, \\ \overline{\mathsf{pc}} = 1 \wedge \overline{\mathsf{reg}} = \mathsf{reg_{in}} \wedge \mathsf{reg} = \mathsf{reg_{out}}, \\ [\![\mathsf{mul}_j]\!]_{\mathsf{ckm}_j} \in \mathcal{R}_{\mathsf{com}} & \forall j \in [T], \\ \mathsf{plkiv} = \sum_{j=1}^{T} \mathsf{mul}_j \cdot \left(\chi + \mathsf{plkcp}'_j\right)^{-1}, \\ \mathsf{miv} = \mathsf{miv}' \wedge \mathsf{macs}^\star = \mathsf{macs}' \end{cases}$$

where

- $\mathsf{pp} = (\mathsf{tck}, \mathsf{tck}') = ((\mathsf{ck}_1, \dots, \mathsf{ck}_5, \mathsf{cke}), (\mathsf{ck}'_1, \dots, \mathsf{ck}'_3, \mathsf{cke}'))$;
- $\mathbb{z} = (\mathsf{lt}, \mathsf{rt}, \mathbb{x}, \mathbf{i}, \mathbf{o}, \mathbb{x}^\star, \mathbf{s}, \mathbf{a})$;
- $[\![\mathbb{z}]\!]_{\mathsf{pp}} = (\mathsf{lt}, \mathsf{rt}, [\![\mathbb{x}]\!]_{\mathsf{tck}}, [\![\mathbf{i}]\!]_{\mathsf{ck}_1}, [\![\mathbf{o}]\!]_{\mathsf{ck}_2}, [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'}, [\![\mathbf{s}]\!]_{\mathsf{ck}_5}, [\![\mathbf{a}]\!]_{\mathsf{ck}_5})$ as in (12);
- $\overline{\mathsf{pc}}$, $\overline{\mathsf{reg}}$, reg, $\mathsf{macs}^\star$, macs' are obtained by parsing $\mathbf{i} = (\overline{\mathsf{pc}} \| \overline{\mathsf{reg}} \| \overline{\mathsf{macs}} \| \mathsf{macs}^\star)$ and $\mathbf{o} = (\mathsf{pc} \| \mathsf{reg} \| \mathsf{macs} \| \mathsf{macs}')$;
- $\mathsf{reg_{in}}$ and $\mathsf{reg_{out}}$ are input and output registers of RAM program;
- $\mathsf{gchal} = (\gamma, \delta, \chi, \psi, \tau, \omega)$ as described in equivalence (31);
- values $\mathsf{plkiv}, \mathsf{miv}$ and miv' are obtained by parsing vector $\mathbf{s} = (\mathsf{plkiv} \| \mathsf{miv} \| \mathsf{miv}')$, and plkcp'_j, for $j \in [T]$, is computed by

$$\mathsf{plkcp}'_j := \left\langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k=0}^{n_{\mathsf{plk}}} \right\rangle$$

  according to (21).

Moreover, from (16), $[\![\mathbb{z}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$ is equivalent to

$$\begin{cases} \mathsf{lt}, \mathsf{rt} \in \mathbb{N} \wedge \mathsf{lt} < \mathsf{rt}, \\ [\![\mathbb{z}]\!]_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}com}}, \\ [\![\mathbb{x}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}, \\ [\![\mathbb{x}^\star]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}} \end{cases}$$

and $[\![\mathbb{z}]\!]_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}com}}$ (see (15)) is equivalent to

$$\begin{cases} [\![\mathbb{z}]\!]_{\mathsf{pp}} \text{ of the form (12)}, \\ [\![\mathbf{i}]\!]_{\mathsf{ck}_1},\ [\![\mathbf{o}]\!]_{\mathsf{ck}_2} \in \mathcal{R}_{\mathsf{com}}, \\ [\![\mathbf{s}]\!]_{\mathsf{ck}_5},\ [\![\mathbf{a}]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}. \end{cases}$$

Recall, from Sec. 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:*

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

  where
  $$\mathbf{i} = (\overline{\mathsf{pc}} \| \overline{\mathsf{reg}} \| \overline{\mathsf{macs}} \| \mathsf{macs}^\star), \mathbf{o} = (\mathsf{pc} \| \mathsf{reg} \| \mathsf{macs} \| \mathsf{macs}'), \mathsf{s} = (\mathsf{plkiv} \| \mathsf{miv} \| \mathsf{miv}'),$$

  involving commitment keys cki, $\mathsf{ckm}_1, \ldots, \mathsf{ckm}_T$, ck_1, ck_2 and ck_5;
- *Constraint set 4:* $[\![\mathbf{a}]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}$ involving ck_5.

Thus, with $L = 4$, we can split $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ into 4 split-relations with respect to those above constraints.
Efficiency. Since Pedersen commitments have constant commitment size, i.e., $\mathcal{O}(1)$, the communication cost of protocol Π_{pvr} is $\mathcal{O}(N + T + \mathsf{p})$ where p is the communication cost $\Pi_{\mathsf{pvr\text{-}prf}}$. Notice that the communicate cost in Π_{pvr} before executing $\Pi_{\mathcal{CF}_{\mathsf{gnr}}}$ is $\mathcal{O}(N + T)$.

7.2 Instantiation From Dual-Mode NIWIs

Dual-mode NIWIs [GS08] allow to set up the system in one of the two modes, namely, hiding and binding modes. If the system is initiated with hiding mode, the resulting NIWI proof system satisfies statistical witness indistinguishability while, in the binding mode, the resulting proof satisfies statistical extractability.

In [GS08], this dual-mode NIWI proof system can be instantiated from bilinear pairings with respect to the bilinear map $F : \mathbb{G} \times \mathbb{G} \to \mathbb{G}^\star$ such that $\mathbb{G}$ and $\mathbb{G}^\star$ are $\mathbb{Z}_q$-modules and $\mathbb{G}$ has a generator $g \in \mathbb{G}$. The commitment scheme employed in [GS08] allows committing to each value $v \in \mathbb{Z}_q$ for some prime q to obtain a commitment c in $\mathbb{G}$. Moreover, this commitment scheme is additively homomorphic. The commitment c is either perfectly hiding or perfectly binding depending on the mode of setup of the commitment scheme, namely, hiding and binding modes, respectively. Moreover, both setups, namely, hiding and binding, are indistinguishable. However, extracting elements in $\mathbb{Z}_q$ is not possible in [GS08]. To overcome this issue, as suggested in [LNPY21, NGSY22], we can parse v into binary and commit to each bit of v, i.e., committing to either $1 \in \mathbb{G}$ or g where 1 here is understood to be the identity element of $\mathbb{G}$. The technique for parsing elements in $\mathbb{Z}_q$ into binary can be found in [LNSW13]. Specifically, by setting $k = \lfloor \log_2 v \rfloor + 1$, this technique specifies a basis $\{B_i\}_{i \in [k]} \subset \mathbb{Z}$ such that

$$\{\langle \mathbf{b}, \mathbf{x} \rangle : \mathbf{b} = (B_1, \ldots, B_k) \wedge \mathbf{x} \in \{0,1\}^k\} = [0, q-1].$$

In other words, the set of linear combinations of $B_1, \ldots, B_k$ by vectors in $\{0,1\}^k$ spans the set $[0, q-1]$.

With the above commitment scheme, the dual-mode NIWIs in [GS08] allow to prove linear and pairing relations between elements of $\mathbb{G}$ which implies the relations between elements in $\mathbb{Z}_q$ that are maps to those group elements in $\mathbb{G}$. Witness indistinguishability is guaranteed by perfect hiding and knowledge soundness is guaranteed perfect binding of the dual-mode commitment scheme discussed above.

Efficiency. Since commitment size is linear in message size and each element in $\mathbb{Z}_q$ is split into $\mathcal{O}(\log q)$ elements, the communication cost of $\mathit{\Pi}_{\mathsf{pvr}}$ when instantiated by dual-mode NIWIs in [GS08] for bilinear pairings is $\mathcal{O}(Nw + T) \cdot \log |\mathbb{F}|$ where w is the witness size of execution of a single instruction.

7.3 Instantiation From MPC-in-the-Head Paradigm

MPC-in-the-Head paradigm [IKOS07, IKOS09] are constructed by having a prover simulating the execution of an MPC protocol "in his head" for computing some function and transforming this execution into zero-knowledge proofs for satisfaction of such a function. A line of works [GMO16, CDG$^+$17, KKW18, BN20, DOT21] constructed post-quantum zero-knowledge proofs for circuit satisfiability with linear proof size, linear prover and verifier time. However, commitment scheme employed in these works can be instantiated by collision-resistant hash functions, in practice, and are not required to be additively homomorphic.

To be able to instantiate by the above zero-knowledge proofs, as required from Thm. 2, we employ additively homomorphic commitment schemes instead. The strategy is similar to protocol $\mathit{\Pi}_{\mathsf{pvr}}$ specified in Cstr. 4 except the last proof, i.e., $\mathit{\Pi}_{\mathsf{pvr\text{-}prf}}$. Here, prover has commitments to necessary witnesses, including randomness, for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$. Since commitment scheme employed is homomorphic and prover knows all randomness, prover can split the witnesses, including randomness, behind the commitments into commitments to shares that sum up to those witnesses. Specifically, for a commitment $\tilde{\mathsf{m}}$ to $\mathbf{m}$ with respect to commitment key ckm and randomness $\hat{\mathsf{m}}$, split $(\mathbf{m}, \hat{\mathsf{m}})$ into k shares, i.e., $(\mathbf{m}_1, \hat{\mathsf{m}}_1), \ldots, (\mathbf{m}_k, \hat{\mathsf{m}}_k)$, such that $\sum_{i=1}^{k} \mathbf{m}_i = \mathbf{m}$ and $\sum_{i=1}^{k} \hat{\mathsf{m}}_i = \hat{\mathsf{m}}$. It can be seen that $\sum_{i=1}^{k} \mathsf{C.Commit}_{\mathsf{ckm}}(\mathbf{m}_i, \hat{\mathsf{m}}_i) = \tilde{\mathsf{m}}$. Here, prover can simulate the MPC-in-the-head paradigm as specified in those works, namely, [GMO16, CDG$^+$17, KKW18, BN20, DOT21]. Hence, the i-th share also belongs to the i-th view of the MPC "in the head" of prover. When being required by verifier to open the i-th view, prover opening all witnesses in the i-th view including witnesses and randomness of the i-th share. Verifier also needs to check witnesses of the i-th view are consistent with the i-th share.

Efficiency. Let $\mathsf{c}(k)$ be the size of commitments to messages in $\mathbb{F}^k$ and p be the communication cost of $\mathit{\Pi}_{\mathsf{pvr\text{-}prf}}$ instantiated by a ZKAoK from MPC-in-the-head paradigm. The total communication cost of $\mathit{\Pi}_{\mathsf{pvr}}$ is

$$\mathcal{O}\left(N \cdot (\mathsf{c}(m_1) + \cdots + \mathsf{c}(m_5) + \mathsf{c}(n) + \mathsf{c}(m_3') + \mathsf{c}(n')) + T \cdot \mathsf{c}(1) + \mathsf{p}\right).$$

Acknowledgements

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). The second author, Minh Pham, also thanks Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for supporting this work during his study.

References

AC20. Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 513–543. Springer, Heidelberg, Germany, 2020.

ACK21. Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 549–579. Springer, Heidelberg, Germany, 2021.

AFR23. Thomas Attema, Serge Fehr, and Nicolas Resch. Generalized special-sound interactive proofs and their knowledge soundness. In *TCC 2023: 21st Theory of Cryptography Conference, Part III*, pages 424–454. Springer, Heidelberg, Germany, 2023.

BBBF18. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, Heidelberg, Germany, 2018.

BBC$^+$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 *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 551–579. Springer, Heidelberg, Germany, 2017.

BC23. Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. In *Advances in Cryptology – ASIACRYPT 2023, Part II*, volume 14439 of Lecture Notes in Computer Science, pages 77–110. Springer Nature Singapore, 2023.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *45th Annual ACM Symposium on Theory of Computing*, pages 111–120. ACM Press, 2013.

BCG$^+$13. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, Heidelberg, Germany, 2013.

BCI$^+$13. Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 315–333. Springer, Heidelberg, Germany, 2013.

BCL$^+$21. Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710. Springer, Heidelberg, Germany, 2021.

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, Germany, 2020.

BCR$^+$19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, Heidelberg, Germany, 2019.

BCTV14. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security 2014: 23rd USENIX Security Symposium*, pages 781–796. USENIX Association, 2014.

BDD20. Carsten Baum, Bernardo David, and Rafael Dowsley. Insured MPC: Efficient secure computation with financial penalties. In *FC 2020: 24th International Conference on Financial Cryptography and Data Security*, volume 12059 of *Lecture Notes in Computer Science*, pages 404–420. Springer, Heidelberg, Germany, 2020.

BFR$^+$13. Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Symposium on Operating Systems Principles - SOSP 2013*, pages 341–357. ACM, 2013.

BGH19. Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. `https://eprint.iacr.org/2019/1021`.

BMM$^+$21. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 65–97. Springer, Heidelberg, Germany, 2021.

BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 495–526. Springer, Heidelberg, Germany, 2020.

CCDW20. Weikeng Chen, Alessandro Chiesa, Emma Dauterman, and Nicholas P. Ward. Reducing participation costs via incremental verification for ledger systems. Cryptology ePrint Archive, Report 2020/1522, 2020. `https://eprint.iacr.org/2020/1522`.

CCG$^+$23. Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. Proof-carrying data from arithmetized random oracles. In *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 379–404. Springer, Heidelberg, Germany, 2023.

CCS22. Megan Chen, Alessandro Chiesa, and Nicholas Spooner. On succinct non-interactive arguments in relativized worlds. In *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 336–366. Springer, Heidelberg, Germany, 2022.

CDG$^+$17. Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1825–1842. ACM Press, 2017.

CDP12. Ronald Cramer, Ivan Damgård, and Valerio Pastro. On the amortized complexity of zero knowledge protocols for multiplicative relations. In *ICITS 12: 6th International Conference on Information Theoretic Security*, volume 7412 of *Lecture Notes in Computer Science*, pages 62–79. Springer, Heidelberg, Germany, 2012.

CGG$^+$23. Arka Rai Choudhuri, Sanjam Garg, Aarushi Goel, Sruthi Sekar, and Rohit Sinha. Sublonk: Sublinear prover plonk. Cryptology ePrint Archive, Paper 2023/902, 2023. `https://eprint.iacr.org/2023/902`.

CGSY23. Alessandro Chiesa, Ziyi Guan, Shahar Samocha, and Eylon Yogev. Security bounds for proof-carrying data from straightline extractors. Cryptology ePrint Archive, Paper 2023/1646, 2023. `https://eprint.iacr.org/2023/1646`.

CHM$^+$20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768. Springer, Heidelberg, Germany, 2020.

COS20. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 769–793. Springer, Heidelberg, Germany, 2020.

CT10. Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS 2010: 1st Innovations in Computer Science*, pages 310–331. Tsinghua University Press, 2010.

DOT21. Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge MPCitH-based arguments. In *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 3022–3036. ACM Press, 2021.

dSGOTV22. 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 *SCN 22: 12th International Conference on Security in Communication Networks*, volume 13409 of *Lecture Notes in Computer Science*, pages 615–638. Springer, Heidelberg, Germany, 2022.

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`.

FKL$^+$21. Nicholas Franzese, Jonathan Katz, Steve Lu, Rafail Ostrovsky, Xiao Wang, and Chenkai Weng. Constant-overhead zero-knowledge for RAM programs. In *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 178–191. ACM Press, 2021.

FMNV22. Daniele Friolo, Fabio Massacci, Chan Nam Ngo, and Daniele Venturi. Cryptographic and financial fairness. *IEEE Trans. Inf. Forensics Secur.*, 17:3391–3406, 2022.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, Heidelberg, Germany, 2013.

GKR$^+$21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX*

Security 2021: 30th USENIX Security Symposium, pages 519–535. USENIX Association, 2021.

GMO16. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In *USENIX Security 2016: 25th USENIX Security Symposium*, pages 1069–1083. USENIX Association, 2016.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, Heidelberg, Germany, 2008.

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`.

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

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *39th Annual ACM Symposium on Theory of Computing*, pages 21–30. ACM Press, 2007.

IKOS09. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.

IOS23. Yuval Ishai, Rafail Ostrovsky, and Akash Shah. Succinct arguments for RAM programs via projection codes. In *Advances in Cryptology - CRYPTO 2023*, volume 14082 of Lecture Notes in Computer Science, pages 159–192. Springer, 2023.

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 525–537. ACM Press, 2018.

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`.

KS23. 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 *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388. Springer, Heidelberg, Germany, 2022.

Kus19. John Kuszmaul. Verkle trees. *Verkle Trees*, 1:1, 2019.

KVV16. Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 406–417. ACM Press, 2016.

LFKN90. Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st Annual Symposium on Foundations of Computer Science*, pages 2–10. IEEE Computer Society Press, 1990.

LGZX23. Xun Liu, Shang Gao, Tianyu Zheng, and Bin Xiao. Snarkfold: Efficient SNARK proof aggregation from split incrementally verifiable computation. *IACR Cryptol. ePrint Arch.*, page 1946, 2023.

LNPY21. Benoît Libert, Khoa Nguyen, Thomas Peters, and Moti Yung. Bifurcated signatures: Folding the accountability vs. anonymity dilemma into a single private signing scheme. In *Advances in Cryptology – EUROCRYPT 2021, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 521–552. Springer, Heidelberg, Germany, 2021.

LNSW13. San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 107–124. Springer, Heidelberg, Germany, 2013.

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*, 2024. to appear.

NGSY22. Khoa Nguyen, Fuchun Guo, Willy Susilo, and Guomin Yang. Multimodal private signatures. In *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 792–822. Springer, Heidelberg, Germany, 2022.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, Heidelberg, Germany, 1992.

Sch80. Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, Heidelberg, Germany, 2020.

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, Germany, 2008.

W$^+$14. Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger, 2014. Ethereum project yellow paper.

WLP$^+$14. Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic, Volume I User level Isa, Andrew Waterman, Yunsup Lee, and David Patterson. The risc-v instruction set manual. *Volume I: User-Level ISA', version*, 2, 2014.

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.

YH24. Yibin Yang and David Heath. Two shuffles make a RAM: improved constant overhead zero knowledge RAM. In *USENIX Security 2024: 33rd USENIX Security Symposium*. USENIX Association, 2024. to appear.

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`.

Zip79. Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

ZSZG23. Yuncong Zhang, Shi-Feng Sun, Ren Zhang, and Dawu Gu. Polynomial IOPs for Memory Consistency Checks in Zero-Knowledge Virtual Machines. In *Advances in Cryptology – ASIACRYPT 2023, Part II*, pages 111–141. Springer Nature Singapore, 2023.

ZV23. Yan X Zhang and Aard Vark. Parallelizing Nova - Visualizations and Mental Models behind Paranova. zkResearch, 2023. `https://zkresear.ch/t/parallelizing-nova-visualizations-and-mental-models-behind-paranova/198`, accessed at 1 Feb 2024, 8:00PM SGT.

A Preliminaries (Extended)

We recall the necessary preliminaries in complement to Sec. 2.

A.1 RAM Program

We model 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 register reg, a program counter pc, an initial value val 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 $\mathbb{F} \cup \{\bot\}$ where $\bot$ is understood to be an uninitialized value which cannot be read by the instructions. At the beginning, every mem_i is set to be $\bot$.
- The register reg contains a constant number of elements in $\mathbb{F}$, i.e., $\mathsf{reg} \in \mathbb{F}^{c_{\mathsf{reg}}}$ for some constant $c_{\mathsf{reg}} \in \mathbb{Z}_+$. This register can be viewed as a small temporary memory for the instructions to receive all of its elements and to perform the computations.
- 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. Specifically, instruction F_{pc} receives as input the tuple $(\mathsf{reg}, \mathsf{val})$ including the register reg and the value $\mathsf{val} \in \mathbb{F}$. It then returns a new tuple $(\mathsf{pc}', \mathsf{reg}', \ell, \mathsf{val}', \mathsf{mop})$ containing a new program counter pc', new register reg', 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 $\mathsf{reg} := \mathsf{reg}'$ and $\mathsf{pc} := \mathsf{pc}'$.
  - If $\mathsf{mop} = \text{WRITE}$, set $\mathsf{mem}_\ell := \mathsf{val}'$ and $\mathsf{val} := \mathsf{val}'$. Otherwise, if $\mathsf{mop} = \text{READ}$, set $\mathsf{val} := \mathsf{mem}_\ell$.

A.2 Memory Consistency Check

We recall the technique for checking memory consistency in [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, during the course of 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] shows that there exists a sequence $(\mathsf{macs}'_i)_{i\in[N]}$ such that (i) $(\mathsf{macs}'_i)_{i\in[N]}$ is a permutation of $(\mathsf{macs}_i)_{i\in[N]}$ and (ii) $(\mathsf{macs}'_i)_{i\in[N]}$ is sorted lexicographically. Mathematically speaking, it suffices 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

$$\begin{cases} 1 \le \ell_i \le M \wedge \mathsf{mop}_i \in \{0,1\} & \forall i \in [N], \\ (\mathsf{macs}'_i)_{i\in[N]} \text{ is a permutation of } (\mathsf{macs}_i)_{i\in[N]} \end{cases} \tag{33}$$

and

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

Note that, the system (34) contains all the sufficient conditions to check whether the sequence $(\mathsf{macs}'_i)_{i\in[N]}$ is sorted lexicographically. It also check that $\mathsf{time}_{i-1} < \mathsf{time}_i$ for all $i \in [2, N]$ to ensure that the the original memory access sequence is computed chronically.

A.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 $F : \mathbb{F}^{n_{\mathsf{in}}} \rightarrow \mathbb{F}^{n_{\mathsf{out}}}$ that maps an input vector $\mathbf{x} = (x_1, \dots, x_{n_{\mathsf{in}}}) \in \mathbb{F}^{n_{\mathsf{in}}}$ to an output vector $\mathbf{y} = (y_1, \dots, y_{n_{\mathsf{out}}}) \in \mathbb{F}^{n_{\mathsf{out}}}$. 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} to be 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 structure of circuit C. Moreover, the i-th gate is associated with the selectors $s_i^{\mathsf{left}}, s_i^{\mathsf{right}}, s_i^{\mathsf{mul}}, s_i^{\mathsf{const}}$ such that the relation between a_i, b_i and c_i is captured by the equation

$$s_i^{\mathsf{left}} \cdot a_i + s_i^{\mathsf{right}} \cdot b_i + s_i^{\mathsf{mul}} \cdot (a_i \cdot b_i) + s_i^{\mathsf{const}} - c_i = 0. \tag{35}$$

Each equation in the form of (35) is a *gate constraint.* Hence, we define a witness satisfying circuit C to be

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

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 circuit 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 $\varphi : [n_{\mathsf{wit}}] \rightarrow [n_{\mathsf{wit}}]$ based on C such that the copy constraint is satisfied if and only if $((1, w_1), \dots, (n_{\mathsf{wit}}, w_{n_{\mathsf{wit}}}))$ is a permutation of $((\varphi(1), w_1), \dots, (\varphi(n_{\mathsf{wit}}), w_{n_{\mathsf{wit}}}))$. According to [GWC19], for value $\gamma, \delta \xleftarrow{\$} \mathbb{F}$, if

$$\prod_{i=1}^{n_{\mathsf{wit}}} (\gamma + i \cdot \delta + w_i) = \prod_{i=1}^{n_{\mathsf{wit}}} (\gamma + \varphi(i) \cdot \delta + w_i) \tag{36}$$

holds, then it would imply that $((1, w_1), \dots, (n_{\mathsf{wit}}, w_{n_{\mathsf{wit}}}))$ is a permutation of $((\varphi(1), w_1), \dots, (\varphi(n_{\mathsf{wit}}), w_{n_{\mathsf{wit}}}))$ 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 PLONK structure

$$\begin{aligned} \mathsf{plkst} = (s_1^{\mathsf{left}}, s_1^{\mathsf{right}}, s_1^{\mathsf{mul}}, s_1^{\mathsf{const}}, \dots, s_{n_{\mathsf{gate}}}^{\mathsf{left}}, s_{n_{\mathsf{gate}}}^{\mathsf{right}}, s_{n_{\mathsf{gate}}}^{\mathsf{mul}}, s_{n_{\mathsf{gate}}}^{\mathsf{const}}, \\ \varphi(1), \dots, \varphi(n_{\mathsf{wit}})) \in \mathbb{F}^{n_{\mathsf{plk}}} \end{aligned} \tag{37}$$

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

From (35) and , by sampling $\gamma, \delta \xleftarrow{\$} \mathbb{F}$, if the system

$$\begin{cases} s_i^{\mathsf{left}} \cdot a_i + s_i^{\mathsf{right}} \cdot b_i + s_i^{\mathsf{mul}} \cdot (a_i \cdot b_i) + s_i^{\mathsf{const}} - c_i = 0 \quad \forall i \in [n_{\mathsf{gate}}], \\ \prod_{i=1}^{n_{\mathsf{wit}}} (\gamma + i \cdot \delta + w_i) = \prod_{i=1}^{n_{\mathsf{wit}}} (\gamma + \varphi(i) \cdot \delta + w_i) \end{cases} \tag{38}$$

is satisfied, we see that $\mathbf{w}_{\mathsf{plk}}$ is a valid witness of C with respect to the compact PLONK structure plkst with probability at least $1 - \frac{n_{\mathsf{wit}}}{|\mathbb{F}|}$. Notice that (38) can be represented under the form of an R1CS constraint system with public matrices determined based 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.

A.4 Logarithmic Derivative Supporting Permutation and Lookup Arguments

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

Lemma 2 (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 if and only if*

$$\sum_{i=1}^{n} \frac{1}{X + a_i} = \sum_{i=1}^{n} \frac{1}{X + b_i} \tag{39}$$

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 if and only if 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 Lm. 3:

Lemma 3 (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 $\mathbf{b}$ if and only if*

$$\sum_{i=1}^{n} \frac{1}{X + \left\langle \mathbf{a}_i, (Y^k)_{k=0}^{s-1} \right\rangle} = \sum_{i=1}^{n} \frac{1}{X + \left\langle \mathbf{b}_i, (Y^k)_{k=0}^{s-1} \right\rangle} \tag{40}$$

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 (40) 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 $u \le n$ 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 $v \le n$ 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:

$$\begin{cases} u = v, \\ \exists \sigma \in \mathsf{S}_u \text{ s.t. } \mathbf{a}'_i = \mathbf{b}'_{\sigma(i)} \wedge \mathsf{mul}_i = \mathsf{mul}'_{\sigma(i)} \ \forall\ i \in [u] \end{cases} \tag{41}$$

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 (40) holds if and only if $f(X,Y) = 0 \in \mathbb{F}(X,Y)$. Since we are assuming that (40) 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 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 $\mathbf{a}'_1, \dots, \mathbf{a}'_u$ 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 $\sigma \in \mathsf{S}_u$ such that $\mathbf{a}'_k = \mathbf{b}'_{\sigma(k)}$ 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

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

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 (41). □

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 (40) holds with probability at most

$$\mathsf{err}_{\mathsf{perm}}(\mathbb{F}, s, n) = \frac{(s-1)(2n-1)}{|\mathbb{F}|}. \tag{42}$$

according the following Lm. 4.

Lemma 4 (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* $f_{\mathbf{v}}(Y) = \langle \mathbf{v}, (Y^k)_{k=0}^{s-1} \rangle$. *Assume that*

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

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 $f(X,Y)\neq 0$ if and only if $g(X,Y)\neq 0$.

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

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

as desired. □

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

Lemma 5 (Lm. 5 of [Hab22] and Lm. 4 of [BC23]). *Let n and t be a 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, $\{a_i\}_{i=1}^n\subseteq\{b_i\}_{i=1}^t$ if and only if there exists $(\mathsf{mul}_j)_{j=1}^t$ over $\mathbb{F}$ satisfying*

$$\sum_{i=1}^{n}\frac{1}{X+a_i}=\sum_{j=1}^{t}\frac{\mathsf{mul}_i}{X+b_j} \tag{43}$$

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

Remark 7. In Lm. 5, 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 Lm. 5 is reduced into proving that a certain polynomial $p(X)$ of degree $d=n+t-1$ is equal to zero. By Schwartz-Zippel lemma, 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 Lm. 2.

Tuple Lookup. We may encounter 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, Lm. 5, to achieve the following Lm. 6:

Lemma 6. *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, $\{\mathbf{a}_i\}_{i=1}^n\subseteq\{\mathbf{b}_j\}_{j=1}^t$ if and only if there exists $\mathsf{mul}_1,\ldots,\mathsf{mul}_t\in\mathbb{F}$ satisfying*

$$\sum_{i=1}^{n}\frac{1}{X+\left\langle\mathbf{a}_i,(Y^k)_{k=0}^{s-1}\right\rangle}=\sum_{j=1}^{t}\frac{\mathsf{mul}_j}{X+\left\langle\mathbf{b}_i,(Y^k)_{k=0}^{s-1}\right\rangle} \tag{44}$$

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 (44) trivially holds.

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

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

- (44) 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 $\mathbf{a}'_1,\dots,\mathbf{a}'_m$ 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 Lm. 6, i.e., exists $\mathbf{a}_i \notin \{\mathbf{b}_j\}_{j=1}^t$, by sampling $\psi \overset{\$}{\leftarrow} \mathbb{F}$ and $\chi \overset{\$}{\leftarrow} \mathbb{F} \setminus \{\langle \mathbf{b}_i, (\psi^k)_{k=0}^{s-1} \rangle\}_{i=1}^t$, and setting $X = \chi$ and $Y = \psi$, we can see that (44) holds with probability at most

$$\mathsf{err}_{\mathsf{lookup}}(\mathbb{F}, s, n, t) = \frac{(s-1)(n+t-1)}{|\mathbb{F}| - t} \tag{45}$$

according to the following Lm. 7

Lemma 7 (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,*

$$\Pr\left[f(\chi, \psi) = 0 \middle| \psi \overset{\$}{\leftarrow} \mathbb{F} \wedge \chi \overset{\$}{\leftarrow} \mathbb{F} \setminus \{-f_{\mathbf{b}_i}(\psi)\}_{i=1}^t\right] \leq \frac{(s-1)(n+t-1)}{|\mathbb{F}| - t}$$

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 $f(X,Y) \neq 0$ if and only if $g(X,Y) \neq 0$.

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 $j \in [n]$. However, in such cases, $g(\chi, \psi)$ is still computable since there is no denominator in $g(X,Y) \in \mathbb{F}(X,Y)$. Now, let us denote $D_\psi = \mathbb{F} \setminus \{-f_{\mathbf{b}_i}(\psi)\}_{i=1}^t$. We see that, for any $(\chi, \psi) \in \mathbb{F} \times D_\psi$, if $f(\chi, \psi)$ is computable and $f(\chi, \psi) = 0$, then $g(\chi, \psi) = 0$. We deduce that

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

as desired. □

A.5 Schwartz-Zippel Lemma

We recall the Schwartz-Zippel lemma [Zip79, Sch80] in the following Lm. 8.

Lemma 8 (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*

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

A.6 Commitment Scheme (Extended)

We formally describe the security properties of commitment scheme defined in Section 2.1, namely *binding* and *hiding* in Def. 7, 8 and 9, respectively.
Security of Commitment Schemes. Let $\mathcal{C}$ be a commitment scheme with syntax in Def. 1. We now define the completeness, binding and hiding of $\mathcal{C}$ in the following Def. 7, 8 and 9, respectively.

Definition 7 (Perfect Correctness of $\mathcal{C}$). *$\mathcal{C}$ satisfies correctness if for all message M, randomness R, it holds that*

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

Definition 8 (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, in case that $\mathcal{A}$ is computationally unbounded, we say that $\mathcal{C}$ is statistically binding.

Definition 9 (Hiding of $\mathcal{C}$). *A commitment scheme satisfies hiding if for any messages M and M', then the two following distributions are close:*

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

If the above two distribution 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 Def. 1, 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 8. By using notation in Rmk. 1, 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 $\mathbb{F}$. Here, due to 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], which is a secure and homomorphic commitment for committing to length-n vectors. As a remark, just for this instantiation, the addition "+" is 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.

```
$\mathsf{pp} \leftarrow \mathsf{FS.Setup}(1^\lambda)$.
$(Z_0, Z_1, \rho) \leftarrow \mathcal{A}(\mathsf{pp})$.
$(Z, W) \leftarrow \mathit{\Pi}_{\mathcal{FS}}(\mathsf{HS}, \mathsf{pp}, (Z_0, Z_1;\ W_0, W_1))$
$(W_0, W_1) \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{pp}, Z_0, Z_1, \rho)$.
$b_0 = ((\mathsf{pp}, Z, W) \in \mathcal{R})$
$b_1 = ((\mathsf{pp}, Z_0, W_0) \notin \mathcal{R}) \vee ((\mathsf{pp}, Z_1, W_1) \notin \mathcal{R})$
Return $b_0 \wedge b_1$.
```

Fig. 3. Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{fs\text{-}knwlg\text{-}sound}}(\lambda)$.

A.7 Folding Scheme

We recall the definition and security properties of folding scheme [KST22] in Def. 11, 12 and 13 respectively.

Definition 10 (Syntax of Folding Scheme). *Let $\mathcal{P}$, $\mathcal{Z}$ and $\mathcal{W}$ be the sets of public parameters, instances and witnesses, respectively. Let $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{Z} \times \mathcal{W}$ be a 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.*

$\mathsf{FS.Fold}(\mathsf{pp}, Z_0, Z_1;\ W_0, W_1) \to (Z;\ W)$ *: This algorithm is run by prover. On inputs public parameter pp and instance-witness pairs $(Z_0, W_0), (Z_1, W_1) \in \mathcal{Z} \times \mathcal{W}$, it returns an instance-witness pair $(Z, W) \in \mathcal{Z} \times \mathcal{W}$.*

$\mathsf{FS.Verify}(\mathsf{pp}, Z_0, Z_1) \to Z$**:** *This algorithm is run by verifier. On inputs public parameter pp and instances $Z_0, Z_1 \in \mathcal{Z}$, it returns an instance $Z \in \mathcal{Z}$.*

Defining Transcript. For common public inputs pp, Z_0, Z_1, we denote

$$(Z, W) \leftarrow \mathit{\Pi}_{\mathcal{FS}}(\mathsf{pp}, (Z_0, Z_1;\ W_0, W_1))$$

the output of prover and verifier when prover executes $\mathsf{FS.Fold}$ with inputs $\mathsf{pp}, Z_0, Z_1, W_0, W_1$ while verifier executes $\mathsf{FS.Verify}$ with inputs pp, Z_0, Z_1. Define the public transcript

$$\mathsf{tr} \leftarrow \mathsf{View}(\mathit{\Pi}_{\mathcal{FS}}(\mathsf{pp}, (Z_0, Z_1;\ W_0, W_1))$$

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

Definition 11 (Perfect Correctness of $\mathcal{FS}$). *Let $\mathsf{pp} \leftarrow \mathsf{FS.Setup}(1^\lambda)$. Then for any $\{(\mathsf{pp}, Z_i;\ W_i)\}_{i \in \{0,1\}} \subseteq \mathcal{R}$, it holds that*

$$\Pr\left[\begin{matrix} Z = Z' \wedge \\ (\mathsf{pp}, Z;\ W) \in \mathcal{R} \end{matrix} \middle| \begin{matrix} (Z;\ W) \leftarrow \mathsf{FS.Fold}(\mathsf{pp}, Z_0, Z_1;\ W_0, W_1, W') \\ Z' \leftarrow \mathsf{FS.Verify}(\mathsf{pp}, Z_0, Z_1) \end{matrix}\right] = 1.$$

Definition 12 (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 $\mathcal{E}$, with rewindable oracle access to $\mathcal{A}$, it holds that $\Pr[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{fs\text{-}knwlg\text{-}sound}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ where $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{fs\text{-}knwlg\text{-}sound}}(\lambda)$ is described in Fig. 3. A folding scheme $\mathcal{FS}$ is statistically (resp., computationally) knowledge-sound if $\mathcal{A}$ is computationally unbounded (resp., bounded). $\mathcal{FS}$ has soundness error ϵ if $\mathcal{A}$ can break knowledge soundness with probability at most ϵ.*

Definition 13 (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*

$$\begin{aligned} \big|\Pr\big[\mathcal{A}(\mathsf{tr}) = 1 \big| \mathsf{pp} \leftarrow \mathsf{FS.Setup}(1^\lambda), \mathsf{tr} \leftarrow \mathsf{View}(\mathit{\Pi}_{\mathcal{FS}}(, \mathsf{pp}, Z_0, Z_1;\ W_0, W_1))\big] \\ - \Pr\big[\mathcal{A}(\mathsf{tr}) = 1 \big| \mathsf{pp} \leftarrow \mathsf{FS.Setup}(1^\lambda), \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{pp}, Z_0, Z_1)\big]\big| \leq \mathsf{negl}(\lambda) \end{aligned}$$

A.8 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 Def. 14, then we recall the generalization for multi-round protocols in Def. 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 tree of transcripts [ACK21] in Def. 15. Then we state the special soundness property for multi-round protocols in Def. 16.

Definition 15 (Tree of Transcripts). *Let Π be a public-coin $(2\mu+1)$-move protocol. A $(k_1, \dots, k_\mu)$-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 ($(k_1, \dots, k_\mu; n_1, \dots, n_\mu)$-Special Soundness). *Let $k_1, \dots, k_\mu$, and $n_1, \dots, n_u$ be positive integers in $\mathbb{Z}_+$. Let $\mathrm{CH}_1, \mathrm{CH}_2, \dots, \mathrm{CH}_\mu$ 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 $(k_1, \dots, k_\mu; n_1, \dots, n_\mu)$-special soundness if there exists a PPT algorithm $\mathcal{E}$, such that, given an instance $\mathbb{z}$ and a $(k_1, \dots, k_\mu)$-tree of accepted transcripts, outputs a witness W such that $(Z, W) \in \mathcal{R}$.*

A.9 Interactive Folding Protocol for Folding Relaxed R1CS Instance-Witness Pairs (Extended)

Explanation of Equations in Sec. 2.2. We now explain in details the equations in Sec. 2.2. First, Eq. (4) 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 (2.2), 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 Lm. 1. Here, we provide a proof of Lm. 1. We recall Lm. 1 for readability purpose as following Lm. 9.

Lemma 9 (Recall of Lm. 1). *Let $\mathcal{C}$ be a secure and homomorphic commitment scheme. Assume that Π_{rr1cs} in Cstr. 1 are rewinded thrice, with the same $\tilde{\mathsf{g}}$ (commitment in $[\![\mathbf{g}]\!]_{\mathsf{cke}}$) and distinct challenges $\{\alpha^{(i)}\}_{i\in[3]}$ to produce $\{\tilde{\mathbb{x}}^{(i)}\}_{i\in[3]}$, respectively. If we have have corresponding witnesses to recover $\{[\![\mathbb{x}^{(i)}]\!]_{\mathsf{tck}}\}_{i\in[3]}$ satisfying $[\![\mathbb{x}^{(i)}]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ $\forall i \in [3]$, then we can extract witnesses to construct $\{[\![\mathbb{x}_i]\!]_{\mathsf{tck}}\}_{i\in\{0,1\}}$ s.t. $[\![\mathbb{x}_i]\!]_{\mathsf{tck}} \in \mathcal{R}^{\mathfrak{S}}_{\mathsf{rr1cs}}$ $\forall i \in \{0,1\}$.*

As a remark, this lemma satisfies $(3; |\mathbb{F}|)$-special soundness defined in Appdx. A.8.

Before going to the proof, we first recall the notations. Recall in Sec. 2.2, for a vector $\mathbb{x}$, we can parse

$$\mathbb{x} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, \mathbb{x}.\mathbf{z}_1, \dots, \mathbb{x}.\mathbf{z}_c, \mathbb{x}.\mathbf{e}) \tag{46}$$

where $\mathbb{x}.u$ and $\mathbb{x}.\mathsf{pub}$ are scalars and $\mathbb{x}.\mathbf{z}_1, \dots, \mathbb{x}.\mathbf{z}_c, \mathbb{x}.\mathbf{e}$ are vectors. Also, for a tuple commitment key $\mathsf{tck} = (\mathsf{ck}_1, \dots, \mathsf{ck}_c, \mathsf{cke})$, from (2), we can parse $[\![\mathbb{x}]\!]_{\mathsf{tck}}$ as

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

According to Rmk. 1, we additionally parse

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

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

$$\tilde{\mathbb{x}} = (\mathbb{x}.u, \mathbb{x}.\mathsf{pub}, \mathbb{x}.\tilde{\mathsf{z}}_1, \dots, \mathbb{x}.\tilde{\mathsf{z}}_c, \mathbb{x}.\tilde{\mathsf{e}}) \tag{47}$$

to contains all public information and commitments,

$$\hat{\mathbb{x}} = (\mathbb{x}.\hat{\mathsf{z}}_1, \dots, \mathbb{x}.\hat{\mathsf{z}}_c, \mathbb{x}.\hat{\mathsf{e}}) \tag{48}$$

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 (3), $\mathbb{x}$ is a valid pair of rR1CS if it satisfies the relation

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

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

Proof (Proof of Lm. 1 (recalled in Lm. 9)). First, consider the system

$$\begin{cases} \mathbb{x}^{(i)}.\mathsf{pub} = \mathbb{x}_0.\mathsf{pub} + \alpha^{(i)} \cdot \mathbb{x}_1.\mathsf{pub} & \forall i \in [2], \\ \mathbb{x}^{(i)}.\mathbf{z}_j = \mathbb{x}_0.\mathbf{z}_j + \alpha^{(i)} \cdot \mathbb{x}_1.\mathbf{z}_j & \forall i \in [2], \forall j \in [c], \\ \mathbb{x}^{(i)}.\mathbf{e} = \mathbb{x}_0.\mathbf{e} + \alpha^{(i)} \cdot \mathbf{g} + (\alpha^{(i)})^2 \cdot \mathbb{x}_1.\mathbf{e} & \forall i \in [3], \\ \mathbb{x}^{(i)}.\hat{\mathsf{z}}_j = \mathbb{x}_0.\hat{\mathsf{z}}_j + \alpha^{(i)} \cdot \mathbb{x}_1.\hat{\mathsf{z}}_j & \forall i \in [2], \forall j \in [c], \\ \mathbb{x}^{(i)}.\hat{\mathsf{e}} = \mathbb{x}_0.\hat{\mathsf{e}} + \alpha^{(i)} \cdot \hat{\mathsf{g}} + (\alpha^{(i)})^2 \cdot \mathbb{x}_1.\hat{\mathsf{e}} & \forall i \in [3]. \end{cases} \tag{49}$$

As each $\alpha^{(i)}$, for $i \in [3]$, has powers at most 3 in system (49) and we have 3 distinct challenges $\{\alpha^{(i)}\}_{i\in[3]}$, hence, we can solve system (49). By solving (49), we can extract $\mathbb{x}_i.\mathsf{pub}$, $\mathbb{x}_i.\mathbf{z}_j$ and $\mathbb{x}_i.\hat{\mathsf{z}}_j$ for all $i \in [2]$ and all $j \in [c]$. Hence, we can extract $\mathbb{x}_0$ and $\mathbb{x}_1$ of the form (46), $\hat{\mathbb{x}}_0$ and $\hat{\mathbb{x}}_1$ of the form (48), vector $\mathbf{g}$ and randomness $\hat{\mathsf{g}}$ 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 $[\![\mathbb{x}_i.\mathbf{z}_j]\!]_{\mathsf{ck}_j}, [\![\mathbb{x}_i.\mathbf{e}]\!]_{\mathsf{cke}} \in \mathcal{R}_{\mathsf{com}}$ $\forall i \in \{0,1\}, \forall j \in [c]$. 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 $j \in [c]$. In addition, according to Π_{rr1cs} in Cstr. 1 and due to the homomorphic and binding properties of the commitment scheme, for each $i \in [2]$, it implies that

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

Since the equation above holds for three distinct $\{\alpha^{(i)}\}_{i\in[3]}$, we must have $\mathbb{x}_i.\tilde{z}_j = \mathsf{C.Commit}(\mathbb{x}_i.\mathbf{z}_j, \mathbb{x}_i.\hat{z}_j)$ for all $j \in \{0,1\}$, or equivalently, $[\![\mathbb{x}_i.\mathbf{z}_j]\!]_{\mathsf{ck}_j} \in \mathcal{R}_{\mathsf{com}}\ \forall i \in \{0,1\}, \forall j \in [c]$. 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 $i' \in \{0,1\}, j \in [c]$, 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 Rmk. 8.

Similarly, by considering $\mathbb{x}^{(i)}.\tilde{\mathsf{e}} = \mathbb{x}_0.\tilde{\mathsf{e}} + \alpha^{(i)} \cdot \tilde{\mathsf{g}} + \left(\alpha^{(i)}\right)^2 \cdot \mathbb{x}_1.\tilde{\mathsf{e}}$, 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 $\mathbf{z}'_i = (\mathbb{x}_i.\mathsf{pub} \| \mathbb{x}_i.\mathbf{z}_1 \| \ldots \| \mathbb{x}_i.\mathbf{z}_c)$. Note that, since

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

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 holds, 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 dinstinct 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. □

A.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 Def. 17 and their security properties in Def. 18, 19 and 20.

Definition 17 (Syntax of Honest-Verifier ZKAoK/ZKPoK). *Let $\mathcal{Z}$ and $\mathcal{W}$ denotes the instance and witness set, respectively. Let $\mathcal{R} \subseteq \mathcal{Z} \times \mathcal{W}$ be a relation. A ZKAoK for $\mathcal{R}$ is a tuple*

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

consists of the algorithm ZK.Setup *and interactive protocol* 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.

$\mathsf{ZK.Prove}(\mathsf{pp}, Z;\ W) \to \{0,1\}$ *This is an interactive protocol between prover and verifier, where the prover holds an instance-witness pair $(Z, W) \in \mathcal{Z} \times \mathcal{W}$ and the verifier holds an instance $Z \in \mathcal{Z}$, such that the prover tries to convince the verifier that he knows $W \in \mathcal{W}$ 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 with syntax defined in Def. 17. We now recall the completeness, knowledge soundness and (honest-verifier) zero-knowledge for $\mathcal{ZK}$ in the following Def. 18, 19 and 20, respectively.

Definition 18 (Completeness of $\mathcal{ZK}$). *$\mathcal{ZK}$ satisfies completeness if for any $(Z;\ W) \in \mathcal{R}$ it holds that*

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

Definition 19 (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*

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

We write $\mathcal{E}^{\mathcal{A}}(\mathsf{pp}, Z)$ 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, Z). In addition, $\mathcal{E}$ has an ability to 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 $(Z, W) \in \mathcal{R}$. 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 20 (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 $(Z, W) \in \mathcal{R}$ and for any (PPT) adversary $\mathcal{A}$, then the two following distributions are indistinguishable from the view of $\mathcal{A}$:*

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

where $\mathsf{tr} \leftarrow \mathsf{View}(\mathsf{ZK.Prove}(\mathsf{pp}, Z; W))$ *denotes the public transcript which contains all the public inputs and exchanged messages between 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.*

A.11 Lagrange Interpolation

We recall the Lagrange interpolation theorem in the following Thm. 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 $\ell_i(X) = \prod_{\substack{j \in [0,m] \\ s.t.\ j \neq i}} \dfrac{X - x_j}{x_i - x_j}$.

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

We recall the technique for handling basic circuit satisfiability from compressed Σ-protocol theory (Sec. 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 (see Appdx. A.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

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

for all $i \in [1, m]$, 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.

B Generic CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$ Supporting pvRAM (Extended)

We provide a proof of Thm. 1 in Appdx. B.1.

B.1 Proof of Thm. 1

We first recall Thm. 1 in the following Thm. 4.

Theorem 4 (Recall of Thm. 1). *If* $\mathsf{CF.Prove}$ *is an (HV)ZKAoK/PoK and* $\mathcal{C}$ *is a secure homomorphic commitment scheme, then* $\mathcal{CF}_{\mathsf{gnr}}$ *satisfies perfect completeness, HVZK and knowledge soundness with soundness error* $\mathcal{O}(N/|\mathbb{F}| + \mathsf{serr}_{\mathsf{prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda))$ *where* $\mathsf{serr}_{\mathsf{prf}}(\mathsf{pp})$ *is the 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 Sec. 5.1, for a witness vector

$$\mathbb{z} = (\mathsf{lt}, \mathsf{rt}, \mathbb{x}, \mathbf{i}, \mathbf{o}, \mathbb{x}^\star, \mathbf{s}, \mathbf{a})$$

having the form of (11), we can parse

$$\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} = (\mathsf{lt}, \mathsf{rt}, \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}, \llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}, \llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a} \rrbracket_{\mathsf{ck}_5})$$

where, for $\tilde{\mathbb{x}}$ and $\tilde{\mathbb{x}}^\star$ of the form (47), and for $\hat{\mathbb{x}}$ and $\hat{\mathbb{x}}^\star$ of the form (48),

$$\begin{aligned}
\llbracket \mathbb{x} \rrbracket_{\mathsf{tck}} &= (\mathsf{tck}, \tilde{\mathbb{x}};\ \mathbf{x}, \hat{\mathbb{x}}), & \llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'} &= (\mathsf{tck}', \tilde{\mathbb{x}}^\star;\ \mathbb{x}^\star, \hat{\mathbb{x}}^\star)\\
\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1} &= (\mathsf{ck}_1, \tilde{\mathsf{i}};\ , \mathbf{i}, \hat{\mathsf{i}}) & \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2} &= (\mathsf{ck}_2, \tilde{\mathsf{o}};\ , \mathbf{o}, \hat{\mathsf{o}})\\
\llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5} &= (\mathsf{ck}_5, \tilde{\mathsf{s}};\ , \mathbf{s}, \hat{\mathsf{s}}) & \llbracket \mathbf{a} \rrbracket_{\mathsf{ck}_5} &= (\mathsf{ck}_5, \tilde{\mathsf{a}};\ , \mathbf{a}, \hat{\mathsf{a}})
\end{aligned}$$

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

$$\tilde{\mathbb{z}} = (\mathsf{lt}, \mathsf{rt}, \tilde{\mathbb{x}}, \tilde{\mathsf{i}}, \tilde{\mathsf{o}}, \tilde{\mathbb{x}}^\star, \tilde{\mathsf{s}}, \tilde{\mathsf{a}}), \tag{50}$$

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

$$\hat{\mathbb{z}} = (\hat{\mathbb{x}}, \hat{\mathsf{i}}, \hat{\mathsf{o}}, \hat{\mathbb{x}}^\star, \hat{\mathsf{s}}, \hat{\mathsf{a}}). \tag{51}$$

The notation $\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}}$ is the same as (12).

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

Proof (Proof of Thm. 1). The proof follows Lm. 10, 12 and 13 for correctness, knowledge soundness and HVZK, respectively. □

Correctness of $\mathcal{CF}_{\mathsf{gnr}}$. Correctness follows the following Lm. 10.

Lemma 10 (Correctness of $\mathcal{CF}_{\mathsf{gnr}}$). $\mathcal{CF}_{\mathsf{gnr}}$ *is perfectly correct if* $\mathcal{C}$ *is a homomorphic and perfectly correct commitment scheme and* $\mathsf{CF.Prove}$*, for relation* $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$*, is perfectly complete for relation* $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$.

Proof. The proof is straightforward. □

Knowledge Soundness of $\mathcal{CF}_{\mathsf{gnr}}$. Notice that protocol $\Pi_{\mathcal{CF}_{\mathsf{gnr}}}$ (in (18)) uses $\Pi_{\mathsf{fold\text{-}gnr}}$ (in Cstr. 2) as a building block to fold following a binary-tree-like hierarchical structure $\mathsf{HS} \in \mathcal{HS}_{0N}$. Therefore, we first analyze the extraction in each folding by $\Pi_{\mathsf{fold\text{-}gnr}}$ according Lm. 11. Then, we formalize Lm. 12 the knowledge soundness of $\mathcal{CF}_{\mathsf{gnr}}$ by extracting following a binary-tree-like HS and employing Lm. 11 as a building block. Details are as follows.

Lemma 11 ($(3,3;|\mathbb{F}|,|\mathbb{F}|)$-Special Soundness of $\Pi_{\mathsf{fold\text{-}gnr}}$). *Assume that* $\mathcal{C}$*, with respect to relation* $\mathcal{R}_{\mathsf{com}}$*, is homomorphic and binding. Assume that, on inputs* $\llbracket \mathbb{z}_0 \rrbracket_{\mathsf{pp}}$ *and* $\llbracket \mathbb{z}_1 \rrbracket_{\mathsf{pp}}$ *of the forms*

$$\llbracket \mathbb{z}_i \rrbracket_{\mathsf{pp}} = (\mathsf{lt}_i, \mathsf{rt}_i, \llbracket \mathbb{x}_i \rrbracket_{\mathsf{tck}}, \llbracket \mathbf{i}_i \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o}_i \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbb{x}^\star_i \rrbracket_{\mathsf{tck}'}, \llbracket \mathbf{s}_i \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a}_i \rrbracket_{\mathsf{ck}_5})\ \forall i \in \{0,1\},$$

protocol $\Pi_{\mathsf{fold\text{-}gnr}}$ *in Cstr. 2 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*

$$\begin{aligned}
\llbracket \mathbb{z}^{(i_1,i_2)} \rrbracket_{\mathsf{pp}} = (&\mathsf{lt}, \quad \mathsf{rt}, \quad \llbracket \mathbb{x}^{(i_1)} \rrbracket_{\mathsf{tck}}, \quad \llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, \quad \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2},\\
&\llbracket (\mathbb{x}^\star)^{(i_1,i_2)} \rrbracket_{\mathsf{tck}'}, \quad \llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5}, \quad \llbracket \mathbf{a}^{(i_1)} \rrbracket_{\mathsf{ck}_5}),
\end{aligned}$$

as follows:

– $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'} = (1, 1, \llbracket \mathbf{o}_0 \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbf{i}_1 \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{w} \rrbracket_{\mathsf{ck}_3'}, \llbracket \mathbf{0}^{n'} \rrbracket_{\mathsf{cke}'})$ *(step 2 in Cstr. 2).*

– $\{\alpha_1^{(i_1)}\}_{i_1 \in [3]}$ *are distinct and, for each* $i_1 \in [3]$*, by running step 4 of* Π_{rr1cs}*,*
  - $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}$ *and* $\llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}$ *are folded into* $\llbracket \mathbb{x}^{(i_1)} \rrbracket_{\mathsf{tck}}$ *w.r.t.* $\alpha_1^{(i_1)}$*, and*
  - $\llbracket \mathbb{x}_0^\star \rrbracket_{\mathsf{tck}'}$ *and* $\llbracket \mathbb{x}_1^\star \rrbracket_{\mathsf{tck}'}$ *are folded into* $\llbracket \mathbb{y}'^{(i_1)} \rrbracket_{\mathsf{tck}'}$ *w.r.t.* $\alpha_1^{(i_1)}$*.*

  (The above process is similar to step 4 in Cstr. 2.)

– *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}*,*
  - $\llbracket \mathbb{y}'^{(i_1)} \rrbracket_{\mathsf{tck}'}$ *and* $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'}$ *are folded into* $\llbracket (\mathbb{x}^\star)^{(i_1,i_2)} \rrbracket$ *w.r.t.* $\alpha_2^{(i_1,i_2)}$*.*

  (The above process is similar to step 8 in Cstr. 2.)

– $\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1} = \llbracket \mathbf{i}_0 \rrbracket_{\mathsf{ck}_1}$*,* $\llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2} = \llbracket \mathbf{o}_1 \rrbracket_{\mathsf{ck}_2}$*,* $\llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5} = \llbracket \mathbf{s}_0 \rrbracket_{\mathsf{ck}_5} + \llbracket \mathbf{s}_1 \rrbracket_{\mathsf{ck}_5}$*, and* $\llbracket \mathbf{a}^{(i_1)} \rrbracket_{\mathsf{ck}_5} = \llbracket \mathbf{a}_0 \rrbracket_{\mathsf{ck}_5} + \alpha_1 \cdot \llbracket \mathbf{a}_1 \rrbracket_{\mathsf{ck}_5} + \alpha_1^2 \cdot (\llbracket \mathbf{s}_0 \rrbracket_{\mathsf{ck}_5} - \llbracket \mathbf{s}_1 \rrbracket_{\mathsf{ck}_5})$ *(step 9 in Cstr. 2).*

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

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

where relations $\mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$ *and* $\mathcal{R}_{\mathsf{gnr\text{-}cond}}^{\mathfrak{S}'}$ *are defined in* (16) *and* (17)*, respectively.*

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

Extracting all Witnesses and Randomness of $\llbracket \mathbb{z}_i \rrbracket_{\mathsf{pp}}$ $\forall i \in \{0,1\}$. We proceed as follows:

– *Extracting witnesses and randomness of* $\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}$ *and* $\llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2}$. As we have all witnesses and randomness of all $\llbracket \mathbb{z}^{(i_1,i_2)} \rrbracket_{\mathsf{pp}}$ for all $i_1 \in [3]$ and all $i_2 \in [3]$, we know (from the statement of this lemma) that $\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1} = \llbracket \mathbf{i}_0 \rrbracket_{\mathsf{ck}_1}$ and $\llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2} = \llbracket \mathbf{o}_1 \rrbracket_{\mathsf{ck}_2}$. Therefore, we also achieve all witnesses and randomness of $\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}$ and $\llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2}$ according to Rmk. 3.
  As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

$$\llbracket \mathbb{z}^{(i_1,i_2)} \rrbracket_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'} \implies \llbracket \mathbb{z}^{(i_1,i_2)} \rrbracket_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}com}} \implies (\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2} \in \mathcal{R}_{\mathsf{com}}).$$

  Hence, we deduce that $\llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2} \in \mathcal{R}_{\mathsf{com}}$.

– *Extracting witnesses and randomness of* $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}$ *and* $\llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}$. Since there are 3 distinct challenges $\{\alpha_1^{(i_1)}\}_{i_1 \in [3]}$, and, for each $i_1 \in [3]$, $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}$ and $\llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}$ are folded into $\llbracket \mathbb{x}^{(i_1)} \rrbracket_{\mathsf{tck}}$ w.r.t. $\alpha_1^{(i_1)}$. Moreover, we also have witnesses and randomness of $\llbracket \mathbb{z}^{(i_1,i_2)} \rrbracket_{\mathsf{pp}}$, for all $i_1 \in [3]$ and all $i_2 \in [3]$, containing those of $\llbracket \mathbb{x}^{(i_1)} \rrbracket_{\mathsf{tck}}$, for all $i_1 \in [3]$. We hence can apply Lm. 1 (recalled in Lm. 9) to extract witnesses and randomness of $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}$ and $\llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}}$.
  As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

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

  Hence, by Lm 1, we deduce that $\llbracket \mathbb{x}_0 \rrbracket_{\mathsf{tck}}, \llbracket \mathbb{x}_1 \rrbracket_{\mathsf{tck}} \in \mathcal{R}_{\mathsf{rr1cs}}^{\mathfrak{S}}$.

– *Extracting witnesses and randomness of* $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'}$*,* $\llbracket \mathbb{x}_0^\star \rrbracket_{\mathsf{tck}'}$ *and* $\llbracket \mathbb{x}_1^\star \rrbracket_{\mathsf{tck}'}$. For each $i_1 \in [3]$, since there are 3 distinct challenges $\{\alpha_2^{(i_1,i_2)}\}_{i_2 \in [3]}$, and, for each $i_2 \in [3]$, $\llbracket \mathbb{y}'^{(i_1)} \rrbracket_{\mathsf{tck}'}$ and $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'}$ are folded into $\llbracket (\mathbb{x}^\star)^{(i_1,i_2)} \rrbracket$ w.r.t. $\alpha_2^{(i_1,i_2)}$. Moreover, we also have all witnesses and randomness of all $\{\llbracket \mathbb{z}^{(i_1,i_2)} \rrbracket_{\mathsf{pp}}\}_{i_1 \in [3], i_2 \in [3]}$ containing those of $\{\llbracket (\mathbb{x}^\star)^{(i_1,i_2)} \rrbracket\}_{i_1 \in [3], i_2 \in [3]}$. For each $i_1 \in [3]$, we hence can apply Lm. 1 (recalled in Lm. 9), we can extract all witnesses and randomness of $\llbracket \mathbb{y}'^{(i_1)} \rrbracket_{\mathsf{tck}'}$ and $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'}$. Moreover, from the statement of this lemma, for all $i_1 \in [3]$, $\llbracket \mathbb{x}_0^\star \rrbracket_{\mathsf{tck}'}$ and $\llbracket \mathbb{x}_1^\star \rrbracket_{\mathsf{tck}'}$ are folded into $\llbracket \mathbb{y}'^{(i_1)} \rrbracket_{\mathsf{tck}'}$ w.r.t. $\alpha_1^{(i_1)}$. Since $\{\alpha_1^{(i_1)}\}_{i_1 \in [3]}$ are distinct, we again apply Lm. 1 to extract all witnesses and randomness of $\llbracket \mathbb{x}_0^\star \rrbracket_{\mathsf{tck}'}$ and $\llbracket \mathbb{x}_1^\star \rrbracket_{\mathsf{tck}'}$.
  As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

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

  By Lm. 1, the extracted witnesses and randomness $\llbracket \mathbb{y}'^{(i_1)} \rrbracket_{\mathsf{tck}'}$ and $\llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'}$ satisfy $\llbracket \mathbb{y}'^{(i_1)} \rrbracket_{\mathsf{tck}'}, \llbracket \mathbb{y} \rrbracket_{\mathsf{tck}'} \in \mathcal{R}_{\mathsf{rr1cs}}^{\mathfrak{S}'}$. Again, by Lm. 1, $\llbracket \mathbb{x}_0^\star \rrbracket_{\mathsf{tck}'}, \llbracket \mathbb{x}_1^\star \rrbracket_{\mathsf{tck}'} \in \mathcal{R}_{\mathsf{rr1cs}}^{\mathfrak{S}'}$ as desired.

- *Extracting all witnesses and randomnesses of* $[\![\mathbf{a}_0]\!]_{\mathsf{ck}_5}$, $[\![\mathbf{a}_1]\!]_{\mathsf{ck}_5}$, $[\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} - [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5}$. For each $i_1 \in [3]$,

$$[\![\mathbf{a}^{(i_1)}]\!]_{\mathsf{ck}_5} = [\![\mathbf{a}_0]\!]_{\mathsf{ck}_5} + \alpha_1^{(i_1)} \cdot [\![\mathbf{a}_1]\!]_{\mathsf{ck}_5} + (\alpha_1^{(i_1)})^2 \cdot ([\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} - [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5})$$

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 $\{[\![\mathbf{a}^{(i_1)}]\!]_{\mathsf{ck}_5}\}_{i_1\in[3]}$. Therefore, we can extract all witnesses and randomnesses of $[\![\mathbf{a}_0]\!]_{\mathsf{ck}_5}$, $[\![\mathbf{a}_1]\!]_{\mathsf{ck}_5}$ and $[\![\mathbf{s}']\!]_{\mathsf{ck}_5} = [\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} - [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5}$ 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 $i_2 \in [3]$,

$$\begin{aligned}[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} &\implies [\![(\mathbf{a})^{(i_1)}]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}} \\ &\implies [\![\mathbf{a}_0]\!]_{\mathsf{ck}_5}, [\![\mathbf{a}_1]\!]_{\mathsf{ck}_5}, [\![\mathbf{s}']\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}.\end{aligned}$$

- *Extracting all witnesses and randomness of* $[\![\mathbf{s}_0]\!]_{\mathsf{ck}_5}$ *and* $[\![\mathbf{s}_1]\!]_{\mathsf{ck}_5}$. We have $[\![\mathbf{s}]\!]_{\mathsf{ck}_5} = [\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} + [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5}$ in the problem statement, and $[\![\mathbf{s}']\!]_{\mathsf{ck}_5} = [\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} - [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5}$. Moreover, we also have all witnesses and randomness of $[\![\mathbf{s}]\!]_{\mathsf{ck}_5}$ and $[\![\mathbf{s}']\!]_{\mathsf{ck}_5}$. Therefore, by solving equations, we can obtain all witnesses and randomness of $[\![\mathbf{s}_0]\!]_{\mathsf{ck}_5}$ and $[\![\mathbf{s}_1]\!]_{\mathsf{ck}_5}$.

As in the statement of this lemma, for all $i_1 \in [3]$ and $i_2 \in [3]$,

$$[\![\mathbb{z}^{(i_1,i_2)}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}} \implies [\![\mathbf{s}]\!]_{\mathsf{ck}_5} = [\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} + [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}.$$

Since $[\![\mathbf{s}']\!]_{\mathsf{ck}_5} = [\![\mathbf{s}_0]\!]_{\mathsf{ck}_5} - [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}$ as proved above, hence the extracted $[\![\mathbf{s}_0]\!]_{\mathsf{ck}_5}, [\![\mathbf{s}_1]\!]_{\mathsf{ck}_5} \in \mathcal{R}_{\mathsf{com}}$.

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

$$[\![\mathbb{z}_i]\!]_{\mathsf{pp}} = (\mathsf{lt}_i, \mathsf{rt}_i, [\![\mathbb{x}_i]\!]_{\mathsf{tck}}, [\![\mathbf{i}_i]\!]_{\mathsf{ck}_1}, [\![\mathbf{o}_i]\!]_{\mathsf{ck}_2}, [\![\mathbb{x}^\star_i]\!]_{\mathsf{tck}'}, [\![\mathbf{s}_i]\!]_{\mathsf{ck}_5}, [\![\mathbf{a}_i]\!]_{\mathsf{ck}_5}) \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}.$$

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

As proved above, $[\![\mathbb{y}]\!]_{\mathsf{tck}'} \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{rr1cs}}$. 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 $\mathbf{c}' = (1\|\mathbf{o}_0\|\mathbf{i}_1\|\mathbf{w})$. Hence, it holds that the witness $\mathbf{w}$ in $\mathbb{y}$ satisfies $([\![\mathbb{z}_0]\!]_{\mathsf{pp}}, [\![\mathbb{z}_1]\!]_{\mathsf{pp}};\ \mathbf{w}) \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$. □

Before proceeding the proof of knowledge soundness of $\mathcal{CF}_{\mathsf{gnr}}$, we additionally define the notion of *levels* in hierarchical structures in the following Def. 21.

Definition 21 (Levels in Hierarchical Structures). *We view each hierarchical structure* $\mathsf{HS} \in \mathcal{HS}_{lr}$*, according to Def. 2, for any* $l, r \in \mathbb{N}$ *satisfying* $l < r$*, as a binary tree such that*

- *Each node is labeled by* (i,j) *for all* $(i,j) \in \mathsf{HS}$*;*
- *For all* $i \in [0, N-1]$*,* $(i, i+1)$ *is a leaf node; and*
- *For all* $(i,j) \in \mathsf{HS}$ *satisfying* $i+1 < j$*, by definition of* HS*, there exists a unique* k *such that* $(i,k), (k,j) \in \mathsf{HS}$ *are direct child nodes of* (i,j)*.*

We denote the level of a node $(i,j) \in \mathsf{HS}$ *to be* $\mathsf{level}(i,j) = 0$ *if* (i,j) *is a leaf node, and* $\mathsf{level}(i,j) = \max\{\mathsf{level}(i,k), \mathsf{level}(k,j)\} + 1$ *if* $i+1 < j$ *and* $(i,j), (i,k), (k,j) \in \mathsf{HS}$*.*

Lemma 12 (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* $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$*, is knowledge-sound, by constructing extractor rewinding* 9^L *times where* L *is the depth of* $\mathsf{HS} \in \mathcal{HS}_{0N}$*. If* $L = \mathcal{O}(\log N)$*, then* $9^L = \mathsf{poly}(N)$ *implying that extractor is PPT. Moreover,* $\mathcal{CF}_{\mathsf{gnr}}$ *has soundness error*

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

where $\mathsf{serr}_{\mathsf{prf}}(\mathsf{pp})$ *is the soundness error of* $\mathsf{CF.Prove}$*, with respect to relation* $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$*.*

Proof. Assume that prover and verifier fold N instance-witness pairs, namely, $(\llbracket \mathbb{z}_{(i-1)i} \rrbracket_{\mathsf{pp}})_{i=1}^{N}$, following a hierarchical structure $\mathsf{HS} \in \mathcal{HS}_{0N}$ to obtain the final instance-witness pair $\llbracket \mathbb{z}_{0N} \rrbracket_{\mathsf{pp}}$ and proceed $\mathsf{CF.Prove}$ for $\llbracket \mathbb{z}_{0N} \rrbracket_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{pvr\text{-}prf}}^{\mathfrak{S},\mathfrak{S}'}$. The entire process can be summarize by writing

$$\mathit{\Pi}_{\mathcal{CF}_{\mathsf{gnr}}}(\mathsf{HS}, \mathsf{pp}, \{\llbracket \mathbb{z}_{(i-1)i} \rrbracket_{\mathsf{pp}}\}_{i\in[N]};\ \mathsf{sec}) \to 1$$

as in (18) where sec is some prover's secret. Let L be the depth of HS, namely, $L = \mathsf{level}(0, N)$ where $\mathsf{level} : [0, N] \times [0, N] \to \mathbb{Z}_+$ is defined in Def. 21. We now show how to extract witnesses for each level from L to 0 by constructing a sequence of extractors $(\mathcal{E}_\ell)_{\ell=0}^{L}$. Intuitively, when $\mathcal{E}_\ell$, for $\ell \in [0, L]$, finishes its job, all witnesses $\mathbb{z}_{ij}$, satisfying $\mathsf{level}(i, j) \geq \ell$, are extracted.

Before extracting, we recall some important notations: For the notation

$$\llbracket \mathbb{z} \rrbracket_{\mathsf{pp}} = (\mathsf{lt}, \mathsf{rt}, \llbracket \mathbb{x} \rrbracket_{\mathsf{tck}}, \llbracket \mathbf{i} \rrbracket_{\mathsf{ck}_1}, \llbracket \mathbf{o} \rrbracket_{\mathsf{ck}_2}, \llbracket \mathbb{x}^\star \rrbracket_{\mathsf{tck}'}, \llbracket \mathbf{s} \rrbracket_{\mathsf{ck}_5}, \llbracket \mathbf{a} \rrbracket_{\mathsf{ck}_5}),$$

its public instance $\tilde{\mathbb{z}}$ and corresponding randomness $\hat{\mathbb{z}}$ are defined in (50) and (51) respectively. The public instance $\tilde{\mathbb{z}}$ and randomness $\hat{\mathbb{z}}$ will be used throughout the proof.

We proceed the proof of this lemma in three steps: (i) extracting at top level, (ii) extracting from below levels and (iii) analyzing soundness error.

Extracting at Top Level. At level L, we denote by $\mathcal{E}_L$ to be the extractor at level L that, on inputs pp and $\tilde{\mathbb{z}}_{0N}$ of the form of (50), returns $\mathbb{z}_{0N}$ and $\tilde{\mathbb{z}}_{0N}$. This extractor runs by calling $\mathcal{E}_{\mathsf{prf}}$ of $\mathsf{CF.Prove}$, since $\mathsf{CF.Prove}$ is knowledge-sound, to obtain the witnesses and randomness for $\llbracket \mathbb{z}_{0N} \rrbracket_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$.

Extracting from Below Levels. For each level $\ell \in [0, L-1]$, we construct extractor $\mathcal{E}_\ell$ for level ℓ in a way that $\mathcal{E}_\ell$ first collects all nodes at level ℓ, such that corresponding witnesses are unable to be extracted by $\mathcal{E}_{\ell+1}$, into the set $\mathcal{M}'_\ell$. From here, we denote by $\mathcal{N}_\ell$ the nodes in higher levels (from $\ell+1$ to L) that nodes in $\mathcal{M}'_\ell$ are direct children. Then, we additionally define $\mathcal{M}_\ell$ to be set of nodes that are direct children of those in $\mathcal{N}_\ell$. It can be seen that $\mathcal{M}'_\ell \subseteq \mathcal{M}_\ell$. We refer readers to check Fig. 4 for an example of making these sets, namely, $\mathcal{M}'_\ell, \mathcal{N}_\ell, \mathcal{M}_\ell$ for all $\ell \in [0, L-1]$.

Since each node $\mathfrak{n}$ in $\mathcal{N}_\ell$ has direct pair of children $\mathfrak{m}_0, \mathfrak{m}_1$ in $\mathcal{M}_\ell$, according to Lm. 11, if we can repeat 9 times to

- rewind by folding pairs, e.g., $\llbracket \mathbb{z}_{\mathfrak{m}_0} \rrbracket_{\mathsf{pp}}$ and $\llbracket \mathbb{z}_{\mathfrak{m}_1} \rrbracket_{\mathsf{pp}}$, at nodes $\mathfrak{m}_0, \mathfrak{m}_1$ into pair, e.g., $\llbracket \mathbb{z}_{\mathfrak{n}}^{(i_1,i_2)} \rrbracket_{\mathsf{pp}}$, at node $\mathfrak{n}$ by following $(3, 3)$-tree of transcript w.r.t. constraints for challenges $\{\alpha_1^{(i_1)}\}_{i_1\in[3]}$ and $\{\alpha_2^{(i_1,i_2)}\}_{i_1\in[3],i_2\in[3]}$ specified in Lm. 11, and
- obtain witnesses and randomness for $\llbracket \mathbb{z}_{\mathfrak{n}}^{(i_1,i_2)} \rrbracket_{\mathsf{pp}}$ (which is done by extractor $\mathcal{E}_{\ell+1}$),

then we can extract witnesses and randomness for $\llbracket \mathbb{z}_{\mathfrak{m}_0} \rrbracket_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$ and $\llbracket \mathbb{z}_{\mathfrak{m}_1} \rrbracket_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$ with conditions between them hold according to relation $\mathcal{R}_{\mathsf{gnr\text{-}cond}}^{\mathfrak{S}'}$.

Notice that, since HS is a binary-tree structure, and any node $\mathfrak{n} \in \mathcal{N}_\ell$ has exactly 2 direct child nodes in $\mathcal{M}_\ell$, we see that $\mathcal{M}_\ell$ partitions into $|\mathcal{N}_\ell|$ pairs such that each pair has a common parent node in $\mathcal{N}_\ell$. Hence, in the folding context, the foldings of $|\mathcal{N}_\ell|$ pairs are independent. Therefore, we instruct extractor $\mathcal{E}_\ell$ to do the above process for all nodes in $\mathcal{N}_\ell$ with their respective pairs of children in the following sense: $\mathcal{E}_\ell$ devises independently each $(3, 3)$-tree of challenges for each folding among those $|\mathcal{N}_\ell|$ foldings. $\mathcal{E}_\ell$ finishes its job when all rewindings w.r.t. all those $|\mathcal{N}_\ell|$ $(3, 3)$-trees of challenges finish. Then, we can extract all witnesses and randomness for pairs corresponding to nodes in $\mathcal{M}_\ell$. Notice that, by working simultaneously, we only need 9 times of rewinding.

Since extract $\mathcal{E}_\ell$ uses $\mathcal{E}_{\ell+1}$ as a subroutine with 9 times of rewinding, hence, in general, to extract the full L-level hierarchical HS, we need 9^L times of rewinding. Specifically, $\mathcal{E}_0$ calls $\mathcal{E}_1$ for 9 times and, for each call to $\mathcal{E}_1$, $\mathcal{E}_1$ again calls $\mathcal{E}_2$ for 9 times. This deduces that $\mathcal{E}_0$ calls $\mathcal{E}_2$ for 9^2 times. By generalizing, $\mathcal{E}_0$ calls $\mathcal{E}_L$ for 9^L times. Here, $\mathcal{E}_0$ is the one mentioned in the statement of this lemma.

Thus, by rewinding 9^L times with the above strategy, we can obtain all witnesses $\{\mathbf{w}_i\}_{i\in N-1}$ and all witnesses and randomness for $\llbracket \mathbb{z}_{(i-1)i} \rrbracket_{\mathsf{pp}}$, for all $i \in [N]$, such that

$$\bigwedge_{i=1}^{N} \left(\llbracket \mathbb{z}_{(i-1)i} \rrbracket_{\mathsf{pp}} \in \mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'} \right) \text{ and}$$

$$\bigwedge_{i=1}^{N-1} \left((\llbracket \mathbb{z}_{(i-1)i} \rrbracket_{\mathsf{pp}}, \llbracket \mathbb{z}_{i(i+1)} \rrbracket_{\mathsf{pp}};\ \mathbf{w}_i) \in \mathcal{R}_{\mathsf{gnr\text{-}cond}}^{\mathfrak{S}'} \right)$$

For example, in Fig. 5, we describe a hierarchical structure $\mathsf{HS} \in \mathcal{HS}_{03}$ for folding. This tree has depth $L = 3$ such that level 0 is the set $\{(0,1),(1,2),(2,3)\}$, level 1 is the set $\{(0,2)\}$ and level 2 is the set $\{(0,3)\}$. By using our strategy, extractor $\mathcal{E}_2$ extracts witness at node $(0,3)$. Then, extractor $\mathcal{E}_1$ extracts witnesses at nodes $(0,2)$ and $(2,3)$. Notice that, $(2,3)$ is in level 0. Therefore, extractor $\mathcal{E}_0$ only needs to extract witnesses at nodes $(0,1)$ and $(1,2)$ at level 0 since witness at $(2,3)$ is already extracted. Hence,

- $\mathcal{E}_2$ (extractor of $\mathsf{CF.Prove}$) can extract at node $(0,3)$.
- $\mathcal{E}_1$ collects $\mathcal{M}_1' = \{(0,2)\}$. Then, we deduce

$$\mathcal{N}_1 = \{(0,3)\} \implies \mathcal{M}_1 = \{(0,2),(2,3)\} \implies \mathcal{M}_1' \subseteq \mathcal{M}_1.$$

- $\mathcal{E}_0$ collects $\mathcal{M}_0' = \{(0,1),(1,2)\}$. Then, we deduce

$$\mathcal{N}_0 = \{(0,2)\} \implies \mathcal{M}_0 = \{(0,1),(1,2)\} \implies \mathcal{M}_0' \subseteq \mathcal{M}_0.$$

Fig. 4. An example for collecting $\mathcal{M}_\ell'$,$\mathcal{N}_\ell$ and $\mathcal{M}_\ell$ for $\ell \in \{0,1\}$ with a $\mathsf{HS} \in \mathcal{HS}_{03}$.

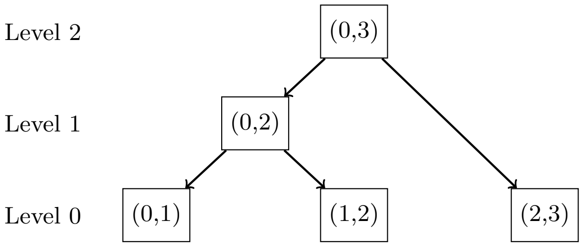


Fig. 5. An example of $\mathsf{HS} \in \mathcal{HS}_{03}$.

by calling extractor $\mathcal{E}_0$.

Soundness error analysis. An adversary breaking knowledge soundness of $\mathcal{CF}_{\mathsf{gnr}}$ either breaks binding property of commitment scheme $\mathcal{C}$ or luckily receives bad challenges, namely, challenges that benefits cheating prover, from verifier. We analyze the case of receiving bad challenges from verifier.

By Lm. 11, having valid witnesses for three transcripts with respect to $(3,3)$-tree of challenges $\left(\alpha_1^{(i_1)}, \alpha_2^{(i_1,i_2)}\right)_{i_1,i_2 \in [3]}$ is sufficient to extract back valid witnesses $\mathbb{z}_i$ for all $i \in \{0,1\}$. Hence, we see that the probability that $\alpha_1 \xleftarrow{\$} \mathbb{F}$ is bad is at most $\frac{2}{|\mathbb{F}|}$. Moreover, the probability that α_2 is bad is also at most $\frac{2}{|\mathbb{F}|}$ given a good α_1. Hence, the probability that (α_1,α_2) is bad is at most $\frac{2}{|\mathbb{F}|} + \left(1 - \frac{2}{|\mathbb{F}|}\right) \cdot \frac{2}{|\mathbb{F}|} \leq \frac{4}{|\mathbb{F}|}$.

Now recall that, at level $\ell \in [0, L-1]$, there are $|\mathcal{N}_\ell|$ independent foldings. By using union bound, the probability that bad challenges appear among those $|\mathcal{N}_\ell|$ foldings is at most $\frac{4 \cdot |\mathcal{N}_\ell|}{|\mathbb{F}|}$.

By denoting P_ℓ, for $\ell \in [0, L-1]$, to be the probability that bad challenges appear among the foldings from levels 0 to ℓ, we see

$$\begin{aligned}
P_0 &= |\mathcal{N}_0| \cdot \frac{4}{|\mathbb{F}|} \text{ and} \\
P_\ell &\leq P_{\ell-1} + (1 - P_{\ell-1}) \cdot \frac{4 \cdot |\mathcal{N}_\ell|}{|\mathbb{F}|} \\
&= P_{\ell-1} + \frac{4 \cdot |\mathcal{N}_\ell|}{|\mathbb{F}|} - \frac{4 \cdot |\mathcal{N}_\ell| \cdot P_{\ell-1}}{|\mathbb{F}|} \\
&\leq P_{\ell-1} + \frac{4 \cdot |\mathcal{N}_\ell|}{|\mathbb{F}|} \\
&\leq \sum_{i=0}^{\ell} \frac{4 \cdot |\mathcal{N}_i|}{|\mathbb{F}|} = \frac{4 \cdot \sum_{i=0}^{\ell} |\mathcal{N}_i|}{|\mathbb{F}|} \qquad \forall \ell \in [1, L-1].
\end{aligned}$$

Hence, we see that $P_{L-1} = \frac{4 \cdot \sum_{i=0}^{L-1} |\mathcal{N}_i|}{|\mathbb{F}|} = \frac{4 \cdot (N-1)}{|\mathbb{F}|}$ since there are exactly $N-1$ such foldings.

Finally, since we denoted $\mathsf{serr}_{\mathsf{prf}}(\mathsf{pp})$ to be the soundness error of protocol $\mathsf{CF.Prove}$, with respect to relation $\mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$. We see that soundness error of $\Pi_{\mathcal{CF}_{\mathsf{gnr}}}$ is upper bounded by

$$\begin{aligned} & P_{L-1} + (1 - P_{L-1}) \cdot \mathsf{serr}_{\mathsf{prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda) \\ & \leq P_{L-1} + \mathsf{serr}_{\mathsf{prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda) \\ & = \mathcal{O}\left(\frac{N}{|\mathbb{F}|} + \mathsf{serr}_{\mathsf{prf}}(\mathsf{pp}) + \mathsf{negl}(\lambda) \right) \end{aligned}$$

where $\mathsf{negl}(\lambda)$ stands for the probability of breaking binding property of $\mathcal{C}$. Hence, we conclude the proof. □

HVZK of $\mathcal{CF}_{\mathsf{gnr}}$. HVZK follows the following Lm. 13.

Lemma 13 (HVZK of $\mathcal{CF}_{\mathsf{gnr}}$). *CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ is HVZK if $\mathcal{C}$ is an additively homomorphic and hiding commitment scheme and* $\mathsf{CF.Prove}$*, for relation $\mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$, is (honest-verifier) zero-knowledge.*

Proof. Let tr be the transcript

$$\mathsf{tr} \leftarrow \mathsf{View}(\Pi_{\mathcal{CF}_{\mathsf{gnr}}}(\mathsf{HS}, \mathsf{pp}, (\tilde{\mathbb{z}}_{(i-1)i})_{i=1}^{N};\ \mathsf{sec}))$$

as defined in (9) with respect to CF scheme $\mathcal{CF}_{\mathsf{gnr}}$ where $\tilde{\mathbb{z}}_{(i-1)i}$ is defined in (50). Notice that tr contains the transcript of all executions of $\mathsf{CF.Fold}$, the final instance $\tilde{\mathbb{z}}_{0N}$ and execution of $\mathsf{CF.Prove}$. Hence, we can write

$$\mathsf{tr} = (\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathbb{z}}_{0N} \| \mathsf{tr}_{\mathsf{prf}})$$

where $\mathsf{tr}_{\mathsf{fold}}$ is the transcript of all executions of $\mathsf{CF.Fold}$ and $\mathsf{tr}_{\mathsf{prf}}$ is the transcript of $\mathsf{CF.Prove}$ when $\tilde{\mathbb{z}}_{0N}$ is achieved after all foldings.

Assume that $\mathcal{S}_{\mathsf{prove}}$ is the simulator for $\mathsf{CF.Prove}$, namely, simulator of ZAKoK argument $\mathsf{CF.Prove}$ for relation $\mathcal{R}_{\mathsf{gnr\text{-}inst}}^{\mathfrak{S},\mathfrak{S}'}$. To show that $\mathcal{CF}_{\mathsf{gnr}}$ satisfies HVZK, we first call simulator $\mathcal{S}_{\mathsf{prove}}$ on inputs $(\mathsf{pp}, \tilde{\mathbb{z}}_{0N})$ to obtain transcript

$$\mathsf{tr}' = \left(\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathbb{z}}_{0N} \| \mathsf{tr}'_{\mathsf{prf}}\right)$$

where $\mathsf{tr}'_{\mathsf{prf}}$ is simulated transcript output by $\mathcal{S}_{\mathsf{prove}}$. By HVZK of $\mathsf{CF.Prove}$, we see that tr and tr' are indistinguishable.

We now show how to construct

$$\mathsf{tr}^{\star} = (\mathsf{tr}^{\star}_{\mathsf{fold}} \| \tilde{\mathbb{z}}^{\star}_{0N} \| \mathsf{tr}^{\star}_{\mathsf{prf}}),$$

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 $\mathsf{tr}^{\star}$. Notice that, for each folding, prover needs to send $(\tilde{\mathbb{g}}, \tilde{\mathbb{w}}, \tilde{\mathbb{g}}_1)$, namely, commitments in $\llbracket \mathbf{g} \rrbracket_{\mathsf{cke}}, \llbracket \mathbf{w} \rrbracket_{\mathsf{ck}'_3}$ and $\llbracket \mathbf{g}_1 \rrbracket_{\mathsf{cke}'}$ as in Cstr. 2, before receiving challenge α_1, and $\tilde{\mathbb{g}}_2$, namely, commitment in $\llbracket \mathbf{g}_2 \rrbracket_{\mathsf{cke}'}$ in Cstr. 2, before receiving α_2. Therefore, simulator simply commits to zero vectors with randomness sampled appropriately, namely, following the correct distribution of randomness sampling, to obtain those *dummy* commitments $\tilde{\mathbb{g}}, \tilde{\mathbb{w}}, \tilde{\mathbb{g}}_1$ and $\tilde{\mathbb{g}}_2$. By 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 $\mathsf{CF.Prove}$ only computes dummy commitments and send to verifier to obtain $(\mathsf{tr}^{\star}_{\mathsf{fold}} \| \tilde{\mathbb{z}}^{\star}_{0N})$. Then, it calls simulator $\mathcal{S}_{\mathsf{prove}}$, on input $(\mathsf{pp}, \tilde{\mathbb{z}}^{\star}_{0N})$, of $\mathsf{CF.Prove}$ to get the simulated transcript $\mathsf{tr}^{\star}_{\mathsf{prf}}$. Finally, form the simulated transcript $\mathsf{tr}^{\star} = (\mathsf{tr}^{\star}_{\mathsf{fold}} \| \tilde{\mathbb{z}}^{\star}_{0N} \| \mathsf{tr}^{\star}_{\mathsf{prf}})$.

We now analyze how $\mathsf{tr}^{\star}$ is indistinguishable from tr'. We first notice that $(\mathsf{tr}^{\star}_{\mathsf{fold}} \| \tilde{\mathbb{z}}^{\star}_{0N})$ and $(\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathbb{z}}_{0N})$ are indistinguishable according to the hiding property of commitment scheme $\mathcal{C}$. Then, $(\mathsf{pp}, \tilde{\mathbb{z}}^{\star}_{0N})$ is passed to $\mathcal{S}_{\mathsf{prove}}$ for producing simulated proof. Notice that, if $\mathcal{S}_{\mathsf{prove}}$ is unable to produce $\mathsf{tr}^{\star}_{\mathsf{prf}}$, then we can construct a distinguisher to employ $\mathcal{S}_{\mathsf{prove}}$ as a sub-routine to distinguish $(\mathsf{tr}^{\star}_{\mathsf{fold}} \| \tilde{\mathbb{z}}^{\star}_{0N})$ and $(\mathsf{tr}_{\mathsf{fold}} \| \tilde{\mathbb{z}}_{0N})$, since simulator $\mathcal{S}_{\mathsf{prove}}$ can produce simulated proof on input $(\mathsf{pp}, \tilde{\mathbb{z}}_{0N})$, contradicting the hiding property of $\mathcal{C}$. Similarly, if the output $\mathsf{tr}^{\star}_{\mathsf{prf}}$ is distinguishable from $\mathsf{tr}'_{\mathsf{prf}}$, then it also contradicts hiding property of commitment scheme $\mathcal{C}$.

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

Remark 9. A variant of Lm. 13 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 witness mainly contains commitments and the final proof which ensure witness indistinguishability by hiding of commitments and witness indistinguishability of the final proof.

C pvRAM From CF Scheme $\mathcal{CF}_{\mathsf{gnr}}$ (Extended)

In Appdx. C.1, we discuss the set of constraints that $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C})$ and $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$ capture as said in Sec. 6.2. Then, in Appdx. C.2, we provide the proof of Thm. 2.

C.1 Detailed Description of Public Matrices for Construction of $\mathit{\Pi}_{\mathsf{pvr}}$

In this section, we provide a detail description of the public matrices $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C})$ and $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$ for the construction of $\mathit{\Pi}_{\mathsf{pvr}}$ in Sec. 6. Recall (29) from Section 6.2 that, for all $i \in [N]$,

$$\begin{aligned}
&\mathbf{z}_{i1} = (\overline{\mathsf{pc}}_i \| \overline{\mathsf{reg}}_i \| \overline{\mathsf{macs}}_i \| \mathsf{macs}^\star_i), \quad && \mathbf{z}_{i2} = (\mathsf{pc}_i \| \mathsf{reg}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i), \quad \mathbf{z}_{i3} = \mathsf{plkst}_i,\\
&\mathbf{z}_{i4} = \mathsf{aux}_i, && \mathbf{z}_{i5} = (\mathsf{plkiv}_i \| \mathsf{miv}_i \| \mathsf{miv}'_i)
\end{aligned}$$

Here, we slightly change notation for $\mathbf{z}_{i1}$ to contain $\mathsf{macs}^\star_i$ instead of macs'_i so that, below, we use $\mathbf{A}, \mathbf{B}, \mathbf{C}$ to enforce $\mathsf{macs}^\star_i = \mathsf{macs}'_i$. Moreover, we also have global challenge $\mathsf{gchal} = (\gamma, \delta, \chi, \psi, \tau, \omega) \in \mathbb{F}^6$. By denoting $\mathbf{z}_i = (1 \| \mathbf{z}_{i1} \| \ldots \| \mathbf{z}_{i5})$, we now describe the meaning of matrices $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ for the conditions of a single computation step below:

$$\begin{aligned}
&\mathbf{A} \cdot \mathbf{z}_i \circ \mathbf{B} \cdot \mathbf{z}_i = \mathbf{C} \cdot \mathbf{z}_i\\
\iff &\begin{cases}
\mathsf{macs}^\star_i = \mathsf{macs}'_i,\\
(\mathsf{pc}_i, \mathsf{reg}_i, \mathsf{macs}_i) = F_{\overline{\mathsf{pc}}_i}(\overline{\mathsf{reg}}_i, \overline{\mathsf{macs}}_i, \mathsf{aux}_i)\\
\overline{\mathsf{macs}}_i \text{ and } \mathsf{macs}_i\\
\qquad \text{are related through constraints in (34)}\\
\qquad \text{by viewing } \mathsf{macs}_{i-1} \text{ as } \overline{\mathsf{macs}}_i\\
\mathsf{macs}_i \text{ satisfies first equation in (33)}\\
\mathsf{plkst}_i \text{ is the PLONK structure of } F_{\overline{\mathsf{pc}}_i}\\
\qquad \text{w.r.t randomness } \gamma, \delta\\
\mathsf{plkiv}_i(\chi + \mathsf{plkcp}_i) = 1 \text{ where } \mathsf{plkcp}_i = \langle (\overline{\mathsf{pc}}_i \| \mathsf{plkst}_i), \boldsymbol{\psi} \rangle\\
\mathsf{miv}_i(\tau + \mathsf{mcp}_i) \in \{0, 1\} \text{ where } \mathsf{mcp}_i = \langle \mathsf{macs}_i, \boldsymbol{\omega} \rangle\\
\mathsf{miv}'_i(\tau + \mathsf{mcp}'_i) \in \{0, 1\} \text{ where } \mathsf{mcp}'_i = \langle \mathsf{macs}'_i, \boldsymbol{\omega} \rangle
\end{cases}
\end{aligned}$$

Next recall that for all $i \in [N]$

$$\begin{aligned}
&\mathbf{i}_i = (\overline{\mathsf{pc}}_i \| \overline{\mathsf{reg}}_i \| \overline{\mathsf{macs}}_i \| \mathsf{macs}^\star_i) \quad \mathbf{o}_i = (\mathsf{pc}_i \| \mathsf{reg}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i),\\
&\mathbf{c}_i = (1 \| \mathbf{o}_{i-1} \| \mathbf{i}_i \| \mathbf{w}_i)
\end{aligned}$$

where $\mathbf{w}_i$ is a supporting witness. We describe the meaning of matrices $\mathbf{A}', \mathbf{B}$ and $\mathbf{C}'$ for the conditions of consecutive computation steps below:

$$\mathbf{A}' \cdot \mathbf{c}_i \circ \mathbf{B}' \cdot \mathbf{c}_i = \mathbf{C}' \cdot \mathbf{c}_i \iff \begin{cases}
\overline{\mathsf{macs}}_i = \mathsf{macs}_{i-1}\\
\mathsf{macs}'_{i-1} \text{ and } \mathsf{macs}^\star_i\\
\qquad \text{are related through constraints in (34)}\\
\qquad \text{by viewing } \mathsf{macs}'_i \text{ as } \mathsf{macs}^\star_i\\
\overline{\mathsf{reg}}_i = \mathsf{reg}_{i-1}, \ \overline{\mathsf{pc}}_i = \mathsf{pc}_{i-1}
\end{cases}$$

C.2 Proof of Thm. 2

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

Theorem 5 (Recall of Thm. 2). *If $\mathcal{C}$ is secure homomorphic commitment scheme and $\Pi_{\mathsf{pvr\text{-}prf}}$ is an HVZKAoK/HVZKPoK for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ then Π_{pvr} is an HVZKAoK/HVZKPoK for relation $\mathcal{R}_{\mathsf{ram}}$ in* (30) *with soundness error $\mathcal{O}(n_{\mathsf{plk}} \cdot (N+T)/|\mathbb{F}|+\mathsf{serr}_{\mathsf{pvr\text{-}prf}}(\mathsf{pp})+\mathsf{negl}(\lambda))$ where $\mathsf{serr}_{\mathsf{pvr\text{-}prf}}(\mathsf{pp})$ is soundness error of $\Pi_{\mathsf{pvr\text{-}prf}}$.*

Proof (Proof of Thm. 2). The proof follows Lm. 14, 15 and 16. Specifically, we present Lm. 14, 15 and 16 for proving the security of $\Pi_{\mathsf{fold\text{-}gnr}}$ in Thm. 2. Specifically, Lm. 14, 16 and 16 are for correctness, knowledge soundness and HVZK of Π_{pvr}, respectively, hence together they imply the proof of Thm. 2. □

Correctness of Π_{pvr}. Correctness follows Lm. 14.

Lemma 14 (Completeness of Π_{pvr}). *If $\mathcal{C}$ is perfectly correct and $\Pi_{\mathsf{pvr\text{-}prf}}$, for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$, is perfectly complete, then Π_{pvr} is perfectly complete.*

Proof. The proof is straightforward. □

Knowledge Soundness of Π_{pvr}. Knowledge soundness follows Lm. 15 below. It employs Lm. 15 for extracting the witnesses and prove that the extracted witnesses together satisfy (i) the constraints of a single computation step, (ii) the constraints of consecutive computational steps and (iii) constraints of all computation steps, as specified in Sec. 6.1. Details are as follows.

Lemma 15 (Knowledge Soundness of Π_{pvr}). *If $\mathcal{C}$ is an additively homomorphic and binding and $\Pi_{\mathsf{pvr\text{-}prf}}$, for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$, is knowledge-sound, then Π_{pvr} is knowledge-sound. Moreover, Π_{pvr} has soundness error*

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

where $\mathsf{serr}_{\mathsf{pvr\text{-}prf}}(\mathsf{pp})$ is soundness error of $\Pi_{\mathsf{pvr\text{-}prf}}$.

Proof. Since $\Pi_{\mathsf{pvr\text{-}prf}}$ 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 $\Pi_{\mathsf{pvr\text{-}prf}}$ as specified in (32). By Lm. 12, we see that knowledge soundness of step 7 implies knowledge soundness protocol $\Pi_{\mathcal{CF}_{\mathsf{gnr}}}$ defined in (18). Hence, we can extract $\mathbb{z}_{ij}$, for all $(i,j) \in \mathsf{HS}$ such that $[\![\mathbb{z}_{ij}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{gnr\text{-}inst}}$. By $\Pi_{\mathsf{pvr\text{-}prf}}$, it also holds that $[\![\mathbb{z}_{0N}]\!]_{\mathsf{pp}} \in \mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$. Moreover, all of the conditions between to-be-folded instance-witness pairs also hold, i.e.,

$$\bigwedge_{\substack{i,k,j \text{ s.t.} \\ (i,k),(k,j)\in\mathsf{HS}}} \left(([\![\mathbb{z}_{ik}]\!]_{\mathsf{pp}}, [\![\mathbb{z}_{kj}]\!]_{\mathsf{pp}};\ \mathbf{w}_k) \in \mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}\right)$$

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

Moreover, we also can extract $(\mathsf{mul}_j)_{j=1}^T$ and randomness $(\hat{\mathsf{m}}_j)_{j=1}^T$ by the extractor of $\Pi_{\mathsf{pvr\text{-}prf}}$ for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$ such that

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

for all $j \in [T]$. Below, it suffices that our extracted witnesses $\mathbb{z}_{(i-1)i}$ together satisfy the constraints of a single computation step, constraints between consecutive computation steps, and constraints for all computation steps for a RAM program, as specified in Sec. 6.1 and we are done.
Extracting components for single computation steps. Recall that in Sec. 6.2, 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 global challenge $\mathsf{gchal} = (\gamma, \delta, \chi, \psi, \tau, \omega)$. 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 (12). Hence, for each $i \in [N]$, by following Sec. 2.2, 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 $\mathit{\Pi}_{\mathsf{pvr}}$ in Cstr. 4, 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

$$\mathbf{A} \cdot \mathbf{z}'_i \circ \mathbf{B} \cdot \mathbf{z}'_i = \mathbf{C} \cdot \mathbf{z}'_i \tag{52}$$

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 Sec. 6.2, we parse

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

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 macs'_i is due to the constraints in $\mathfrak{S} = (\mathbf{A}, \mathbf{B}, \mathbf{C})$, specified in Appdx. C.1. Hence, we have

$$\begin{aligned}
\mathbf{z}'_i = (1 \| \overline{\mathsf{pc}}_i \| \overline{\mathsf{reg}}_i \| \overline{\mathsf{macs}}_i \| \mathsf{pc}_i \| \mathsf{reg}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i&\\
\| \mathsf{plkst}_i \| \mathsf{aux}_i \| \mathsf{plkiv}_i \| \mathsf{miv}_i \| \mathsf{miv}'_i)&
\end{aligned} \tag{53}$$

for all $i \in [N]$. By (52), we deduce that the computation of the i-th execution step is sound with respect to the above detailed parsing of $\mathbf{z}'_i$ in (53).

Since $\mathbf{A}, \mathbf{B}, \mathbf{C}$ realize the testing of hidden evaluation of hidden circuits by employing PLONK's arithmetization with respect to plkst_i, for all $i \in [N]$, by employing challenges γ and δ, according to Appdx. A.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 all $i \in [N]$.

Extracting components for conditions. For each $i \in [N-1]$, by parsing $\mathfrak{S}' = (\mathbf{A}', \mathbf{B}', \mathbf{C}')$, where $\mathbf{A}', \mathbf{B}', \mathbf{C}'$ are public matrices defined in Sec. 6.2, recall that our extracted witnesses $\mathbb{z}_{(i-1)i}$ and auxiliary witnesses $\mathbf{w}_i$ satisfy

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

it holds that

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

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

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

in Cstr. 4. By the binding property of $\mathcal{C}$, it holds that

$$\mathbf{c}_i = (1 \| \mathsf{pc}_i \| \mathsf{reg}_i \| \mathsf{macs}_i \| \mathsf{macs}'_i \| \overline{\mathsf{pc}}_{i+1} \| \overline{\mathsf{reg}}_{i+1} \| \overline{\mathsf{macs}}_{i+1} \| \mathsf{macs}'_{i+1} \| \mathbf{w}_i).$$

Hence, constraints between two consecutive steps holds with respect to matrices $\mathbf{A}', \mathbf{B}'$ and $\mathbf{C}'$ and relation $\mathcal{R}^{\mathfrak{S}'}_{\mathsf{gnr\text{-}cond}}$.

Guaranteeing constraints for all computation steps. For all $i, k, j \in \mathbb{N}$ satisfying $(i,k), (k,j) \in \mathsf{HS}$, our extracted witnesses satisfy $\mathbf{s}_{ij} = \mathbf{s}_{ik} + \mathbf{s}_{kj}$ according to the proof of Lm. 12, we deduce that

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

According to step 6 in Cstr. 4, since verifier has computed $\tilde{\mathsf{s}}_{(i-1)i} = \mathbb{x}_{(i-1)i}.\tilde{\mathsf{z}}_5$, by the binding property of $\mathcal{C}$, it holds that

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

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

$$\begin{cases} \sum_{i=1}^N \mathsf{plkiv}_i = \mathsf{plkiv}, \\ \sum_{i=1}^N \mathsf{miv}_i = \mathsf{miv}, \\ \sum_{i=1}^N \mathsf{miv}'_i = \mathsf{miv}'. \end{cases}$$

Since $\Pi_{\mathsf{pvr\text{-}prf}}$ for relation $\mathcal{R}_{\mathsf{pvr\text{-}prf}}^{\mathfrak{S},\mathfrak{S}'}$ guarantees that

$$\mathsf{plkiv} = \sum_{j=1}^{T} \mathsf{mul}_j \cdot (\chi + \mathsf{plkcp}'_j)^{-1} \text{ and } \mathsf{miv} = \mathsf{miv}',$$

we hence see that

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

according to (21) for using $\mathsf{plkcp}'_j = \big\langle (j \| \mathsf{plkst}'_j), (\psi^k)_{k=0}^{n_{\mathsf{plk}}} \big\rangle$. By (52) and (53), we know that

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

according to (22), and

$$\begin{cases} \mathsf{miv}_i \neq 0 \wedge \big(\tau + \big\langle \mathsf{macs}_i, (\omega^k)_{k=0}^3 \big\rangle \big) \cdot \mathsf{miv}_i \in \{0,1\}, \text{ and} \\ \mathsf{miv}'_i \neq 0 \wedge \big(\tau + \big\langle \mathsf{macs}'_i, (\omega^k)_{k=0}^3 \big\rangle \big) \cdot \mathsf{miv}'_i \in \{0,1\}, \end{cases}$$

according to (25), (26) and (27). We now divide the proof into two cases, namely, testing tuple lookup and testing tuple permutation, as follows:

- *Testing tuple lookup.* By the above extractions, we know that

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

  By employing (45) and Lm. 7, if $\{(\mathsf{pc}_{i-1} \| \mathsf{plkst}_i)\}_{i=1}^N \not\subseteq \{(j \| \mathsf{plkst}'_j)\}_{j=1}^T$, then the above equality holds with probability at most

$$\frac{n_{\mathsf{plk}}(N+T-1)}{|\mathbb{F}| - T} = \mathcal{O}\left(\frac{n_{\mathsf{plk}} \cdot (N+T)}{|\mathbb{F}|}\right).$$

- *Testing tuple permutation.* Recall that

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

  if the case that $\tau + \big\langle \mathsf{macs}_i, (\omega^k)_{k=0}^3 \big\rangle = 0$ or $\tau + \big\langle \mathsf{macs}'_i, (\omega^k)_{k=0}^3 \big\rangle = 0$, for some $i \in [1, N]$, does not happen. Since $\tau \xleftarrow{\$} \mathbb{F}$ and $\omega \xleftarrow{\$} \mathbb{F}$, such a case happens with probability at most $\frac{6N}{|\mathbb{F}|}$ by Schwartz-Zippel lemma and union bound. In case that such a case does not happen, by employing (42) and Lm. 4, if $(\mathsf{macs}_i)_{i=1}^N$ and $(\mathsf{macs}'_i)_{i=1}^N$ are not permutation of each other, then

$$\sum_{i=1}^{N} \mathsf{miv}_i = \sum_{i=1}^{N} \mathsf{miv}'_i$$

if the case that there exists some $i \in [N]$ satisfying $\tau + \langle \mathsf{macs}_i, (\omega^k)_{k=0}^3 \rangle = 0$ or $\tau + \langle \mathsf{macs}'_i, (\omega^k)_{k=0}^3 \rangle = 0$ does not happen. Since $\tau \xleftarrow{\$} \mathbb{F}$ and $\omega \xleftarrow{\$} \mathbb{F}$, such a case happens with probability at most $\frac{6N}{|\mathbb{F}|}$ by Schwartz-Zippel lemma and union bound. In case that such a case does not happen, by employing (42) and Lm. 4, if $(\mathsf{macs}_i)_{i=1}^N$ and $(\mathsf{macs}'_i)_{i=1}^N$ are not permutation of each other, then

$$\sum_{i=1}^N \mathsf{miv}_i = \sum_{i=1}^N \mathsf{miv}'_i$$

happens with probability at most $\frac{3(2N-1)}{|\mathbb{F}|}$. Hence, the error probability for checking tuple permutation is at most

$$\frac{12N-3}{|\mathbb{F}|} = \mathcal{O}\left(\frac{N}{|\mathbb{F}|}\right)$$

by using union bound.

- *Testing copy constraints in PLONK.* Since we need two additional challenges $\gamma, \delta \xleftarrow{\$} \mathbb{F}$ for testing the copy constraints of each computation step, by union bound, the error probability for checking copy constraints for all steps is at most $n_{\mathsf{wit}} \cdot N / |\mathbb{F}| \leq n_{\mathsf{plk}} \cdot N / |\mathbb{F}|$ which is equal to

$$\mathcal{O}\left(\frac{n_{\mathsf{plk}} \cdot N}{|\mathbb{F}|}\right)$$

where error probability of each step is $\mathcal{O}\left(\frac{n_{\mathsf{plk}}}{|\mathbb{F}|}\right)$ according to Appdx. A.3.

Correct input and output. Parse

$$\mathbf{i}_{0N} = (\overline{\mathsf{pc}} \| \overline{\mathsf{reg}} \| \overline{\mathsf{macs}} \| \mathsf{macs}^\star) \text{ and } \mathbf{o}_{0N} = (\mathsf{pc} \| \mathsf{reg} \| \mathsf{macs} \| \mathsf{macs}').$$

Notice that relation $\mathit{\Pi}_{\mathsf{pvr}}$ enforces $\overline{\mathsf{pc}} = 1, \overline{\mathsf{reg}} = \mathbf{i}$, $\mathsf{reg} = \mathbf{o}$ and $(\mathsf{cki}, \tilde{\mathsf{i}};\ \mathbf{i}, \hat{\mathsf{i}}) \in \mathcal{R}_{\mathsf{com}}$, by the binding property of $\mathcal{C}$.

For all $i, k, j \in \mathbb{N}$ satisfying $(i,k), (k,j) \in \mathsf{HS}$, since verifier has computed $\tilde{\mathsf{i}}_{ij} = \tilde{\mathsf{i}}_{ik}$ and $\tilde{\mathsf{o}}_{ij} = \tilde{\mathsf{o}}_{kj}$ according to step 9 of $\mathit{\Pi}_{\mathcal{CF}_{\mathsf{gnr}}}$ in Cstr. 2, by the binding property of $\mathcal{C}$, it holds that $\mathbf{i}_{0N} = \mathbf{i}_{01}$ and $\mathbf{o}_{0N} = \mathbf{o}_{(N-1)N}$.

Moreover, according to step 6 of Cstr. 4, since verifier has set $\tilde{\mathsf{i}}_{(i-1)i} = \mathbb{x}_{(i-1)i}.\tilde{\mathsf{z}}_1$ and $\tilde{\mathsf{o}}_{(i-1)i} = \mathbb{x}_{(i-1)i}.\tilde{\mathsf{z}}_2$ for all $i \in [N]$, by binding property of $\mathcal{C}$, it also holds that $\mathbf{i}_{0N} = \mathbf{i}_{01} = \mathbb{x}_{01}.\mathbf{z}_1$ and $\mathbf{i}_{0N} = \mathbf{o}_{(N-1)N} = \mathbb{x}_{(N-1)N}.\mathbf{z}_2$.

Hence, $\mathbf{i} = \overline{\mathsf{reg}}$ and $\mathbf{o} = \mathsf{reg}$ are the input and output, respectively, of the first and last computation steps, respectively. Moreover, $\overline{\mathsf{pc}} = 1$ implies that the first instruction is the starting instruction in the instruction set. By using $\mathit{\Pi}_{\mathsf{pvr\text{-}prf}}$ for relation $\mathcal{R}_{\mathsf{pvr\text{-}prf}}^{\mathfrak{S},\mathfrak{S}'}$ in (32), we know that $\overline{\mathsf{reg}} = \mathsf{reg}_{\mathsf{in}}$ and $\mathsf{reg} = \mathsf{reg}_{\mathsf{out}}$ implying that RAM program receives as input $\mathsf{reg}_{\mathsf{in}}$ and returns output $\mathsf{reg}_{\mathsf{out}}$ correctly.
Analysis of soundness error. In summary, soundness error of $\mathit{\Pi}_{\mathsf{pvr}}$ is upper bounded by soundness error of $\mathit{\Pi}_{\mathcal{CF}_{\mathsf{gnr}}}$ plus the probability that testing correct executions of hidden circuits, testing tuple lookup, testing tuple permutation or testing copy constraints is error is at most

$$\frac{N \cdot n_{\mathsf{plk}}}{|\mathbb{F}|} + \frac{n_{\mathsf{plk}}(N+T-1)}{|\mathbb{F}| - T} + \frac{12N-3}{|\mathbb{F}|} = \mathcal{O}\left(\frac{n_{\mathsf{plk}} \cdot (N+T)}{|\mathbb{F}|}\right).$$

Notice that the proof $\mathit{\Pi}_{\mathcal{CF}_{\mathsf{gnr}}}$ has soundness error

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

according to Lm. 12 where $\mathsf{serr}_{\mathsf{pvr\text{-}prf}}(\mathsf{pp})$ is soundness error of $\mathit{\Pi}_{\mathsf{pvr\text{-}prf}}$. Hence, soundness error of $\mathit{\Pi}_{\mathsf{pvr}}$ is

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

Thus, we conclude the proof. □

HVZK of $\mathit{\Pi}_{\mathsf{pvr}}$. HVZK of $\mathit{\Pi}_{\mathsf{pvr}}$ follows Lm. 16.

Lemma 16 (HVZK of $\mathit{\Pi}_{\mathsf{pvr}}$). *If $\mathcal{C}$ is an additively homomorphic and hiding commitment scheme and $\mathit{\Pi}_{\mathsf{pvr\text{-}prf}}$, for relation $\mathcal{R}^{\mathfrak{S},\mathfrak{S}'}_{\mathsf{pvr\text{-}prf}}$, is HVZK, then $\mathit{\Pi}_{\mathsf{pvr}}$ is HVZK.*

Proof. Since $\mathit{\Pi}_{\mathsf{pvr\text{-}prf}}$ is HVZK, it implies that $\mathit{\Pi}_{\mathcal{CF}_{\mathsf{gnr}}}$ is HVZK according to Lm. 13. Since $\mathit{\Pi}_{\mathsf{pvr}}$ employs $\mathit{\Pi}_{\mathcal{CF}_{\mathsf{gnr}}}$ as a sub-routine, we can straightforwardly conclude that $\mathit{\Pi}_{\mathsf{pvr}}$ is HVZK. □