KiloNova: Zero-Knowledge PCD from Non-Uniform Multi-Folding Schemes

Abstract—**Proof-carrying data (PCD) is a powerful cryptographic primitive generalizing incrementally verifiable computation (IVC) that enables sequential computation by multiple distrusting parties. However, the development of PCD faces some bottlenecks. First, directly adapting folding schemes for IVC to PCD leads to inefficiencies due to increasing cross terms in the proofs. Second, existing PCD can hardly support non-uniform circuits and zero-knowledge proof, which are required in building virtual machines like zkEVM.**

This paper introduces KiloNova, a new zero-knowledge PCD construction built from non-uniform multi-folding schemes. Compared to its predecessors, KiloNova provides more desirable features while maintaining high efficiency. Firstly, it avoids cross terms in the multi-folding schemes, even when handling non-uniform circuits with high-degree gates. Secondly, KiloNova provides zero knowledge without compromising the prover time, through an efficient zero-knowledge folding scheme and a new construction for zero-knowledge PCD. We also introduce a technique for delegating the costly folding of non-uniform circuits, enhancing KiloNova's efficiency in real-world applications.

Experimental evaluations demonstrate that KiloNova outperforms competing approaches when dealing with multiple non-uniform instances. By avoiding the cross terms found in prior work, KiloNova's recursion overhead is only dominated by $O(\log(n))$ random-oracle-like hashes and $O(k)$ scalar multiplications, where n is the circuit input length and k is the instance number. As a result, the recursion overhead of KiloNova is an order of magnitude smaller than other schemes when dealing with multiple non-uniform instances.

1. Introduction

Recently, there has been a surge of interest in the realization of *Incrementally Verifiable Computation* (IVC) [1], a cryptographic primitive that runs sequential computations while allowing efficient verification of the execution at any point. As a generalization of IVC to directed acyclic graphs, *Proof-Carrying Data* (PCD) [2] allows multiple distrusting parties to perform computations sequentially. This property enables a wider range of applications such as distributed computation [3], [4] and blockchain technology [5], [6], [7]. Meanwhile, the ability to handle multiple instances in each round provides a broader spectrum of trade-offs for system performance, as discussed in [8].

The traditional approach to constructing IVC/PCD employs a general-purpose SNARK at each step i to attest the correctness of the proof output by step $i-1$ recursively. This construction requires implementing the whole verification logic in the SNARK proving circuit (the recursive circuit), incurring a significant overhead because the verification may include costly operations such as elliptic curve pairings [9]. A recent line of work proposes a more practical idea to "defer" the expensive operations in the proof verification and run them together at the end of IVC/PCD, including Halo [9], Halo infinite [10], BCMS20 [11], BCLMS21 [12], Nova [13], HyperNova [14], Protostar [15], etc. Instead of checking the SNARK/NARK proofs (new instances) from the prior step in the recursive circuit, the prover defers them by folding/accumulating (use interchangeably) with a so-called running instance and continues the incremental computations. As a result, the expensive recursive verification of the new instances is replaced with a cheaper verification of the folding scheme. Finally, the IVC/PCD verifier (or the decider), conducts a batch verification [16] to the single running instance.

Although some IVC/PCDs' built from folding schemes [13], [14], [15] have been proposed in recent years, their capabilities are limited. First, few existing solutions can satisfy the implementation of virtual machines with rich instruction sets (e.g., EVM, RISC-V, Wasm), which requires handling *non-uniform* high-degree circuits. Second, previous work only considers IVC/PCD run by a *single* prover without providing *zero-knowledge* new instances at each step (refer to Section 5 in Nova [13]), which hinders application on some privacy-focused scenarios with mutually distrustful parties. Although these are two desirable features, implementing them based on existing PCD constructions is not trivial work. We highlight the main challenges as follows.

Non-uniform circuits. Generally, achieving non-uniform circuits encounters two problems. The first is the exponentially growing communication cost introduced in folding instances with non-uniform circuits. For example, when folding two quadratic instances (w_i, t_i) such that $w_i^2 = t_i$ for $i = 1, 2$, the folded instance $(w, t) = (w_1 + rw_2, t_1 + rt_2)$, where r is a random challenge. The folding prover has to compute and send an extra "cross term" as $2w_1w_2$ for verification. When considering folding s instances in d-degree relations, the number of cross terms will grow in $O(d^s)$. Despite the communication cost problem, a non-uniform folding scheme also leads to the growing density of circuit matrices because folding two sparse matrices yields a denser one as $A^* = A_1 + r \cdot A_2$. This problem makes it difficult to preprocess the circuits [17] to reduce the verification cost.

Zero-knowledge PCD. As mentioned above, the requirement for zero-knowledge proofs spans a broad spectrum of applications, including anonymous De-Fi, confidential transactions, and private smart contracts [18], [19], [20]. However, implementing zero knowledge in IVC/PCD presents significant challenges. A primary issue arises from the gap between PCD constructions and folding schemes. According to prior research [11], a PCD prover has to ensure zero knowledge for all the running instance and new instances [1]. This requirement incurs heavy costs when employing folding schemes because the instances output by NARK systems do not guarantee succinctness. Assume adding zero-knowledge by masking the witness in a high-degree plonkish circuit, the prover has to send extra elements growing in exponential number. Consequently, we have to develop an efficient zero-knowledge folding scheme and a more practical construction for zero-knowledge PCD.

Upon the current status, we develop our motivation in two steps. First, can we build a non-uniform multi-folding scheme for high-degree relations without cross terms? If so, can we efficiently turn it into a zero-knowledge folding scheme and construct zero-knowledge PCD based on it?

1.1. Our Contributions

We answer the above questions positively and present KiloNova, a zero-knowledge PCD from generic folding schemes based on the theory and systems contributions below.

(1) Relaxed CCS relations. Motivated by Nova [13], and HyperNova [14], we introduce a new relaxed version of Customizable Constraint Systems (CCS) relation [21] called Atomic CCS (ACCS) to enable our folding scheme to efficiently deal with non-uniform circuits. Since CCS is expressive to generalize Plonkish, R1CS, and AIR with high-degree constraints without overheads, our folding scheme is capable for current applications such as zkVM. Similar to the linearized committed CCS relation in HyperNova, this new ACCS relation is also reduced from the original CCS relation by partially running an "early stopping" version of SuperSpartan [21]. The difference is that our protocol runs an extra round of sum-check protocol than HyperNova, which stops right before the costly compilation of the polynomial Oracle queries for the second sum-check protocol. Thus, the obtained ACCS instances as input of the folding schemes contain independent linear claims on structures (circuits), introducing no cross terms in the folding of multiple non-uniform instances.

1. This conclusion conforms with the theorem given in [11], requiring both a zk accumulation scheme and zk NARK (for CCS in our case) to construct a zk-PCD.

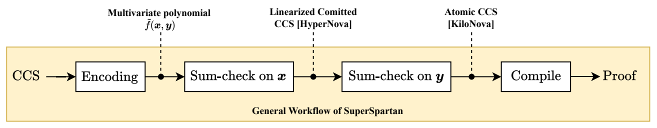
Multivariate polynomial
$\tilde{f}(\boldsymbol{x}, \boldsymbol{y})$
Linearized Comitted
CCS [HyperNova]
Atomic CCS
[KiloNova]
CCS
Encoding
Sum-check on $\boldsymbol{x}$
Sum-check on $\boldsymbol{y}$
Compile
Proof
General Workflow of SuperSpartan

Figure 1: Illustrations for Atomic CCS relations.

(2) Generic folding schemes. Based on the atomic CCS relations, we design a folding scheme for multiple non-uniform CCS instances, denoted as the generic folding scheme. Generally speaking, the generic folding scheme runs the "early-stopping" SuperSpartan protocol for each CCS instance in parallel and aggregates their transcripts together. The remaining query operations are folded and formalized as *one* atomic CCS instance, and its verification is "deferred" to the final verifier. Therefore, the recursive circuit at each step only contains the cheap verification of folding. The following theorem captures its cryptographic and efficiency characteristics. Since the folding scheme is public coin, it can be transformed into a non-interactive version in the random oracle model according to the Fiat-Shamir heuristic [22].

Theorem 1. *There exists a constant-round, public-coin folding scheme for multiple N-sized CCS instances with non-uniform circuit "structures" (i.e., CCS coefficient matrices), the prover's work is $O_\lambda(s \cdot N)$, and the verifier's work and the communication are both $O_\lambda(\log N)$, assuming the existence of any additively-homomorphic commitment scheme that provides $O_\lambda(1)$-sized commitments to N-sized vectors over $\mathbb{F}$ (e.g., Pedersen's commitments), where λ is the security parameter and s is the number of instances.*

For the convenience of building zero-knowledge PCD, we further introduce a zero-knowledge version of the generic folding scheme by utilizing the *padding technique* [23] and *dummy instances*. The obtained protocol neither introduces cross terms nor increases the asymptotic complexity of the prover and verifier.

(3) New Constructions of Zero-Knowledge PCD. Based on the generic folding schemes, it is feasible to obtain a non-uniform PCD according to the previous constructions [11], [13], which enables runtime circuit selection with the proving cost and recursive overhead independent of the sizes of "uninvoked" circuits. However, constructing zero-knowledge PCD encounters the efficiency problem in transforming the high-degree NARK proof into zero-knowledge version as mentioned previously. To solve this issue, we modify the construction of PCD by splitting the task for each recursive step into two independent sub-circuits. This modification allows the PCD prover to transform the CCS instances generated from the first sub-circuit into zero-knowledge atomic CCS instances by reusing our generic folding scheme in the second one rather than introducing an extra costly protocol. Theoretically, this new approach removes the requirement of a zero-knowledge NARK in the construction of zero-knowledge PCD. It can also be applied to existing IVC systems to achieve zero-knowledge. We summarize the new construction model as a theorem below.

Theorem 2. *(informal). There is an efficient construction that compiles any NARK with a zero-knowledge multi-folding scheme into zero-knowledge PCD. Additionally, if the multi-folding scheme is post-quantum secure, then PCD*

TABLE 1: Features comparison between existing folding/accumulation schemes

Schemes	Degree	Non-Uniform	Multi-Fold	ZK
Nova[13]	2	No/Yes in [24]	No	No
SuperNova[26]	2	Yes	No	No
HyperNova[14]	d	No	Yes in [27]	No
BCLMS21 [12]	2	No	No	Yes
Protostar [15]	d	Yes	No	No
Protogalaxy [8]	d	Yes	Yes	No
KiloNova	d	Yes	Yes	Yes

is also post-quantum secure.[2]

This new construction yields a zero-knowledge and non-uniform PCD as KiloNova. Furthermore, we propose an optimization technique to solve a potential efficiency problem existing in the non-uniform PCD: the increasing cost of dealing with the non-uniform, non-sparse circuits. Our technique delegates the computation of structure folds (folding atomic CCS matrices) to a more powerful third party that runs an extra IVC in parallel, thereby reducing the proving and communication costs for nodes in the PCD.

1.2. Performance Evaluation

1.2.1. Evaluations on Folding Schemes. We first compare features of our proposed generic folding scheme with other existing schemes in Table 1. Our scheme proves in the same CCS language as HyperNova [14] to supports high-degree constraints. Other expressive schemes, such as Protostar [15] and Protogalaxy [8], apply special sound protocols (SPS) to support high-degree constraints. In addition, our scheme efficiently supports both non-uniform circuits and multi-folding, which is only provided by Protogalaxy [8] currently. Although a recent work called UniPlonk [24] uniformizes the verifier's work in Plonk to allow Halo2 [25] and Nova [13] supporting non-uniform circuits, it is unknown whether it can be applied to systems based on multi-folding schemes. The last column indicates whether zero knowledge for IVC/PCD is achieved. Schemes that consider only one prover and apply zkSNARKs for proving the final IVC/PCD proofs, e.g., Nova [13], are not counted as zero-knowledge IVC/PCD. Therefore, BCLMS21 is the only scheme that provides zero knowledge among the schemes we compared. However, it does not support high-degree constraints and multi-folding.

For the theoretical performance of our generic folding scheme, we only compare it with state-of-the-art schemes, including HyperNova [14] and Protostar [15] as shown in Table 2. Note that the performance of our solution is close to HyperNova. For degree d CCS instances with $m \times n$ circuit matrices, the prover needs to compute a multi-scalar multiplication with $|\mathbf{w}|$ $\mathbb{G}$ operations, where $|\mathbf{w}|$ denotes the number of non-zero elements in the witness. For the

2. Note that our new theorem does not conflict with [11] because we actually utilize the zero-knowledge folding scheme to build a zero-knowledge NARK.

TABLE 2: Performance comparison between different folding/accumulation schemes

Schemes	Prover cost	Verifier cost
KiloNova	$\lvert\mathbf{w}\rvert$ $\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d \log^2 d)$ $\mathbb{F}$	$\log m + \log n$ RO 1 $\mathbb{G}$, $O(d \log m)$ $\mathbb{F}$
HyperNova [14]	$\lvert\mathbf{w}\rvert$ $\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d \log^2 d)$ $\mathbb{F}$	$\log m$ RO 1 $\mathbb{G}$, $O(d \log m)$ $\mathbb{F}$
Protostar [15]	$\lvert\mathbf{w}\rvert$ $\mathbb{G}$ $O(\lvert\mathbf{w}\rvert d \log^2 d)$ $\mathbb{F}$	$O(1)$ RO 3 $\mathbb{G}$, $(d + O(1))$ $\mathbb{F}$

verifier cost, our scheme performs $\log m + \log n$ times more random-oracle-like hashes than HyperNova due to the additional $\log n$ rounds in the second sum-check protocol. The performance of ProtoStar equals our computation, while its recursive overhead is minimal with only $O(1)$ hashes. Moreover, KiloNova has more appealing features than the other two schemes, as we mentioned in Table 1. If we consider constructing a non-uniform PCD that folds multiple instances with non-uniform structures, it is impractical to apply HyperNova and Protostar because their performance downgrades significantly.

1.2.2. Evaluations on PCD. BCMS20 [11] introduces the first PCD constructing based on accumulation schemes for SNARKs, such as Sonic [28] and Plonk [29]. This scheme utilizes the "additive" property of polynomial commitments as defined in [10] to achieve batch verification. To further reduce the recursion overhead, BCLMS21 [12] introduces split accumulation schemes to accumulate only the statement part of evaluation proofs under R1CS. Most of folding schemes improved on Nova [13] can only be used for IVC rather than PCD. For example, SuperNova [26] realizes a non-uniform IVC by enhancing Nova [13] with a selector for the list of predefined functions (i.e., instructions). Protostar [15] introduces a more expressive folding/accumulation scheme for special sound protocols supporting Plonkish relations, which yields a non-uniform IVC. Unfortunately, it suffers from the "cross terms" problem when building PCD as mentioned. Protogalaxy [8] reduces the number of cross terms in Protostar by leveraging the property of the Lagrange base, thus making the recursion overhead tolerable for multi-instance situations. Next, we give a detailed comparison between KiloNova and existing PCDs.

BCLMS21. Bünz et al. introduce PCD in BCLMS21 [12] from the split accumulation scheme that accumulates the proof of a NARK for uniform R1CS relations. The verification cost of the obtained PCD is relatively high, requiring 10 multi-scalar multiplication (MSM) of size m, while KiloNova only requires 1 MSM. In terms of folding scheme prover cost and recursive overhead, BCLM21 needs to handle $O(r)$ group operations with a larger coefficient than our scheme. However, it avoids logarithmic random oracle queries. Notably, BCLM21 does not support d-degree circuits and lookup operations.

Protogalaxy. As the following-up work of Protostar [15], Protogalaxy [8] reduces the cross terms from $O(d^s)$ to $O(ds)$ when folding s non-uniform instances and requires only $O(1)$ random oracle queries. Though Protogalaxy seems to be a promising primitive for constructing non-uniform PCD, it faces challenges in concrete implementations. First, it needs to handle cross items with Lagrange bases, yielding a *quasi-linear* prover cost in the folding scheme with instance number s and degree d. While ours is *linear*[3]. Besides, its prover cost is still constantly higher than ours because of computing extra s evaluations on $\widetilde{eq}(\cdot)$. Second, Protogalaxy does not support circuit aggregation as we proposed in Section 5.2 since their folded instances are still non-linear. Lastly, the authors of Protogalaxy neither provide constructions for non-uniform PCD nor add zero knowledge, whereas KiloNova presents explicit descriptions and solves the potential technical problems.

Concurrent work. In a paper concurrent with this work, Zhou et al. also construct a PCD with a multi-folding scheme extended on HyperNova. However, their work seems to be a complementary work for HyperNova[4] and thus lack theoretical contributions. Compared to KiloNova, their PCD supports neither non-uniform circuits nor zero knowledge.

2. Preliminaries

2.1. Notations

In this paper, we use λ to denote the security parameter. Accordingly, $\mathrm{negl}(\lambda)$ denotes an unspecified function that is negligible in λ. We denote by $[n]$ the set $\{1, ..., n\} \subseteq \mathbb{N}$. Let $\mathbb{F}$ denote a finite field, e.g., $\mathbb{F}_p$ is a prime field for a large prime p. The bold-type lower-case letters denote vectors, e.g., $\boldsymbol{a} \in \mathbb{F}^n$ is a vector of elements $a_1, ..., a_n \in \mathbb{F}$. $\boldsymbol{a}[i]$ is also used to denote the i-th element of $\boldsymbol{a}$ when the element is not specified with a concrete value. To represent a set, we use $\{a_i\}_{i=1}^n$ as a short-hand for $\{a_1, ..., a_n\}$. From here on, we let $B_\mu := B_\mu \subseteq \mathbb{F}^\mu$ be the boolean hypercube. For a finite set S, let $x \leftarrow\!\!\$\ S$ denote sampling x from S uniformly at random. We use "PPT algorithms" to refer to "Probabilistic Polynomial Time Algorithms".

2.2. Definitions for Polynomials

We recall some basic definitions for polynomials from [17] as follows. Let $f(\cdot) : \mathbb{F}^n \rightarrow \mathbb{F}$ be a *multivariate polynomial* with n input elements over $\mathbb{F}$, its total degree d is defined as the maximum degree over all monomials in $f(\cdot)$. Moreover, the degree of a polynomial in a specified variable x_i is the maximum exponent that x_i takes in any of the monomials in $f(\cdot)$. Particularly, a multivariate polynomial is a *multilinear* polynomial if the degree of the polynomial in each variable is at most one. To keep consistent with our notation of vectors, we use $f(\boldsymbol{x})$ to denote the polynomial $f(\cdot)$ with the specified input variable as vector $\boldsymbol{x}$. Next, we state the lemmas used in our paper.

3. To amend this problem, Protogalaxy proposes an alternative construction based on sum-check by replacing Lagrange bases with $\widetilde{eq}(\cdot)$. However, the sum-check protocol increases the number of RO queries to $O(\log N)$.

4. In fact, HyperNova already claims to realize multi-folding schemes in the paper.

Lemma 1 (Multilinear extensions [30])**.** *Let $f(\cdot) : B_n \rightarrow \mathbb{F}$ be a function that maps n-bit elements into an element of $\mathbb{F}$. The multilinear extension of $f(\cdot)$ is a unique multilinear n-variate polynomial $\tilde{f}(\cdot) : \mathbb{F}^n \rightarrow \mathbb{F}$ such that $\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x})$ for all $\boldsymbol{x} \in B_n$, which can be computed as follows.*

$$\tilde{f}(\boldsymbol{x}) = \sum_{\boldsymbol{e} \in B_n} f(\boldsymbol{e}) \cdot \widetilde{eq}(\boldsymbol{x}, \boldsymbol{e}),$$

where $\widetilde{eq}(\boldsymbol{x}, \boldsymbol{e}) = \prod_{i=1}^n (x_i \cdot e_i + (1 - x_i) \cdot (1 - e_i))$.

Lemma 2 (Schwartz-Zippel lemma [31])**.** *Assume $f(\cdot) : \mathbb{F}^n \rightarrow \mathbb{F}$ is a non-zero n-variate polynomial of degree at most d. Then on any finite set $S \subseteq \mathbb{F}$,*

$$\Pr_{\boldsymbol{x} \leftarrow\$ S^n}[f(\boldsymbol{x} = 0) \leq d/|S|],$$

where $\boldsymbol{x}$ is a randomly sampled vector from S^n and $|S|$ denotes the size of S.

2.3. Sum-check Protocol

The sum-check protocol is an interactive proof proposed by Lund et al. [32]. It has long attracted the attention of practitioners for its desirable performance, especially in a recent study on proof systems with linear proving time [17], [33]. Here, we only briefly review it. More technical details can be referred to [17].

Assume $f(\cdot) : \mathbb{F}^n \rightarrow \mathbb{F}$ as an n-variate low-degree polynomial with the maximum degree of d for each variable. The prover wants to convince the verifier of the following claim:

$$\mathrm{sum} = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} f(x_1, ..., x_n). \qquad (1)$$

To conduct this protocol with logarithmic verifier cost, the verifier chooses a random vector $\boldsymbol{r} \in \mathbb{F}^n$ as the challenges for the n-round interactions with the prover. At the final step, the verifier outputs a claim about the evaluation $f(\boldsymbol{r})$, i.e., $c \leftarrow \Pi_{\mathsf{sc}}(f, n, d, \mathrm{sum}, \boldsymbol{r})$. If $c = f(\boldsymbol{r})$ holds, then the verifier is convinced of the claim about the sum of $f(\cdot)$ in Equation (1). According to previous work [32], [17], the sum-check protocol satisfies both completeness and soundness properties, and its communication cost takes $O(n \cdot d)$ element of $\mathbb{F}$.

2.4. Proof-Carrying Data

In this paper, we adopt the definition of PCD from [12], [27]. We start with defining some necessary terminologies before presenting the definition.

Definition 1. *A* transcript T *is a directed acyclic graph with each vertex $u \in V(\mathsf{T})$ labeled by local data $z_{\mathsf{loc}}^{(u)}$ and each edge $e \in E(\mathsf{T})$ labeled by a message $z^{(e)} \neq \perp$. The output $o(\mathsf{T})$ of a transcript T is a message $z^{(e)}$ where $e = (u, v)$ is the lexicographically-first edge such that v is a sink.*

Definition 2. *A vertex $u \in V(\mathsf{T})$ is φ-*compliant *for $\varphi \in \mathsf{F}$ if for all outgoing edges $e = (u, v) \in E(\mathsf{T})$:*

- *(base case) if u has no incoming edges, $\varphi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, \perp, ..., \perp)$ accepts,*
- *(recursive case) if u has incoming edges $e_1, ..., e_m$, $\varphi(z^{(e)}, z_{\mathsf{loc}^{(u)}}, z^{(e_1)}, ..., z^{(e_m)})$ accepts.*

We say that T is φ-compliant if all of its vertices are φ-compliant.

Definition 3 (Proof-Carrying Data[27])**.** *A proof-carrying data scheme for a class of compliance predicates F is a tuple of algorithms $\mathsf{PCD} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ where*

- *$\mathcal{G}(1^\lambda) \rightarrow \mathsf{pp}$ on input security parameter λ, samples and outputs public parameter pp.*
- *$\mathcal{K}(\mathsf{pp}, \varphi) \rightarrow (\mathsf{pk}, \mathsf{vk})$ on input public parameter pp and a compliance predicate $\varphi \in \mathsf{F}$, outputs a prover key pk and a verifier key vk.*
- *$\mathcal{P}(\mathsf{pk}, z, z_{\mathsf{loc}}, \{z_i, \Pi_i\}_{i=1}^r) \rightarrow \Pi$ on input public key pk, message z of an outgoing edge, local data z_{loc}, messages $\{z_i\}_{i\in[r]}$ of incoming edges and their corresponding proofs $\{\Pi_i\}_{i\in[r]}$, outputs a new proof Π to attest the correctness of z.*
- *$\mathcal{V}(\mathsf{vk}, z, \Pi) \rightarrow 0/1$ on input verifier key vk, message z and proof Π, outputs $0/1$ to reject or accept.*

A proof-carrying data scheme PCD should satisfy the perfect completeness, knowledge soundness, and zero-knowledge properties described in Appendix A.2.

2.5. Customizable Constraint Systems

The customizable constraint system (CCS) is an intermediate representation of arithmetic circuits introduced by Setty et al. [21], which can simultaneously generalize R1CS, Plonkish, and AIR without overheads. Similar to previous work, we build our folding scheme based on a committed version of this constraint system as follows.

For ease of exposition, a committed CCS relation $\mathcal{R}$ should be defined over public parameters, structure, instance, and witness tuples. Specifically, a structure $\mathcal{S}$ describes constraints and an "instance" consisting of the public input and output where a "witness" should satisfy.

Definition 4 (Committed CCS)**.** *Let $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ denote an additively-homomorphic polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$. Denote the public parameters of size bounds as $m, n, N, l, t, q, d \in \mathbb{N}$ where $n = 2\cdot(l+1)$ and $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, s_y)$. The committed customizable constraint system (CCCS) relation $\mathcal{R}_{\mathsf{CCCS}}$ is defined as follows.*

- *A $\mathcal{R}_{\mathsf{CCCS}}$ structure $\mathcal{S}$ consists of:*
  - *a sequence of sparse multilinear polynomials in $s_x + s_y$ variables $\{\widetilde{M}_j\}_{j\in[t]}$ such that they evaluate to a non-zero value in at most $N = \Omega(m)$ locations over the Boolean hypercube $B_{s_x} \times B_{s_y}$.*
  - *a sequence of q multisets $\{S_i\}_{i\in[q]}$, where an element in each multiset is from the domain $\{1, ..., t\}$ and the cardinality of each multiset is at most d.*
  - *a sequence of q constants $\{c_i\}_{i\in[q]}$, where each constant is from $\mathbb{F}$.*
- *A $\mathcal{R}_{\mathsf{CCCS}}$ instance is $(C, \mathbf{io})$ where C is a commitment to a multilinear polynomial in $s_y - 1$ variables and $\mathbf{io} \in \mathbb{F}^l$.*
- *A $\mathcal{R}_{\mathsf{CCCS}}$ witness consists of a multilinear polynomial $\widetilde{\mathbf{w}}$ in $s_y - 1$ variables.*

A $\mathcal{R}_{\mathsf{CCCS}}$ instance with structure $\mathcal{S}$ is satisfied by a $\mathcal{R}_{\mathsf{CCCS}}$ witness if $\mathsf{Commit}(\mathsf{pp}, \widetilde{\mathbf{w}}) = C$ and if for all $\boldsymbol{x} \in B_{s_x}$,

$$\sum_{i\in[q]} c_i \left(\prod_{j\in S_i} \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}) \right) \right) = 0, \quad (2)$$

where $\tilde{z}(\boldsymbol{y})$ is an s_y-variate multilinear polynomial such that $\tilde{z}(\boldsymbol{y}) = (\mathbf{w}, 1, \mathbf{io})$ for all $\boldsymbol{y} \in B_{s_y}$.

Detailed description of CCS is given in Appendix A.3. We also denote a NARK system for committed CCS relations as $\mathsf{NARK_CCS}$, in which the prover takes public input and directly outputs a committed CCS instance as the NARK proof. Similarly, the verifier takes the proof and check its validity according to Definition 4. Since this construction is trivial, we do not explicitly define it in this paper.

3. Building Blocks

In this section, we describe several essential building blocks for the design of KiloNova. First, we extend the multi-folding scheme introduced in HyperNova [14] to support non-uniform circuit scenarios. The new model is called a *generic folding scheme*. Next, we derive a new "relaxed" relation called *atomic CCS relation* from the SuperSpartan protocol to remove the cross terms generated in the generic folding schemes.

3.1. Generic Folding Schemes

Recall that a folding scheme [KST22] for a relation $\mathcal{R}$ is a protocol between a prover and a verifier that reduces the task of checking two instances in $\mathcal{R}$ with the *same* structure $\mathcal{S}$ into the task of checking a single folded instance in $\mathcal{R}$ also with structure $\mathcal{S}$. Then in HyperNova, the authors introduce a generalization of folding schemes as multi-folding schemes, which can fold two collections of instances in relations $\mathcal{R}^{(1)}$ and $\mathcal{R}^{(2)}$ with the *same* structure $\mathcal{S}$ respectively. This paper extends the multi-folding scheme to allow it to fold relations with *different* structures. Concretely, a *generic folding scheme* is defined with respect to a set of relations $\{\mathcal{R}^{(i)}\}_{i=1}^{\ell}$ with different structures $\{\mathcal{S}^{(i)}\}_{i=1}^{\ell}$ and size parameters $\{s^{(i)}\}_{i=1}^{\ell}$ (the number of repetition for each $\mathcal{S}^{(i)}$). It is an interactive protocol between a prover and a verifier that reduces the task of checking a collection of $s^{(i)}$ instances in $\mathcal{R}^{(i)}$ for all $i \in [\ell]$ ($\sum_{i\in[\ell]} s^{(i)}$ instances in total) into checking a single folded instance in $\mathcal{R}^*$ with structure $\mathcal{S}^*$. We formally define it below.

Definition 5 (Generic folding schemes)**.** *Consider relations $\{\mathcal{R}^{(i)}\}_{i=1}^{\ell}$ over public parameters, structures, instance, and witness tuples such that each $\mathcal{R}^{(i)}$ has distinct structure $\mathcal{S}^{(i)}$. A generic folding scheme for $\{(\mathcal{R}^{(i)}, s^{(i)})\}_{i=1}^{\ell}$ consists of a PPT generator algorithm $\mathcal{G}$, a deterministic encoder* *algorithm $\mathcal{K}$, and a pair of PPT algorithms $\mathcal{P}$ and $\mathcal{V}$ denoting the prover and the verifier respectively, with the following interface:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: on input security parameter λ, samples public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, \{\mathcal{S}^{(i)}\}_{i=1}^{\ell}) \to (\mathsf{pk}, \mathsf{vk})$*: on input* pp*, and common structures $\{\mathcal{S}^{(i)}\}_{i=1}^{\ell}$ among the instances to be folded, outputs a prover key* pk *and a verifier key* vk.
- $\mathcal{P}(\mathsf{pk}, \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}\}_{i=1}^{\ell}) \to (\mathcal{S}^*, \mathbf{u}^*, \mathbf{w}^*)$*: on input ℓ vectors of instances $\{\mathbf{u}^{(i)}\}_{i=1}^{\ell}$, where each vector $\mathbf{u}^{(i)}$ is in $\mathcal{R}^{(i)}$ with a distinct structure $\mathcal{S}^{(i)}$, and corresponding vector of witnesses $\mathbf{w}^{(i)}$ for $i \in [\ell]$, outputs a folded instance-witness pair $(\mathbf{u}^*, \mathbf{w}^*)$ in a new relations $\mathcal{R}^*$ with structure $\mathcal{S}^*$.*
- $\mathcal{V}(\mathsf{vk}, \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell}) \to (\mathcal{S}^*, \mathbf{u}^*)$*: on input ℓ vectors of instances $\{\mathbf{u}^{(i)}\}_{i=1}^{\ell}$, outputs a folded instance $\mathbf{u}^*$ in a new relations $\mathcal{R}^*$ with structure $\mathcal{S}^*$.*

Let Π_{fold} denote the interaction between $\mathcal{P}$ and $\mathcal{V}$. Then Π_{fold} is a function that takes as input $((\mathsf{pk}, \mathsf{vk}), \{(\mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)})\}_{i=1}^{\ell})$ and runs the interaction on prover input $(\mathsf{pk}, \{(\mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)})\}_{i=1}^{\ell})$ and verifier input $(\mathsf{vk}, \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell})$. At the end of interaction Π_{fold} outputs $(\mathbf{u}^*, \mathbf{w}^*)$ where $\mathbf{u}^*$ is the verifier's output folded instance, and $\mathbf{w}^*$ is the prover's output folded witness.

Accordingly, a generic folding scheme should satisfy completeness and knowledge soundness in the random oracle model. We present the security definitions in Appendix A.4. We also present the definition of zero knowledge.

3.2. Atomic CCS Relations

In this part, we first introduce a relaxed CCS relation called *atomic CCS* that is amenable to constructing generic folding schemes. Different from the committed CCS relations or linearized committed CCS in [14], this new variant is satisfied with linear constraints on structures (matrices) and instance-witness pairs, respectively. Therefore, folding multiple atomic CCS instances under different matrices does not produce any cross terms.

Generally speaking, most of the current solutions reduce the cross terms with some "makeup" measures after the folding schemes, which adds extra cost for the prover and verifier sides. While we believe a more practical way is to take measures before the folding schemes to avoid the generation of cross terms. Inspired by this idea, we extend the approach in HyperNova[14], which linearizes the high-degree CCS relation by running an "early stopping" version of SuperSpartan. To state it clearly, we need to first give a review of SuperSpartan. On input as a committed CCS instance, the prover and verifier in SuperSpartan rewrite it into a sum-check statement as:

$$\sum_{\boldsymbol{x} \in B_{s_x}} \widetilde{eq}(\boldsymbol{\alpha}, \boldsymbol{x}) \cdot \sum_{i \in [q]} c_i \left(\prod_{j \in S_i} \left(\sum_{\boldsymbol{y} \in B_{s_y}} \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}) \right) \right) = 0$$

where $\boldsymbol{\alpha}$ is a randomly sampled vector from $\mathbb{F}^{s_x}$ and $\widetilde{eq}(\boldsymbol{\alpha}, \boldsymbol{x})$ is added to reduce the s_x constraints in $\mathcal{R}_{\mathsf{CCCS}}$ to one sum constraint above. To check Equation (3.2), the prover and verifier run two rounds of sum-check protocols recursively on $\boldsymbol{x}$ and $\boldsymbol{y}$. After the first round of the sum-check protocol, the verifier can check the final evaluation on $\boldsymbol{r}_x$ with the following claims given by the prover:

$$\sum_{\boldsymbol{y} \in B_{s_y}} \widetilde{M}_j(\boldsymbol{r}_x, \boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}) = \sigma_j, \ \forall j \in [t]. \tag{3}$$

where $\boldsymbol{r}_x$ is the challenge vector generated among the protocol. These claims can be further checked by running t sum-check protocols in parallel, and the prover sends $t+1$ claims for the verifier to check the final evaluation:

$$\widetilde{M}_j(\boldsymbol{r}_x, \boldsymbol{r}_y) = \theta_j, \ \forall j \in [t] \text{ and } \tilde{z}(\boldsymbol{r}_y) = \epsilon. \tag{4}$$

Note that Equation (3) only contains polynomial $\tilde{z}(\boldsymbol{y})$ with degree of 1. For two instances with same structures $\{\widetilde{M}_j\}_{j=1}^{t}$, folding their claims in Equation (3) produces no cross terms. Therefore, HyperNova formalizes these claims as a restricted form of CCS, i.e., linearized committed CCS. To prevent cross terms in generic folding schemes, folding with the above linearized committed CCS instances is not sufficient. We further run the second round of sum-check protocol and obtain Equation (4). Note that Equation (4) contains claims on $\widetilde{M}_j(\boldsymbol{r}_x, \boldsymbol{r}_y), j \in [t]$ and $\tilde{z}(\boldsymbol{r}_y)$ separately with degree at most 1. As a result, folding these claims produces no cross terms, even for instances with different structures. We denote this new variant of CCS as Atomic CCS and formalize it in the following definition.

Definition 6 (Atomic CCS)**.** *Let* $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *denote an additively-homomorphic polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$. Denote the public parameters of size bounds as $m, n, N, l, t \in \mathbb{N}$ where $n = 2 \cdot (l+1)$ and* $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, s_y - 1)$. *The atomic customizable constraint system (ACCS) relation $\mathcal{R}_{\mathsf{ACCS}}$ is defined as follows.*

- *An $\mathcal{R}_{\mathsf{ACCS}}$ structure $\mathcal{S}$ consists of a sequence of sparse multilinear polynomials in $s_x + s_y$ variables $\{\widetilde{M}_j\}_{j \in [t]}$ such that they evaluate to a non-zero value in at most $N = \Omega(m)$ locations over the boolean hypercube $B_{s_x} \times B_{s_y}$.*
- *An $\mathcal{R}_{\mathsf{ACCS}}$ instance is $(C, v_0, \mathbf{io}, \boldsymbol{r}_x, \boldsymbol{r}_y, v_1, ..., v_t, v_z)$ where $v_0 \in \mathbb{F}, \mathbf{io} \in \mathbb{F}^l, \boldsymbol{r}_x \in \mathbb{F}^{s_x}, \boldsymbol{r}_y \in \mathbb{F}^{s_y}, v_z \in \mathbb{F}, v_j \in \mathbb{F}$ for all $j \in [t]$, and C is a commitment to a multilinear polynomial in $s_y - 1$ variables. An $\mathcal{R}_{\mathsf{ACCS}}$ witness consists of a multilinear polynomial $\widetilde{\mathbf{w}}$ in $s_y - 1$ variables.*

An $\mathcal{R}_{\mathsf{ACCS}}$ instance (structure-instance tuple) is satisfied by a witness if $\mathsf{Commit}(\mathsf{pp}, \widetilde{\mathbf{w}}) = C$, *$v_z = \tilde{z}(\boldsymbol{r}_y)$ and if for all $j \in [t]$, the equation $v_j = \widetilde{M}_j(\boldsymbol{r}_x, \boldsymbol{r}_y)$ holds, where $\widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y})$ is an $(s_x + s_y)$-variate multilinear polynomial, $\tilde{z}(\boldsymbol{y})$ is an s_y-variate multilinear polynomial such that $\tilde{z}(\boldsymbol{y}) = \widetilde{(\mathbf{w}, v_0, \mathbf{io})}$ for all $\boldsymbol{y} \in B_{s_y}$.*

4. Generic Folding Scheme for CCS

4.1. High-level Ideas

This section describes a generic folding scheme for CCS. Its aim is to fold multiple input instances into one ACCS instance. Generally speaking, for each CCCS instance as input, the prover runs the "early stopping" SuperSpartan for it with two rounds of sum-check protocols. This process can be formalized as a so-called special-sound protocol (a class of interactive proofs with trivial security proofs for soundness) following the definition in Protostar [15]. We present the step-by-step protocol Π_{CCCS} in Appendix B.1. However, naively running an independent special-sound protocol for each input instance is expensive for both the prover and verifier. A prior idea [17] introduces a technique to combine them into only one protocol with random linear combinations. As depicted in Figure 2, the prover aggregates polynomials $\{f^{(i)}(\boldsymbol{x})\}_{i=1}^{n}$ derived from the input CCCS instances with a challenge value γ given by the verifier. Then the prover runs a single sum-check protocol instead of n for the aggregated polynomial $f(\boldsymbol{x})$. Similarly, the prover runs the second sum-check on another dimension of variables $\boldsymbol{y}$ for the aggregated polynomial $g(\boldsymbol{x})$. Finally, the prover sends the claim to the evaluations of polynomials $\{\widetilde{M}_j^{(i)}\}_{j\in[t]}$ and $\tilde{z}^{(i)}$ for each i-th instance, which can be formalized and folded as an ACCS instance.

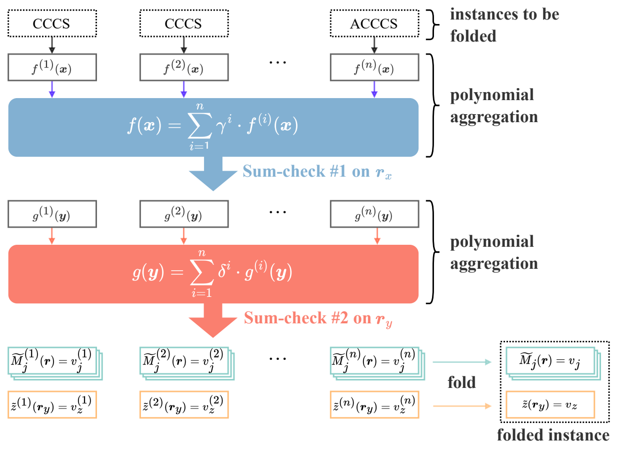
CCCS
CCCS
ACCCS
instances to be folded
$f^{(1)}(\boldsymbol{x})$
$f^{(2)}(\boldsymbol{x})$
$f^{(n)}(\boldsymbol{x})$
$f(\boldsymbol{x}) = \sum_{i=1}^{n} \gamma^i \cdot f^{(i)}(\boldsymbol{x})$
polynomial aggregation
Sum-check #1 on $\boldsymbol{r}_x$
$g^{(1)}(\boldsymbol{y})$
$g^{(2)}(\boldsymbol{y})$
$g^{(n)}(\boldsymbol{y})$
$g(\boldsymbol{y}) = \sum_{i=1}^{n} \delta^i \cdot g^{(i)}(\boldsymbol{y})$
polynomial aggregation
Sum-check #2 on $\boldsymbol{r}_y$
$\widetilde{M}_j^{(1)}(\boldsymbol{r}) = v_j^{(1)}$
$\widetilde{M}_j^{(2)}(\boldsymbol{r}) = v_j^{(2)}$
$\widetilde{M}_j^{(n)}(\boldsymbol{r}) = v_j^{(n)}$
$\tilde{z}^{(1)}(\boldsymbol{r_y}) = v_z^{(1)}$
$\tilde{z}^{(2)}(\boldsymbol{r_y}) = v_z^{(2)}$
$\tilde{z}^{(n)}(\boldsymbol{r_y}) = v_z^{(n)}$
fold
$\widetilde{M}_j(\boldsymbol{r}) = v_j$
$\tilde{z}(\boldsymbol{r_y}) = v_z$
folded instance

Figure 2: Workflow of Generic Folding Scheme. Take totally n committed CCS or atomic CCS instances as inputs, the prover and verifier encode them into one aggregated polynomial $f(\boldsymbol{x})$ and run the "early stopping" SuperSpartan to obtain the n atomic CCS instances, which can be folded into one without cross terms.

Given the special-sound protocol for CCCS and the combination technique above, we are still not ready to build a generic folding scheme. Note that in an IVC/PCD, the prover also needs to deal with the folded ACCS instance received from the previous step. The problem is that the prover can not directly fold the ACCS instance at the previous step with the ACCS instance generated at this step. This is because these two instances use *different* challenges $\boldsymbol{r}_x, \boldsymbol{r}_y$, therefore do not have homomorphic property. To amend this, we design another special-sound protocol to "update" the random vectors of ACCS instances for new ones in the following subsection.

4.2. Special Sound Protocols for Atomic CCS

Based on the observations above, we build a special sound protocol Π_{ACCS} for the generic folding schemes. Before we introduce the detailed protocol, a simple example is presented to illustrate a technique we used. Assuming a claim (constraint) $f(\boldsymbol{r}_x) = v$, the prover wants to update it on a new vector $\boldsymbol{r}'_x$. To achieve this, the prover first writes a polynomial as $g(\boldsymbol{x}) = \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{x}) \cdot f(\boldsymbol{x})$, and then engages in a sum-check protocol with the verifier to show $\sum_{\boldsymbol{x}\in B_{s_x}} g(\boldsymbol{x}) = v$ with randomness $\boldsymbol{r}'_x$. This equation holds because the sum of $g(\boldsymbol{x})$ can be regarded as an MLE of $f(\cdot)$ as $\tilde{f}(\boldsymbol{e}) = \sum_{\boldsymbol{x}\in B_{s_x}} \widetilde{eq}(\boldsymbol{e}, \boldsymbol{x}) \cdot f(\boldsymbol{x})$ according to Lemma 1. By evaluating $\tilde{f}(\boldsymbol{e})$ on $\boldsymbol{e} = \boldsymbol{r}_x$, we obtain

$$v = \tilde{f}(\boldsymbol{r}_x) = \sum_{\boldsymbol{x}\in B_{s_x}} \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{x}) \cdot f(\boldsymbol{x}) = \sum_{\boldsymbol{x}\in B_{s_x}} g(\boldsymbol{x}).$$

As a result, the prover produces a new claim as $g(\boldsymbol{r}'_x) = v'$ with updated $\boldsymbol{r}'_x$. And the evaluation of $f(\boldsymbol{r}'_x)$ can be computed as v'/e, where $e = \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{r}'_y)$. The validity of the original claim can be guaranteed by the soundness of the sum-check protocol.

Inspired by this technique, we design the special-sound protocol Π_{ACCS}. Concretely, this protocol allows the prover and verifier to run a series of interactions to update the random vectors $\boldsymbol{r}_x, \boldsymbol{r}_y$ in the atomic CCS instance. First, the prover rewrites each $\widetilde{M}_j(\boldsymbol{r}_x, \boldsymbol{r}_y), j \in [t]$ as

$$M_j(\boldsymbol{x}) = \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{x}) \cdot \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y}) \right)$$

Then the prover and verifier run the first sum-check protocol on $\boldsymbol{x}$ for each $M_j(\boldsymbol{x})$. With the random challenge γ sampled by the verifier at step 1, the prover constructs an aggregated polynomial $f(\boldsymbol{x})$ as the linear combinations of each $M_j(\boldsymbol{x})$. Then the prover and verifier run the sum-check#1 on $f(\boldsymbol{x})$ at step 4 with the random vector $\boldsymbol{r}'_x$, where the sum_x equals to $\sum_{j\in[t]} \gamma^j \cdot v_j$. If the sum-check#1 is correctly executed, the claims on matrices are updated to

$$\sigma_j = \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{y})$$

for $j \in [t]$ with the new random vector $\boldsymbol{r}'_x$ at step 5. Next, the prover rewrites for $\tilde{z}(\boldsymbol{r}_y)$ and each $\widetilde{M}_j(\boldsymbol{r}_x, \boldsymbol{r}_y)$ as

$$M_j(\boldsymbol{y}) = \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{y}), \quad z(\boldsymbol{y}) = \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y})$$

the prover and verifier run the sum-check#2 on the aggregated $g(\boldsymbol{y})$ at step 9 with the random vector $\boldsymbol{r}'_y$, where $\mathrm{sum}_y = \sum_{j\in[t]} \delta^j \cdot \sigma_j + \delta^{t+1} \cdot v_z$. The remaining process runs similarly. Finally, the protocol updates the original atomic CCS into a new one on $\boldsymbol{r}'_x, \boldsymbol{r}'_y$.

Note that this special-sound protocol has almost the same form as the Π_{CCCS} for committed CCS. Therefore, a combination technique can also be applied, the combined protocol is presented in Appendix B.2 with security proofs in the full-version paper. As a result, the prover is able to fold multiple committed and atomic CCS instances into one atomic CCS instance with an efficient protocol dominated by only two sum-check protocols[5]. The interactive generic folding scheme is not explicitly given since it will be covered in the final version as zero-knowledge non-interactive generic folding scheme in Section 4.4. Next, we explain how to implement zero knowledge for our generic folding scheme.

Special Sound Protocol $\Pi_{\mathsf{ACCS}} = (\mathcal{P}, \mathcal{V})$ for $\mathcal{R}_{\mathsf{ACCS}}$

1. $\mathcal{V}$: Sample $\gamma \leftarrow\!\$\, \mathbb{F}$, and send to $\mathcal{P}$.
2. $\mathcal{V}$: Sample $\boldsymbol{r}_x' \leftarrow\!\$\, \mathbb{F}^{s_x}$.
3. $\mathcal{P}$: Compute $\tilde{z}(\boldsymbol{y}) = (\widetilde{\mathbf{w}}, 1, \mathbf{io})$.
4. **Sum-check#1.** $c_x \leftarrow \Pi_{\mathsf{sc}}(f, s_x, 2, \mathrm{sum}_x)$ with random $\boldsymbol{r}_x'$ where: $f(\boldsymbol{x}) =$
$$\textstyle\sum_{j\in[t]} \gamma^j \cdot \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{x}) \cdot \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y})\right).$$
5. $\mathcal{P}$: Compute $\{\sigma_j\}_{j\in[t]}$ and send to $\mathcal{V}$, where:
$$\sigma_j = \textstyle\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{r}_x', \boldsymbol{y}), \text{ for all } j \in [t].$$
6. $\mathcal{V}$: Compute $e_1 \leftarrow \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{r}_x')$, and abort if:
$$c_x \neq e_1 \cdot \textstyle\sum_{j\in[t]} \gamma^j \cdot \sigma_j.$$
7. $\mathcal{V}$: Sample $\delta \leftarrow\!\$\, \mathbb{F}$, and send to $\mathcal{P}$.
8. $\mathcal{V}$: Sample $\boldsymbol{r}_y' \leftarrow\!\$\, \mathbb{F}^{s_y}$.
9. **Sum-check#2.** $c_y \leftarrow \Pi_{\mathsf{sc}}(g, s_y, 2, \mathrm{sum}_y)$ with random $\boldsymbol{r}_y'$ where: $g(\boldsymbol{y}) =$
$$\widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \left(\textstyle\sum_{j\in[t]} \delta^j \cdot \widetilde{M}_j(\boldsymbol{r}_x', \boldsymbol{y}) + \delta^{t+1} \cdot \tilde{z}(\boldsymbol{y})\right).$$
10. $\mathcal{P}$: Compute $\epsilon, \{\theta_j\}_{j\in[t]}$ and send to $\mathcal{V}$, where:
$$\epsilon = \tilde{z}(\boldsymbol{r}_y'),\ \theta_j = \widetilde{M}_j(\boldsymbol{r}_x', \boldsymbol{r}_y'), j \in [t].$$
11. $\mathcal{V}$: Compute $e_2 \leftarrow \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{r}_y')$, and abort if:
$$c_y \neq e_2 \cdot (\textstyle\sum_{j\in[t]} \delta^j \cdot \theta_j + \delta^{t+1} \cdot \epsilon).$$
12. $\mathcal{P}$: Open the witness $\widetilde{\mathbf{w}}$.
13. $\mathcal{V}$: Check that
$$(1)\ \mathsf{Commit}(\mathsf{pp}, \widetilde{\mathbf{w}}) = C,$$
$$(2)\ \epsilon = \tilde{z}(\boldsymbol{r}_y'),\ \theta_j = \widetilde{M}_j(\boldsymbol{r}_x', \boldsymbol{r}_y').$$

4.3. Adding Zero-knowledge

To achieve zero-knowledge for our folding schemes, a common idea is directly adding a masking value for the witness $\mathbf{w}$. That is, the prover runs the protocol with $\rho \cdot \mathbf{w} + \boldsymbol{w}$ instead of $\mathbf{w}$, where $\boldsymbol{w} \in \mathbb{F}^{n-l-1}$. Although this technique can prove the validity of folding without leaking any information about the witness, the prover needs to do the extra computation, especially for committed CCS instances with high degrees. We illustrate this point with the following example. Assume the prover wants to run the sum-check#1 on a committed CCS instance with a masking vector $\boldsymbol{w}$. The target polynomial is written as:

$$f(\boldsymbol{x}) = \widetilde{eq}(\boldsymbol{\alpha}, \boldsymbol{x}) \cdot \left(\sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \widetilde{Z}(\boldsymbol{y}) \right) \right),$$

where $\widetilde{Z}(\boldsymbol{y}) = (\rho \cdot \widetilde{\mathbf{w} + \boldsymbol{w}}, 1, \mathbf{io})$. Since the masking vector $\boldsymbol{w}$ is sampled randomly, the above target polynomial no longer equals zero. To ensure the equation holds, the prover has to compute the sum of the new polynomial $f(\boldsymbol{x})$, which takes $O(N + tm + qmd \log^2 d)$ $\mathbb{F}$-ops dominated by the computation of non-linear part with degree d. When dealing with multiple committed CCS instances, the prover must execute the above computation for each independently. Moreover, the prover has to send $O(2^d)$ cross terms for each instance to convince the verifier of the original relation on $\mathbf{w}$. To alleviate these problems, we propose a more efficient approach for achieving zero knowledge in three techniques.

(1) Zero knowledge for prover claims. We first consider shielding the witness $\mathbf{w}$ among the verification of the claims. The verifier is expected to learn no information about $\mathbf{w}$ except its validity to the CCS instance from steps 5, 6, and 10-12. As mentioned above, the non-linearity part of the CCS relation prevents us from directly masking the witness as $\rho \cdot \mathbf{w} + \boldsymbol{w}$. Here, we utilize an approach in [23] by Bootle et al. to randomize the claims, which will not introduce extra costs for non-linear parts. Generally speaking, the claims at steps 5 and 10 can be regarded as linear combinations of the values in $\mathbf{w}$. According to the result given by Bootle et al., if we pad the witness with as many non-zero random values as the number of combinations it receives, then all the responses will be uniformly random and leak no information.

Again, we take the committed CCS relation in Equation (5) as an example. With a randomly sampled vector $\boldsymbol{r} = [\boldsymbol{r}_j]_{j=1}^t, \boldsymbol{r}_j \in \mathbb{F}^2$ added to the vector $\boldsymbol{r}$, the equation can be rewritten as

$$\sum_{i\in[q]} c_i \cdot \bigcirc_{j\in S_i} \begin{bmatrix} M_j & O \\ O & I_j \end{bmatrix} \cdot (\boldsymbol{z}, \boldsymbol{r}) = (\mathbf{0}, \sum_{i\in[q]} c_i \cdot \bigcirc_{j\in S_i} \boldsymbol{r}_j),$$

where O denotes zero matrix, I_j is a $2 \times 2t$ matrix with an 2×2 identity matrix I in the j-th position, i.e.,

$$I_j = [\underbrace{O, ..., O}_{j-1}, I, O, ..., O].$$

So far, the zero knowledge of the committed CCS instance is retained against $2t$ queries of the linear combination of $\mathbf{w}$. To accommodate it to our folding scheme, we denote extra two sets of variables as $\boldsymbol{a} \in \{0, 1\}, \boldsymbol{b} \in B_{\log(2t)}$ for representing $I_j, \boldsymbol{r}_j$. The above equation can be further written into the

5. The instances are not required to have structure with same size bounds, we can use a simple padding scheme to ensure they are the same size.

multilinear polynomial form as follows[6]:

$$\sum_{i\in[q]} c_i \prod_{j\in S_i} \left(\sum_{\substack{\boldsymbol{y}\in B_{s_y}\\ \boldsymbol{b}\in B_{\log(2t)}}} (\widetilde{M}_j(\boldsymbol{x},\boldsymbol{y}) + \widetilde{I}_j(\boldsymbol{a},\boldsymbol{b}))(\widetilde{z}(\boldsymbol{y}) + \widetilde{r}(\boldsymbol{b})) \right)$$
$$= \sum_{\boldsymbol{b}\in B_{\log(2t)}} \left(\sum_{i\in[q]} c_i \cdot \bigcirc_{j\in S_i} \widetilde{r}_j(\boldsymbol{b}) \right),$$

for all $\boldsymbol{x} \in B_{s_x}, \boldsymbol{a} \in \{0,1\}$, where $\widetilde{I}_j(\boldsymbol{a},\boldsymbol{b})), \widetilde{r}_j(\boldsymbol{b})$ are the MLE's for I_j, r_j on $\boldsymbol{a}, \boldsymbol{b}$ and $\widetilde{r}(\boldsymbol{b}) = \sum_{j=1}^{t} \widetilde{r}_j(\boldsymbol{b})$.

(2) Zero knowledge for sum-check protocols. Preserving the privacy of $\mathbf{w}$ among the claims sent by the prover is insufficient to ensure zero knowledge of the whole folding scheme. Note that the sum-check protocols#1 and #2 at steps 4 and 9 are not shielded. Thus, the transcripts in the protocol also leak the information of $\mathbf{w}$. To amend this problem, we refer to a previous work as Libra [34], which presents a zero-knowledge sum-check protocol by masking the coefficients of the target polynomial $f(\boldsymbol{x})$ with a random polynomial $f_r(\boldsymbol{x})$ of size $O(d)$, where $s_x = |\boldsymbol{x}|$ is logarithmic of the size of target polynomial $f(\boldsymbol{x})$. Briefly, the random polynomial can be constructed as $f_r(\boldsymbol{x}) = r_0 + r_1(x_1) + r_2(x_2) + \cdots + r_{s_x}(x_{s_x})$, where $r_i(x_i) = r_{i,1}x_i + r_{i,2}x_i^2 + \cdots + r_{i,d}x_i^d$ is a random univariate polynomial of degree d. For simplicity, we denote the zero-knowledge sum-check protocol as $c \leftarrow \Pi_{\mathsf{zksc}}(f, n, d, \mathsf{sum}, \boldsymbol{r})$. The security is guaranteed accordingly by the results in [35].

(3) Zero knowledge for folded instances. The previous two techniques can guarantee zero knowledge of most processes in our generic folding scheme except the final folding operation. The prover outputs the folded witness $\mathbf{w} + \eta \cdot \mathbf{w}'$ without randomization as part of the proof, which leaks the information about witnesses. To amend this, the masking value $\boldsymbol{w} \in \mathbb{F}^{n-l-1}$ mentioned at the beginning has to be introduced. Differently, only one $\boldsymbol{w}$ is needed this time because the privacy in the previous steps is already preserved. Therefore, the prover takes the masking value as another committed CCS instance, i.e., a dummy instance, with empty structure $\mathcal{S}$, empty $\mathbf{io}$ and commitment $C = \mathsf{Commit}(\mathsf{pp}, \widetilde{\boldsymbol{w}})$, and runs the folding scheme accordingly, where O denote $m \times n$ zero matrix.

Besides, we also want to mention a trick for saving the computation of the sum of $\widetilde{\boldsymbol{w}}$. The polynomial $f(\boldsymbol{x})$ aggregates the masking value with all other instances. According to the proving algorithm for the sum-check protocol proposed in [36], it is handy to acquire the sum of $f(\boldsymbol{x})$ from the bookkeeping table. Thus, we can obtain the sum of $\widetilde{\boldsymbol{w}}$ by subtracting sums of other instances, i.e., $\sum_{j\in[t]} \gamma^j \cdot v_j \cdot v_z$ for atomic CCS instance and 0 for committed CCS instance, from $\sum_{\boldsymbol{x}\in B_{s_x}}$ without actually computing the concrete evaluations.

The final zero-knowledge generic folding schemes based on the above three techniques only require the prover to compute *one* additional polynomial sum for masking the folding instances in (3). Besides, there are no extra cross terms introduced in the scheme. The zero-knowledge property is proved in the full-version paper.

6. It is more efficient to write polynomials on $\boldsymbol{x} \in B_{\log(m+2)}, \boldsymbol{y} \in B_{\log(n+2t)}$, we omit this expression for simplicity. In fact, the extra variables may even be unnecessary in real-world implementation since most vector $\boldsymbol{z}$ ends with a number of zeros, which can be replaced with randomness.

4.4. Putting Everything Together

We present a non-interactive generic folding scheme with input as multiple committed CCS or atomic CCS instances in Construction 1. Zero knowledge is achieved as well by applying the techniques mentioned above. Since we already prove the security of the interactive folding scheme in the full-version paper, it is trivial to show that Construction 1 also satisfies completeness, knowledge soundness, and zero knowledge. At the end of this part, we give a comprehensive evaluation of the performance, including the prover cost, verifier cost, and communication complexity.

Construction 1 (Zero-knowledge non-interactive generic folding scheme)**.** *We construct a zero-knowledge non-interactive generic folding scheme as* zk-NIFS*, which consists of 4 PPT algorithms* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$*. Let* $\mathbb{H}$ *be the random oracle,* $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *denote an additively-homomorphic polynomial commitment scheme for multilinear polynomials. For generality, we assume the scheme takes input as multiple CCS instances, including*

- $\sum_{i\in[\ell_1]} s^{(i)}$ *atomic CCS instances in a set of relations* $\{\mathcal{R}^{(i)}\}_{i\in[\ell_1]}$ *with different structures* $\{\mathcal{S}^{(i)}\}_{i\in[\ell_1]}$*, where each relation* $\mathcal{R}^{(i)}$ *corresponds to* $s^{(i)}$ *instances. Let* $s_1 = \sum_{i\in[\ell_1]} s^{(i)}$*, each atomic CCS instance consists of* $(S^{(k)}, \mathbf{u}^{(k)}, \mathbf{w}^{(k)})$*, where* $\mathbf{u}^{(k)} = (C^{(k)}, v_0^{(k)}, \mathsf{io}^{(k)}, \boldsymbol{r}_x^{(k)}, \boldsymbol{r}_y^{(k)}, \{v_j^{(k)}\}_{j\in[t]}, v_z^{(k)})$ *for all* $k = 1, ..., s_1$*.*
- $\sum_{i=\ell_1+1}^{\ell_1+\ell_2} s^{(i)}$ *committed CCS instances in a set of relations* $\{\mathcal{R}^{(i)}\}_{i=\ell_1+1}^{\ell_1+\ell_2}$ *with different structures* $\{\mathcal{S}^{(i)}\}_{i=\ell_1+1}^{\ell_1+\ell_2}$*, where each relation* $\mathcal{R}^{(i)}$ *corresponds to* $s^{(i)}$ *instances. Let* $s_2 = \sum_{i=\ell_1+1}^{\ell_1+\ell_2} s^{(i)}$*, each committed CCS instance is consists of* $(S^{(k)}, \mathbf{u}^{(k)}, \mathbf{w}^{(k)})$*, where* $\mathbf{u}^{(k)} = (C^{(k)}, \mathsf{io}^{(k)})$ *for all* $k = s_1 + 1, ..., s_1 + s_2$

Denote $s = s_1 + s_2$*, we use* i, k *to index the relations (structures)* $\{\mathcal{R}^{(i)}\}_{i\in[\ell]}$ *and instances* $\{\mathbf{u}^{(k)}\}_{k\in[s]}$ *respectively. By applying Fiat-Shamir transformation to the zero-knowledge sum-check protocol mentioned in Section 4.3, we obtain the proving algorithm* $\mathrm{FS}[\Pi_{\mathsf{zksc}}].\mathcal{P}$ *with output transcript as* $\mathsf{ts} = (c, \Delta\mathsf{sum}, \{m_j\}_{j\in[s]}, \boldsymbol{r})$ *and the verifying algorithm* $\mathrm{FS}[\Pi_{\mathsf{zksc}}].\mathcal{V}$ *with output 1 for validity. For conciseness, we only present the concrete algorithms for* $\mathsf{zk\text{-}NIFS}.\mathcal{P}, \mathsf{zk\text{-}NIFS}.\mathcal{V}$*.*

```
zk-NIFS.$\mathcal{P}((\mathsf{pk},\mathsf{vk}),\{S^{(i)}\}_{i\in[\ell]},\{\mathbf{u}^{(k)}\}_{k\in[s]},\{\mathbf{w}^{(k)}\}_{k\in[s]})$ :
1 : Randomly sample $\{\boldsymbol{r}^{(k)}\in\mathbb{F}^{2t}\}_{k\in[s]},\boldsymbol{w}\in\mathbb{F}^{n-l-1}$.
2 : Generate instance $\mathbf{u}^{(0)}$ with $\boldsymbol{w}$.
3 : Pad each $\boldsymbol{z}^{(k)}$ with $\boldsymbol{r}^{(k)}$, update $\mathsf{sum}_x$.
4 : Generate claims on sums of $\widetilde{\boldsymbol{w}}(\boldsymbol{y}),\{\widetilde{\boldsymbol{r}}^{(k)}(\boldsymbol{y})\}_{k=0}^{s}$.
5 : Pad each $M_j^{(i)}$ with $I_j$.
6 : $\gamma,\boldsymbol{\alpha}\leftarrow\mathbb{H}(\{\mathbf{u}^{(k)}\}_{k=0}^{s})$, construct polynomial $f$.
7 : Run sum-check#1: $\mathsf{ts}_x\leftarrow\mathrm{FS}[\Pi_{\mathsf{zksc}}].\mathcal{P}(f,s_x,d+1,\mathsf{sum}_x)$.
8 : Generate claims on $\{\sigma_j^{(k)}\}_{k=0,j=1}^{s,t}$.
9 : $\delta\leftarrow\mathbb{H}(\mathsf{ts}_x,\{\sigma_j^{(k)}\}_{k=0,j=1}^{s,t})$, construct polynomial $g$..
10 : Run sum-check#2: $\mathsf{ts}_y\leftarrow\mathrm{FS}[\Pi_{\mathsf{zksc}}].\mathcal{P}(g,s_y,d+1,\mathsf{sum}_y)$.
11 : Generate claims on $\{\epsilon^{(k)}\}_{k=0}^{s},\{\theta_j^{(k)}\}_{k=0,j=1}^{s,t}$.
12 : $\eta\leftarrow\mathbb{H}(\mathsf{ts}_y,\{\epsilon^{(k)}\}_{k=0}^{s},\{\theta_j^{(k)}\}_{k=0,j=1}^{s,t})$.
13 : Set vectors for each instance $\mathbf{u}^{(k)},k=0,...,s$ as
     $\mathbf{v}^{(k)}:=(\{M_j^{(k)}\}_{j\in[t]},C^{(k)},v_0^{(k)},\mathbf{io}^{(k)},\{\theta_j^{(k)}\}_{j\in[t]},\epsilon^{(k)},\mathbf{w}^{(k)})$.
14 : $\mathbf{v}^*:=\sum_{k=0}^{s}\eta^k\cdot\mathbf{v}^{(k)}$, $M_j^*:=\sum_{i\in[\ell]}\eta^i\cdot M_j^{(i)},\forall j\in[t]$.
15 : Set folding proof as
     $\mathsf{pf}:=\begin{pmatrix}\mathbf{u}^{(0)},\gamma,\boldsymbol{\alpha},\boldsymbol{r}'_x,\mathsf{ts}_x,\{\sigma_j^{(k)}\}_{k=0,j=1}^{s,t},\\ \delta,\boldsymbol{r}'_y,\mathsf{ts}_y,\{\epsilon^{(k)}\}_{k=0}^{s},\{\theta_j^{(k)}\}_{k=0,j=1}^{s,t}\end{pmatrix}$.
16 : Output $(\{M_j^*\}_{j\in[t]},\mathbf{v}^*,\mathsf{pf})$.
```

```
zk-NIFS.$\mathcal{V}((\mathsf{pk},\mathsf{vk}),\{S^{(i)}\}_{i\in[\ell]},\{\mathbf{u}^{(k)}\}_{k\in[s]},\mathsf{pf})$ :
1 : Check $\mathsf{sum}_x$ with claims on $\widetilde{\boldsymbol{w}}(\boldsymbol{y}),\{\widetilde{\boldsymbol{r}}^{(k)}(\boldsymbol{y})\}_{k=0}^{s}$.
2 : Pad each $M_j^{(i)}$ with $I_j$.
3 : $\gamma,\boldsymbol{\alpha}\leftarrow\mathbb{H}(\{\mathbf{u}^{(k)}\}_{k=0}^{s})$.
4 : Check sum-check#1: $1\overset{?}{=}\mathrm{FS}[\Pi_{\mathsf{zksc}}].\mathcal{V}(\mathsf{ts}_x,s_x,d+1,\mathsf{sum}_x)$.
5 : Check claims on $\{\sigma_j^{(k)}\}_{k=0,j=1}^{s,t}$ with $c_x,\gamma,\boldsymbol{\alpha}$.
6 : $\delta\leftarrow\mathbb{H}(\mathsf{ts}_x,\{\sigma_j^{(k)}\}_{k=0,j=1}^{s,t})$.
7 : Check sum-check#2: $1\overset{?}{=}\mathrm{FS}[\Pi_{\mathsf{zksc}}].\mathcal{V}(\mathsf{ts}_y,s_y,d+1,\mathsf{sum}_y)$.
8 : Check claims of $\{\epsilon^{(k)}\}_{k=0}^{s},\{\theta_j^{(k)}\}_{k=0,j=1}^{s,t}$ with $c_y,\delta$.
9 : $\eta\leftarrow\mathbb{H}(\mathsf{ts}_y,\{\epsilon^{(k)}\}_{k=0}^{s},\{\theta_j^{(k)}\}_{k=0,j=1}^{s,t})$.
10 : Set vectors for each instance $\mathbf{u}^{(k)},k=0,...,s$ as
     $\mathbf{v}^{(k)}:=(\{M_j^{(k)}\}_{j\in[t]},C^{(k)},v_0^{(k)},\mathbf{io}^{(k)},\{\theta_j^{(k)}\}_{j\in[t]},\epsilon^{(k)},\mathbf{w}^{(k)})$.
11 : Check $\mathbf{v}^*:=\sum_{k=0}^{s}\eta^k\cdot\mathbf{v}^{(k)}$, $M_j^*:=\sum_{i\in[\ell]}\eta^i\cdot M_j^{(i)},\forall j\in[t]$.
```

Complexity. Denote the random oracle for sum-check protocol as $\mathbb{H}_{\mathsf{sc}}$. The folding scheme prover (1) asks $\log m+\log n$ queries to $\mathbb{H}_{\mathsf{sc}}$ and $\log m+2$ queries to $\mathbb{H}$; (2) computes sum-check protocols (steps 7,8,10,11 in zk-NIFS.$\mathcal{P}$) with $O(s_1(N+tm)+s_2(N+tm+qmd\log^2 d))$ $\mathbb{F}$-ops for all s_1 atomic CCS instances and s_2 committed CCS instances where for each atomic CCS instance, runs $O(N+tm)$ $\mathbb{F}$-ops according to the standard linear-time-sum-check techniques [36]; for each committed CCS instance, runs $O(N+tm+qmd\log^2 d)$ $\mathbb{F}$-ops according to the technique in SuperSpartan [21]; (3) performs $O(s)$ $\mathbb{G}$-ops to combine $\{C^{(k)}\}_{k=0}^{s}$; (4) performs $O(\ell N+s(n+t))$ $\mathbb{F}$-ops to combine $\{M_j^{(i)}\}_{i,j=1}^{\ell,t}$ and $\{\mathbf{v}^{(k)}\}_{k=0}^{s}$.

The folding scheme verifier (1) asks $\log m+\log n$ queries to $\mathbb{H}_{\mathsf{sc}}$ and $\log m+2$ queries to $\mathbb{H}$; (2) checks sum-check protocols (steps 4,5,7,8 in zk-NIFS.$\mathcal{V}$) with $O(s_1(d\log m+\log n)+s_2(dq+d\log m+\log n))$ $\mathbb{F}$-ops for all s_1 atomic CCS instances and s_2 committed CCS instances; (3) performs $O(s)$ $\mathbb{G}$-ops to combine $\{C^{(k)}\}_{k=0}^{s}$; (4) performs $O(\ell N+s(n+t))$ $\mathbb{F}$-ops to combine $\{M_j^{(i)}\}_{i,j=1}^{\ell,t}$ and $\{\mathbf{v}^{(k)}\}_{k=0}^{s}$.

Optimization. We further mention a bonus feature of the ACCS relation for optimizing the performance above. In real-world implementations, instead of directly operating on the raw data of a matrix M, we prefer to first squeeze the $m\cdot n$ field elements into a group element as $\mathsf{Commit}(M)$ with commitment schemes [15], [8]. Assume an output atomic CCS instance contains matrices $\{M_j\}_{j\in[t]}$ and corresponding claim values $\{v_j\}_{j=1}^{t}$ in the generic folding scheme, we observe that it is feasible to apply a batch verification of all claims, which means that instead of checking each claim independently, we can check only *one* claim on the aggregated matrices and values. To achieve this, the prover needs to compute $\sum_{j=1}^{t}\mathsf{Commit}(M_j)$ and $\sum_{j=1}^{t}v_j$ if the discrete logarithm independence is satisfied among these commitments (If not, the prover needs to compute linear combinations). As a result, the t matrix commitments and values of an atomic CCS instance can be aggregated into one matrix and one evaluation value, respectively. The number of matrix commitments in the folded instance sent to the next node is reduced from t to 1. For input two instances, the communication complexity can be reduced from $O(d\log m+\log n+tN)$ to $O(d\log m+\log n+N)$. As a tradeoff, the prover and verifier need to perform extra st group additions (multiple scalar multiplications) to aggregate matrix commitments. In the next section, we will introduce an approach to delegate this part of computation to further reduce the prover cost.

5. KiloNova: PCD from generic folding scheme

This section explains how to build non-uniform zero-knowledge PCD from the above-mentioned generic folding scheme. To begin with, we discuss the optimization technique used to reduce the overhead for handling structure folds in non-uniform PCD in Section 5.2. Based on this technique, we further construct zero-knowledge PCD from the zero-knowledge non-interactive generic folding scheme in Section 5.1.

5.1. New Constructions for Zero-knowledge PCD

This part constructs zero-knowledge PCD from our zero-knowledge non-interactive generic folding scheme. If zero knowledge is not considered, one can directly adapt the scheme in [27] to build a PCD from the folding scheme. However, a technical gap exists when we try to achieve zero knowledge of PCD. According to the conclusion in [11], a zero-knowledge PCD is built from an accumulation scheme (folding schemes in our paper) and an argument system (SNARK in [12] or NARK in [12]) for proving the recursive circuit, both of which are required to satisfy zero knowledge. Unfortunately, in the construction of [27], PCD prover only computes and outputs a committed CCS instance for the recursive circuit without providing zero knowledge. Adding zero-knowledge to the committed CCS instance will introduce extra prover cost, as discussed in Section 4.3.

Therefore, we must find another efficient approach to realize the zero-knowledge PCD. The general idea is to reuse the zero-knowledge folding scheme to transform the committed CCS instance into a zero-knowledge atomic CCS instance. To achieve this, we must redesign the original PCD construction in [27]. To state the problem clearly, we describe the predicates represented by the recursive circuit as follows

1. Check that the compliance predicate $\varphi(z, z_{\mathsf{loc}}, z_1, ..., z_s)$ satisfies.
2. Check that the hash values for all input instances with non-empty $z_k, k \in [s]$ are valid.
3. Run the zk-NIFS. $\mathcal{V}$ algorithm to check the validity of the folded instance.
4. Compute the hash value for the folded instance.

Note that predicates 3 and 4 will not leak information about $z_{\mathsf{loc}}, z_1, ..., z_s$ since the generic folding scheme already achieves zero knowledge. Thus, we only need to preserve the privacy of the witness $z_{\mathsf{loc}}, z_1, ..., z_s$ for predicates 1 and 2 (z is public). Thankfully, predicates 1 and 2 do not require the output of folding scheme zk-NIFS. $\mathcal{P}$, which means that they can be checked before the prover runs zk-NIFS. $\mathcal{P}$. We can split the circuit for the predicate into two parts as $\mathcal{R}_0, \mathcal{R}_1$ and handle them respectively in different steps. As a result, the prover first computes the instance $(\mathbf{u}_0, \mathbf{w}_0)$ for $\mathcal{R}_0$, then folds it with other input instances. In the circuit $\mathcal{R}_1$, the validity of the folded instances $\mathbf{U}$ is checked. And the prover computes another instance for $\mathcal{R}_1$. The idea is illustrated in Figure 3, where the recursive circuit is split into two parts for incremental computation F and folding scheme verification Multi-Fold. $\mathcal{V}$ in the right-side construction. The first part of F is written into a new instance-witness pair as $(\mathsf{u}', \mathsf{w}')$, which is folded into zero-knowledge pair (U_{i+1}, W_{i+1}) by the Multi-Fold. $\mathcal{P}$. Plus, the zero-knowledge folding scheme also ensures the pair (u_{i+1}, w_{i+1}) is zero knowledge.

Compared to the original construction, the PCD prover only needs to run the folding scheme with one more instance for $\mathcal{R}_0$, which adds negligible cost to its asymptotic complexity. This modification does not break the securities of PCD according to our security proofs. Moreover, the same modification can be applied to constructions of zero-knowledge IVC as long as they are built from a multi-folding scheme.

Construction 2 (A PCD from Generic Folding Schemes)**.** *Let* zk-NIFS *be the zero-knowledge non-interactive generic folding scheme for committed CCS and atomic CCS relations* $\{\mathcal{R}^{(i)}\}_{i\in[\ell]}$. *Let* $(\mathbf{u}_\perp, \mathbf{w}_\perp)$ *be a default trivially satisfying atomic CCS instance-witness pair for any structure and public parameters. According to the definition of PCD, we can construct a scheme consisting of polynomial-time algorithms* $\mathsf{PCD} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *for a class of compliance predicates* F. *Besides, we assume all the structures used below are valid, which is guaranteed by the extra IVC for proving the structure folds.*

Denote a compliance predicate φ *selected from* F *with a cryptographic hash function* Hash, *we first define the circuits* R_0 *and* R_1 *realizing the recursion on* s *inputs of* $\{z_k, \mathbf{U}_k, \mathbf{u}_k\}_{k=1}^{s}$.

$0/1 \leftarrow R_0(h; (z, z_{\mathsf{loc}}, \{z_k, \mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathsf{vk}'))$*:*

```
1 : Check the compliance predicate $\varphi(z, z_{\mathsf{loc}}, z_1, ..., z_s)$.
2 : For all $k \in [s]$ such that $z_k \neq \perp$, check that $\mathbf{u}_k.\mathbf{io} =$
    $\mathsf{Hash}(\mathsf{vk}', z_k, \mathbf{U}_k)$, where $\mathbf{u}_k.\mathbf{io}$ is the public IO of $\mathbf{u}_k$.
3 : If the above checks hold, output 1; otherwise, output 0.
```

Since R_0 *can be computed in polynomial time, it can be represented as a* $\mathcal{R}_{\mathsf{CCCS}}$ *structure* s_0. *Let*

$$(\mathsf{s}_0, \mathbf{u}_0, \mathbf{w}_0) \leftarrow \mathsf{trace}(R_0, (h, (z, z_{\mathsf{loc}}, \{z_k, \mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathsf{vk}')))$$

denote the satisfied $\mathcal{R}_{\mathsf{CCCS}}$ *instance for the execution of the circuit* R_0 *on input* $(h, (z, z_{\mathsf{loc}}, \{z_k, \mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathsf{vk}'))$.

$0/1 \leftarrow R_1(h; (\{\mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathbf{u}_0, \mathsf{vk}', \mathbf{U}, \Pi))$*:*

```
1 : If $z_k = \perp$ for all $k \in [s]$, check that $h = \mathsf{Hash}(\mathsf{vk}', z, \mathbf{U})$
    and $\mathbf{u}_0 = \mathbf{U}$, else check that
      (a) $\mathbf{U} = \mathsf{zk\text{-}NIFS}.\mathcal{V}'(\mathsf{vk}', \{\mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathbf{u}_0, \Pi)$.
      (b) $h = \mathsf{Hash}(\mathsf{vk}', z, \mathbf{U})$.
2 : If the above checks hold, output 1; otherwise, output 0.
```

Since R_1 *can be computed in polynomial time, it can be represented as a* $\mathcal{R}_{\mathsf{CCCS}}$ *structure* s. *Let*

$$(\mathsf{s}, \mathbf{u}, \mathbf{w}) \leftarrow \mathsf{trace}(R_0, (h, (\{\mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathbf{u}_0, \mathsf{vk}', \mathbf{U}, \Pi))$$

denote the satisfied $\mathcal{R}_{\mathsf{CCCS}}$ *instance for the execution of the circuit* R_1 *on input* $(h, (\{\mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathbf{u}_0, \mathsf{vk}', \mathbf{U}, \Pi))$.

Now, we can define the algorithms $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *for PCD.*

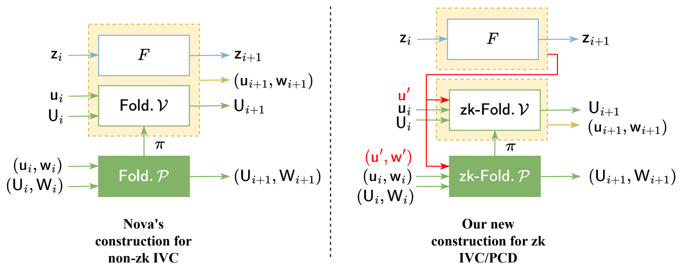
z_i
F
z_{i+1}
$(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$
u_i
U_i
Fold. $\mathcal{V}$
U_{i+1}
π
$(\mathsf{u}_i, \mathsf{w}_i)$
$(\mathsf{U}_i, \mathsf{W}_i)$
Fold. $\mathcal{P}$
$(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$
Nova's construction for non-zk IVC
z_i
F
z_{i+1}
u'
u_i
U_i
zk-Fold. $\mathcal{V}$
U_{i+1}
$(\mathsf{u}_{i+1}, \mathsf{w}_{i+1})$
π
$(\mathsf{u}', \mathsf{w}')$
$(\mathsf{u}_i, \mathsf{w}_i)$
$(\mathsf{U}_i, \mathsf{W}_i)$
zk-Fold. $\mathcal{P}$
$(\mathsf{U}_{i+1}, \mathsf{W}_{i+1})$
Our new construction for zk IVC/PCD

Figure 3: Comparison between previous construction (left) and our construction (right).

$\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$:

```
1 : Compute and output $\mathsf{pp}' \leftarrow \mathsf{zk\text{-}NIFS}.\mathcal{G}(1^\lambda)$.
```

$(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \varphi)$:

```
1 : Compute $(\mathsf{pk_{fs}}', \mathsf{vk_{fs}}') \leftarrow \mathsf{zk\text{-}NIFS}.\mathcal{K}'(\mathsf{pp}', R_\varphi)$.
2 : Output $(\mathsf{pk}, \mathsf{vk}) \leftarrow ((\varphi, \mathsf{pk_{fs}}'), (\varphi, \mathsf{vk_{fs}}'))$.
```

$\Pi \leftarrow \mathcal{P}(\mathsf{pk}, z, z_{\mathsf{loc}}, \{z_k, \Pi_k\}_{k=1}^s)$:

```
1 : For $k \in [s]$, parse $\Pi_k$ as satisfied ACCS instance
    $(\mathsf{S}_k, \mathbf{U}_k, \mathbf{W}_k)$ and satisfied CCCS instance $(\mathsf{s}_k, \mathbf{u}_k, \mathbf{w}_k)$).
2 : $(\mathsf{s}_0, \mathbf{u}_0, \mathbf{w}_0) \leftarrow \mathsf{trace}(R_0, (h, (z, z_{\mathsf{loc}}, \{z_k, \mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathsf{vk}'))$
3 : If $z_k = \perp$ for all $k \in [s]$, then set $(\mathbf{U}, \mathbf{W}, \mathsf{pf}) := (\mathbf{u}_0, \mathbf{w}_0, \perp)$,
    else compute $(\mathsf{S}, \mathbf{U}, \mathbf{W}, \mathsf{pf})$ output by
    $\mathsf{zk\text{-}NIFS}.\mathcal{P}'(\mathsf{pk_{fs}}, \{\mathsf{S}_k, \mathbf{U}_k, \mathbf{W}_k\}_{k\in[s]}, \{\mathsf{s}_k, \mathbf{u}_k, \mathbf{w}_k\}_{k=0}^s)$.
4 : Compute $h \leftarrow \mathsf{Hash}(\mathsf{vk_{fs}}', z, \mathbf{U})$.
5 : $(\mathsf{s}, \mathbf{u}, \mathbf{w}) \leftarrow \mathsf{trace}(R_1, (h, (\{\mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathbf{u}_0, \mathsf{vk_{fs}}', \mathbf{U}, \mathsf{pf})))$.
6 : Output $\Pi := ((\mathsf{S}, \mathbf{U}, \mathbf{W}), (\mathsf{s}, \mathbf{u}, \mathbf{w}))$.
```

$0/1 \leftarrow \mathcal{V}(\mathsf{vk}, z, \Pi)$:

```
1 : Parse $\Pi$ as $((\mathsf{S}, \mathbf{U}, \mathbf{W}), (\mathsf{s}, \mathbf{u}, \mathbf{w}))$.
2 : Check that $\mathbf{u}.\mathbf{io} = \mathsf{Hash}(\mathsf{vk_{fs}}', z, \mathbf{U})$.
3 : Check that $\mathbf{W}$ is a satisfied $\mathcal{R}_{\mathsf{ACCS}}$ witness to $\mathbf{U}$ and $\mathbf{w}$ is a
    satisfied $\mathcal{R}_{\mathsf{CCCS}}$ witness to $\mathbf{u}$.
4 : If the above checks hold, output 1; otherwise, output 0.
```

Theorem 3. $\mathsf{PCD} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *in Construction 2 for a class of compliance predicates* F *with constant depth in definition XX satisfies the perfect completeness, knowledge soundness, and zero knowledge in the random oracle.*

The proof of Theorem 3 is presented in our full-version paper. We present the evaluation of complexity below.

Complexity. Denote the random oracle for sum-check protocol as $\mathbb{H}_{\mathsf{sc}}$. The recursive cost at each step contains (1) computing the compliance predicate φ; (2) computing r times pf hash function Hash; (3) invoking $\mathsf{zk\text{-}NIFS}.\mathcal{V}'$ with $2\log m + \log n$ random oracle queries, $O(s(dq + d\log m + \log n))$ $\mathbb{F}$-ops and s $\mathbb{G}$-ops.

The native prover at each step cost contains (1) invoking $\mathsf{zk\text{-}NIFS}.\mathcal{P}'$ with $2\log m + \log n$ random oracle queries, $O(s(N + tm + qmd\log^2 d))$ $\mathbb{F}$-ops and s $\mathbb{G}$-ops; (2) computing 1 time of hash function Hash; (3) computing two satisfying committed CCS instances for the execution of $\mathcal{R}_0, \mathcal{R}_1$, which is dominated by computing the commitment of witness $\mathbf{w}_0, \mathbf{w}$ with $O(n)$ $\mathbb{G}$-ops.

The proof Π consists of an atomic CCS instance $(\mathsf{S}, \mathbf{U}, \mathbf{W})$ and a committed CCS instance $(\mathsf{s}, \mathbf{u}, \mathbf{w})$, which are linear in the size of $\mathcal{R}_\varphi$. While according to previous work in Nova [13] and HyperNova [14], we can fold these two instances with $\mathsf{zk\text{-}NIFS}.\mathcal{P}$ and apply a general SNARK (the folded instance is already zero knowledge) to prove their validity. For example, instantiating a polynomial IOP based on Bulletproofs polynomial commitment schemes [19] for $(\mathsf{S}, \mathbf{U}, \mathbf{W})$ and $(\mathsf{s}, \mathbf{u}, \mathbf{w})$ can reduce the proof size to $O(\log m)$.

The PCD verifier cost contains (1) invoking $\mathsf{zk\text{-}NIFS}.\mathcal{V}$ for two instances with $2\log m + \log n$ random oracle queries, $O((dq + d\log m + \log n))$ $\mathbb{F}$-ops and 2 $\mathbb{G}$-ops; (2) verification of polynomial commitments with $O(\log n)$ random oracle queries and $O(n)$ $\mathbb{G}$-ops; (3) verification of the IVC system for structure folds with $O(mn)$ $\mathbb{F}$-ops.

The security of the folding scheme still holds. Here, we only give sketch proof of the knowledge soundness. Assume the original folding scheme zk-NIFS satisfies knowledge soundness. Thus, there exists an extractor Ext runs in polynomial time for zk-NIFS, which succeeds in extracting witness with non-negligible probability ϵ. For the new aggregated folding scheme, an extractor Ext' can also be constructed by calling Ext. The extractor Ext' invokes Ext to obtain t transcripts, each with different challenge $\zeta^{(i)}, i = 1, .., t$. By interpolating, the extractor can compute the matrices $\{M_j^*\}_{j\in[t]}$ and values $\{v_j^*\}_{j=1}^t$ from the linear combined M^* and v^*. It is naive to argue that Ext' runs in polynomial time. For the advantage Ext' succeeds with probability $(\epsilon - \mathrm{negl}(\lambda)) \cdot (1 - \mathrm{negl}(\lambda))$. This is because given that Ext does not abort, the probability that different challenges are sampled with less than $\sqrt[d+1]{|\mathbb{F}|}$ rewinds, i.e., $\zeta^{(1)} \neq \cdots \neq \zeta^{(t)}$, is $(1 - O(1)/\sqrt[d+1]{|\mathbb{F}|}) \cdot \epsilon \cdot (1 - \sqrt[d+1]{|\mathbb{F}|}^d / |\mathbb{F}|)$.

5.2. Delegation Schemes.

We first explain the complexity problem caused by structure folds. In real-world implementations, the folding scheme verifier does not directly compute the linear combination of matrices as described in step 11 of $\mathsf{zk\text{-}NIFS}.\mathcal{V}$. Instead, commitments on each matrix are used as mentioned at the end of Section 4, which incurs a large number of group scalar multiplications in the recursive circuits when handling multiple non-uniform instances. Take our generic folding scheme as an example, the verifier $\mathsf{zk\text{-}NIFS}.\mathcal{V}$ needs to compute the linear combination of matrix commitments $\{\mathsf{Commit}(M_j^{(i)})\}_{i\in[\ell]}$ for all $j = [t]$, leading to $O(\ell \cdot t)$ $\mathbb{G}$ operations in total. Even worse, the matrices to be folded are less likely to be sparse, hindering the use of preprocessing approaches such as encoding the matrix into smaller polynomials [17]. This raises complexity concerns for PCD, especially when the prover needs to fold multiple instances with different structures among mutually distrustful nodes. On the one hand, the folding verification logic should be written into the recursive circuit, increasing the prover cost. On the other hand, the folded matrix commitments should be sent to the next node in PCD, incurring a high communication cost.

To alleviate this problem, we introduce an optimization technique for delegating the costly structure folds. In our folding scheme, the prover folds the structures by computing the linear combinations of public matrices, and the verifier repeats the same process. This subprotocol is independent of other steps in the folding scheme. Thus, the original prover

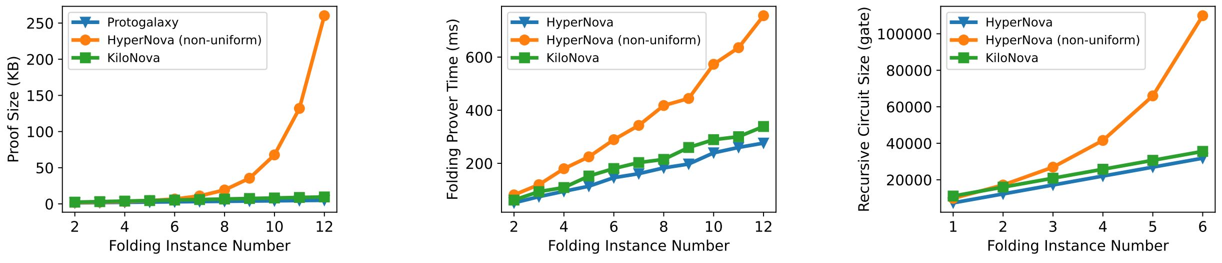

Figure 4: Performance evaluation of KiloNova. See the text for details.

can prove the structure folds by delegating to a third party[7]. This is because the matrices are selected from a public list, e.g., an instruction set. Such delegation is extremely useful for optimizing the performance of PCD because the delegated computation is an IVC executed by only one prover, which avoids communication between different nodes.

As a result, we can safely remove the verification of structure folds from the original prover as long as ensuring the consistency of the challenge for folding. This observation also applies to other non-uniform folding schemes such as Protostar and Protogalaxy [15], [8]. Concretely, the prover runs another IVC system for iteratively computing the linear combination of matrix commitments for folding the structure. We believe an IVC system instantiated from Nova [13] is sufficient for this task. One thing left is to ensure the delegated prover uses the same challenge η as the original prover. To achieve this, we can instantiate an accumulator with the binding property in both parties. A naive approach is computing an extra function as $r_i = \mathsf{Hash}(r_{i-1}, \eta_i)$ at each step. Other existing accumulators, such as [37], can also be utilized. Since these accumulators are also incremental computations, it is natural to add this computation in the original function F in the IVC system. Figure 5 illustrate this construction in details.

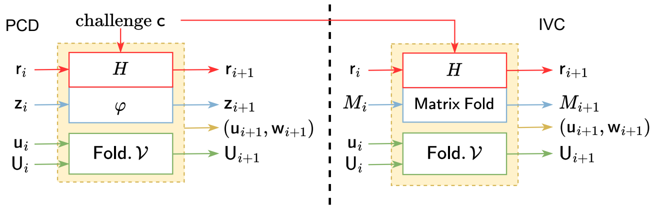

Figure 5: Delegation schemes for PCD. The structure folds are delegated to a third party running the IVC. To ensure the soundness property, an extra accumulation for the challenge c is instantiated in the recursive circuits in the red box.

6. Experimental evaluation

We validate the theoretic results by implementing a proof-of-concept project of KiloNova in Rust with the secp256k1 elliptic curve. Note that this demo PCD only supports R1CS relations because there is no off-the-shelf library for building CCS circuits from rust (left as our future work). Note that applying R1CS in our evaluation only shows the lower bound of our improvements compared to other schemes because the high-degree case is not considered. We executed our experiments on a personal laptop running Linux WSL with an i5 10th generation CPU and 16GB RAM. We ran all tests single-threaded. The codes can be referred to the link (https://github.com/Anonymous-sadh769/KiloNova-poc.git).

In our experiments, we first compare the performance between KiloNova and the other two advanced schemes as HyperNova [14] and Protogalaxy [8]. For the evaluation of folding schemes, we set the R1CS circuit size as 2^4 and scale the folding instance number from 2^2 to 2^{12}. The left subfigure in Figure 4 presents the comparison of their proof size under different numbers of non-uniform instances. Since HyperNova is not designed for non-uniform circuits, directly adopting its folding scheme here incurs exponentially growing proof size. As a comparison, KiloNova and Protogalaxy optimize the cross-term problem and therefore achieve a significantly smaller proof size [8]. However, Protogalaxy sacrifices its prover complexity for the above optimization [9], while KiloNova retains the folding prover cost at the linear level. The middle one in Figure 4 shows that the prover time of KiloNova for non-uniform circuits is only slightly slower than HyperNova for uniform circuits. This difference is mainly owing to the structure folds operations, which is inevitable in non-uniform occasions. For evaluating the PCD performance, the right subfigure compares the recursive circuit size between KiloNova and HyperNova, the results demonstrate that our scheme dramatically reduces the recursion overhead for folding non-uniform instances compared to HyperNova. For instances with fixed circuit sizes, the recursive circuit is dominated by the scalar multiplications in the folding scheme and thus scales linearly to the instance number. Moreover, we also implement zero knowledge property for KiloNova, the experimental results indicate that our zero-knowledge generic folding schemes increase around 25% prover cost compared to the original version, which is acceptable in the applications.

7. It is also practical to delegate this task directly to the final verifier (or decider) if it is not very frequent to run the verifier in PCD.

8. The proof size of KiloNova is almost the same as HyperNova for uniform circuits.

9. The Protogalaxy has quasilinear prover complexity and a large constant part of computing Lagrange basis, our experiment does not include it because lack of an implementation

References

[1] P. Valiant, "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency," in *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*. Springer, 2008, pp. 1–18.

[2] A. Chiesa, "Proof-carrying data," Ph.D. dissertation, Massachusetts Institute of Technology, 2010.

[3] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "Recursive composition and bootstrapping for snarks and proof-carrying data," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 2013, pp. 111–120.

[4] A. Chiesa, E. Tromer, and M. Virza, "Cluster computing in zero knowledge," in *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34*. Springer, 2015, pp. 371–403.

[5] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro, "Mina: Decentralized cryptocurrency at scale," *New York Univ. O (1) Labs, New York, NY, USA, Whitepaper*, pp. 1–47, 2020.

[6] J. Bonneau, I. Meckler, and V. Rao, "Coda: Decentralized cryptocurrency at scale," *Cryptology ePrint Archive*, 2020.

[7] J. Beal and B. Fisch, "Derecho: Privacy pools with proof-carrying disclosures," *Cryptology ePrint Archive*, 2023.

[8] L. Eagen and A. Gabizon, "Protogalaxy: Efficient protostar-style folding of multiple instances," *Cryptology ePrint Archive*, 2023.

[9] S. Bowe, J. Grigg, and D. Hopwood, "Recursive proof composition without a trusted setup," *Cryptology ePrint Archive*, 2019.

[10] D. Boneh, J. Drake, B. Fisch, and A. Gabizon, "Halo infinite: Proof-carrying data from additive polynomial commitments," in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*. Springer, 2021, pp. 649–680.

[11] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner, "Proof-carrying data from accumulation schemes," *Cryptology ePrint Archive*, 2020.

[12] B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner, "Proof-carrying data without succinct arguments," in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*. Springer, 2021, pp. 681–710.

[13] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *Annual International Cryptology Conference*. Springer, 2022, pp. 359–388.

[14] A. Kothapalli and S. Setty, "Hypernova: Recursive arguments for customizable constraint systems," *Cryptology ePrint Archive*, 2023.

[15] B. Bünz and B. Chen, "Protostar: Generic efficient accumulation/folding for special sound protocols," *Cryptology ePrint Archive*, 2023.

[16] M. Bellare, J. A. Garay, and T. Rabin, "Fast batch verification for modular exponentiation and digital signatures," in *Advances in Cryptology—EUROCRYPT'98: International Conference on the Theory and Application of Cryptographic Techniques Espoo, Finland, May 31–June 4, 1998 Proceedings 17*. Springer, 1998, pp. 236–250.

[17] S. Setty, "Spartan: Efficient and general-purpose zksnarks without trusted setup," in *Annual International Cryptology Conference*. Springer, 2020, pp. 704–737.

[18] T. Zheng, S. Gao, Y. Song, and B. Xiao, "Leaking arbitrarily many secrets: Any-out-of-many proofs and applications to ringct protocols," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2533–2550.

[19] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 315–334.

[20] B. E. Diamond, "Many-out-of-many proofs and applications to anonymous zether," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1800–1817.

[21] S. Setty, J. Thaler, and R. Wahby, "Customizable constraint systems for succinct arguments," *Cryptology ePrint Archive*, 2023.

[22] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.

[23] J. Bootle, A. Chiesa, and S. Liu, "Zero-knowledge iops with linear-time prover and polylogarithmic-time verifier," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2022, pp. 275–304.

[24] S. Chu, B. H. Gomes, F. H. Iglesias, T. Norton, and D. Tebbs, "Uniplonk: Plonk with universal verifier," *Cryptology ePrint Archive*, 2023.

[25] S. Bowe, J. Grigg, and D. Hopwood, "Halo2," https://github.com/zcash/halo2, 2020.

[26] A. Kothapalli and S. Setty, "Supernova: Proving universal machine executions without universal circuits," *Cryptology ePrint Archive*, 2022.

[27] Z. Zhou, Z. Zhang, and J. Dong, "Proof-carrying data from multi-folding schemes," *Cryptology ePrint Archive*, 2023.

[28] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2111–2128.

[29] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," *Cryptology ePrint Archive*, 2019.

[30] J. Thaler *et al.*, "Proofs, arguments, and zero-knowledge," *Foundations and Trends® in Privacy and Security*, vol. 4, no. 2–4, pp. 117–660, 2022.

[31] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.

[32] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," *Journal of the ACM (JACM)*, vol. 39, no. 4, pp. 859–868, 1992.

[33] T. Xie, Y. Zhang, and D. Song, "Orion: Zero knowledge proof with linear prover time," in *Annual International Cryptology Conference*. Springer, 2022, pp. 299–328.

[34] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. Springer, 2019, pp. 733–764.

[35] A. Chiesa, M. A. Forbes, and N. Spooner, "A zero knowledge sumcheck and its applications," *arXiv preprint arXiv:1704.02086*, 2017.

[36] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Annual Cryptology Conference*. Springer, 2013, pp. 71–89.

[37] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*. Springer, 2019, pp. 561–586.

[38] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting," in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer, 2016, pp. 327–357.

Appendix A. Formal Definitions

A.1. Definitions for Polynomial Commitment

We adapt the definition of the polynomial commitment scheme from [BFS20].

Definition 7 (Polynomial commitment (PC))**.** *A polynomial commitment (PC) scheme for multilinear polynomials is defined as a tuple of four protocols* $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$*:*

- $\mathsf{Gen}(1^\lambda, \ell) \rightarrow \mathsf{pp}$*: takes as input* ℓ *(the number of variables in a multilinear polynomial); produces public parameters* pp.
- $\mathsf{Commit}(\mathsf{pp}, f) \rightarrow C$*: takes as input an* ℓ*-variate multilinear polynomial* $f : \mathbb{F}^\ell \rightarrow \mathbb{F}$*; produces a commitment* C.
- $\mathsf{Open}(\mathsf{pp}, C, f) \rightarrow b$*: verifies the opening of commitment* C *to the* ℓ*-variate multilinear polynomial* f*; outputs* $b \in \{0, 1\}$.
- $\mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y, \ell, f) \rightarrow b$ *is a protocol between a PPT prover* $\mathcal{P}$ *and verifier* $\mathcal{V}$*. Both* $\mathcal{V}$ *and* $\mathcal{P}$ *hold a commitment* C*, the number of variables* ℓ*, a scalar* $y \in \mathbb{F}$*, and* $\boldsymbol{x} \in \mathbb{F}^\ell$*.* $\mathcal{P}$ *additionally knows an* ℓ*-variate multilinear polynomial* f*.* $\mathcal{P}$ *attempts to convince* $\mathcal{V}$ *that* $f(\boldsymbol{x}) = y$*. At the end of the protocol,* $\mathcal{V}$ *outputs* $b \in \{0, 1\}$.

Definition 8. *A polynomial commitment scheme for multilinear polynomials* $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *is additively homomorphic if for all* ℓ *and* pp *produced from* $\mathsf{Setup}(1^\lambda, \ell)$*, and for any* $f_1, f_2 : \mathbb{F}^\ell \rightarrow \mathbb{F}$*,*

$$\mathsf{Commit}(\mathsf{pp}, f_1) + \mathsf{Commit}(\mathsf{pp}, f_2) = \mathsf{Commit}(\mathsf{pp}, f_1 + f_2).$$

A PC is an extractable polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$ if it satisfies completeness, binding, and knowledge soundness properties as defined below.

Completeness. PC has completeness if for all ℓ-variate multilinear polynomial $g \in \mathbb{F}[\ell]$, the following probability is close to 1:

$$\Pr\left[\begin{array}{l} \mathsf{Eval}(\mathsf{pp}_{\mathsf{pc}}, C, \ell, r, v; f) \\ = 1 \ \wedge \ f(r) = v \end{array} \middle| \begin{array}{l} \mathsf{pp}_{\mathsf{pc}} \leftarrow \mathsf{Setup}(1^\lambda, \ell); \\ C \leftarrow \mathsf{Commit}(\mathsf{pp}_{\mathsf{pc}}, f) \end{array}\right].$$

Binding. PC has binding if for any PPT adversary $\mathcal{A}$, size parameter $\ell > 1$, the following probability is negligible:

$$\Pr\left[\begin{array}{l} b_0 = b_1 \neq 0 \\ \wedge f_0 \neq f_1 \end{array} \middle| \begin{array}{l} \mathsf{pp}_{\mathsf{pc}} \leftarrow \mathsf{Setup}(1^\lambda, \ell); \\ (C, f_0, f_1) \leftarrow \mathcal{A}(\mathsf{pp}_{\mathsf{pc}}); \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}_{\mathsf{pc}}, C, f_0); \\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}_{\mathsf{pc}}, C, f_1) \end{array}\right].$$

Knowledge soundness. PC has knowledge soundness if given $\mathsf{pp}_{\mathsf{pc}} \leftarrow \mathsf{Setup}(1^\lambda, \ell)$, Eval is a succinct argument of knowledge for NP relation

$$\mathcal{R}_{\mathsf{Eval}}(\mathsf{pp}_{\mathsf{pc}}) = \left\{ (C, r, v; f) : \begin{array}{c} f \in \mathbb{F}[\ell] \wedge f(r) = v \\ \wedge \mathsf{Open}(\mathsf{pp}_{\mathsf{pc}}, C, f) = 1 \end{array} \right\}.$$

A.2. Security Definitions for PCD

A proof-carrying data scheme PCD should satisfy the perfect completeness, knowledge soundness, and zero knowledge properties.

Perfect Completeness. PCD has perfect completeness if for every adversary $\mathcal{A}$, the following probabilities equals to 1:

$$\Pr\left[\mathcal{V}(\mathsf{vk}, z, \Pi) = 1 \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda); \\ (\varphi, z, z_{\mathsf{loc}}, \{z_i, \Pi_i\}_{i=1}^r) \leftarrow \mathcal{A}(\mathsf{pp}); \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \varphi); \\ \varphi \in \mathsf{F}; \varphi(z, z_{\mathsf{loc}}, \{z_i\}_{i=1}^r) = 1; \\ \forall i \in [r], z_i = \perp \text{ or} \\ \mathcal{V}(\mathsf{vk}, z_i, \Pi_i) = 1; \\ \Pi \leftarrow \mathcal{P}(\mathsf{pk}, z, z_{\mathsf{loc}}, \{z_i, \Pi_i\}_{i=1}^r) \end{array} \right].$$

Knowledge soundness. PCD has knowledge soundness (w.r.t. an auxiliary input distribution $\mathcal{D}$) if for every expected polynomial time adversary $\mathcal{P}^*$, there exists an expected polynomial time extractor $\mathsf{Ext}_{\mathcal{P}^*}$ such that for every set Z, the difference of the following two probabilities is negligible:

$$\Pr\left[\begin{array}{l} \varphi \in \mathsf{F} \\ \wedge (\mathsf{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z \\ \wedge \mathsf{T} \text{ is } \varphi\text{-compliant} \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda); \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}); \\ (\varphi, \mathsf{T}, \mathsf{ao}) \\ \leftarrow \mathsf{Ext}_{\mathcal{P}^*}(\mathsf{pp}, \mathsf{ao}) \end{array} \right],$$

and

$$\Pr\left[\begin{array}{l} \varphi \in \mathsf{F} \\ \wedge (\mathsf{pp}, \mathsf{ai}, \varphi, \mathsf{o}, \mathsf{ao}) \in Z \\ \wedge \mathcal{V}(\mathsf{vk}, \mathsf{o}, \Pi) = 1 \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda); \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}); \\ (\varphi, \mathsf{o}, \Pi, \mathsf{ao}) \\ \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{ai}); \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \varphi) \end{array} \right].$$

Zero Knowledge. PCD has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator Sim such that for every polynomial-size honest adversary $\mathcal{A}$ the distributions below are computationally indistinguishable:

$$\left\{ (\mathsf{pp}, \Pi) \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda); \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \Pi_i]_{i=1}^r) \leftarrow \mathcal{A}(\mathsf{pp}); \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \varphi); \\ \Pi \leftarrow \mathcal{P}(\mathsf{pk}, \varphi, z, z_{\mathsf{loc}}, [z_i, \Pi_i]_{i=1}^r) \end{array} \right\}$$

and

$$\left\{ (\mathsf{pp}, \Pi) \middle| \begin{array}{r} (\mathsf{pp}, \tau) \leftarrow \mathsf{Sim}(1^\lambda); \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \Pi_i]_{i=1}^r) \leftarrow \mathcal{A}(\mathsf{pp}); \\ \Pi \leftarrow \mathsf{Sim}(\mathsf{pp}, \varphi, z, \tau) \end{array} \right\}.$$

A.3. Definitions for CCS

Definition 9 (CCS [21])**.** *We define the customizable constraint system (CCS) relation* $\mathcal{R}_{\mathsf{CCS}}$ *as follows. Let the public parameter consist of size bounds* $m, n, N, l, t, q, d \in \mathbb{N}$ *where* $n > l$.

An $\mathcal{R}_{\mathsf{CCS}}$ *structure* $\mathcal{S}$ *consists of:*

- *a sequence of matrices* $\{M_j \in \mathbb{F}^{m \times n}\}_{j \in [t]}$ *with at most* $N = \Omega(\max(m, n))$ *non-zero entries in total;*
- *a sequence of* q *multisets* $\{S_i\}_{i \in [q]}$*, where an element in each multiset is from the domain* $\{1, ..., t\}$ *and the cardinality of each multiset is at most* d.

- *a sequence of q constants $\{c_i\}_{i\in[q]}$, where each constant is from $\mathbb{F}$.*

A $\mathcal{R}_{\mathsf{CCS}}$ instance consists of public input and output $\mathbf{io} \in \mathbb{F}^l$.
A $\mathcal{R}_{\mathsf{CCS}}$ witness consists of a vector $\mathbf{w} \in \mathbb{F}^{n-l-1}$.
A $\mathcal{R}_{\mathsf{CCS}}$ instance $\mathbf{io}$ with structure $\mathcal{S}$ is satisfied by a $\mathcal{R}_{\mathsf{CCS}}$ witness $\mathbf{w}$ if

$$\sum_{i\in[q]} c_i \cdot \bigcirc_{j\in S_i} M_j \cdot \boldsymbol{z} = \mathbf{0}, \tag{5}$$

where $\boldsymbol{z} = (\mathbf{w}, 1, \mathbf{io}) \in \mathbb{F}^n$, $M_j \cdot \boldsymbol{z}$ denotes matrix-vector multiplication, $\bigcirc$ denotes the Hadamard product between vectors, and $\mathbf{0}$ is an m-sized vector with entries equal to the additive identity in $\mathbb{F}$.

A.4. Security Definitions for Generic Folding Schemes

We slightly abuse the vector-from denotation $(\mathsf{pp}, \mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}) \in \mathcal{R}^{(i)}$ to represent that $(\mathsf{pp}, \mathcal{S}^{(i)}, \mathbf{u}_j^{(i)}, \mathbf{w}_j^{(i)}) \in \mathcal{R}^{(i)}$ for all $j \in [s^{(i)}]$. A generic folding scheme for $\{\mathcal{R}^{(i)}\}_{i=1}^{\ell}$ satisfies the following requirements.

Perfect Completeness: For all PPT adversaries $\mathcal{A}$, the following probability equals to 1:

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}) & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ \in \mathcal{R}^{(i)} \ \forall\ i = [1, \ell] & \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}\}_{i=1}^{\ell} \leftarrow \mathcal{A}(\mathsf{pp}), \\ \Downarrow & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \{\mathcal{S}^{(i)}\}_{i=1}^{\ell}), \\ (\mathsf{pp}, S^*, \mathbf{u}^*, \mathbf{w}^*) \in \mathcal{R}^* & (\mathcal{S}^*, \mathbf{u}^*, \mathbf{w}^*) \leftarrow \Pi_{\mathsf{fold}}((\mathsf{pk}, \mathsf{vk}), \\ & \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}\}_{i=1}^{\ell}) \end{array}\right].$$

Knowledge Soundness: For any expected polynomial-time adversaries $\mathcal{A}$ and $\mathcal{P}^*$, Π^*_{fold} is run by $\mathcal{P}^*, \mathcal{V}$, there is an expected polynomial-time extractor Ext such that for all randomness ρ, the folowing two probabilities are computationally indistinguishable:

$$\Pr\left[\begin{array}{c|r} (\mathsf{pp}, \mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}) & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ \in \mathcal{R}^{(i)} \ \forall\ i = [1, \ell] & (\{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \rho), \\ & \{\mathbf{w}^{(i)}\}_{i\in[\ell]} \leftarrow \mathsf{Ext}(\mathsf{pp}, \rho) \end{array}\right]$$

and

$$\Pr\left[\begin{array}{c|r} & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ & (\{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \rho), \\ (\mathsf{pp}, \mathcal{S}^*, \mathbf{u}^*, \mathbf{w}^*) & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \{\mathcal{S}^{(i)}\}_{i=1}^{\ell}), \\ \in \mathcal{R}^* & (\mathcal{S}^*, \mathbf{u}^*, \mathbf{w}^*) \leftarrow \Pi^*_{\mathsf{fold}}(\\ & (\mathsf{pk}, \mathsf{vk}), \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell}, \mathsf{st}) \end{array}\right].$$

Efficiency: The communication costs and $\mathcal{V}$'s computation are lower in the case where $\mathcal{V}$ participates in the generic folding scheme and then checks a witness sent by $\mathcal{P}$ for the folded instance than in the case where $\mathcal{V}$ checks witnesses sent by $\mathcal{P}$ for each of the original instances.

Definition 10 (Honest Verifier Zero-knowledge)**.** *Let* $\mathsf{trace}(\Pi_{\mathsf{fold}}, \mathsf{input})$ *denote the non-deterministic function which takes as input an interaction function Π_{fold} and a prescribed input* input*, and produces an interaction transcript between $\mathcal{P}$ and $\mathcal{V}$ on* input*. A generic folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ for $\{R^{(i)}, s^{(i)}\}_{i=1}^{\ell}$ satisfies honest verifier zero-knowledge if there exists a PPT simulator* Sim *such that for all PPT adversaries $\mathcal{A}$, the following distributions are (statistically/computationally) indistinguishable*

$$\left\{\begin{array}{c|r} & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ & (\{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}\}_{i=1}^{\ell}) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell}, \mathsf{tr}) & \{(\mathsf{pp}, \mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}) \in \mathcal{R}^{(i)}\}_{i=1}^{\ell}, \\ & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \{\mathcal{S}^{(i)}\}_{i=1}^{\ell}), \\ & \mathsf{tr} \leftarrow \mathsf{trace}(\Pi_{\mathsf{fold}}, (\mathsf{pk}, \mathsf{vk}), \\ & \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}\}_{i=1}^{\ell}) \end{array}\right\}$$

and

$$\left\{\begin{array}{c|r} & (\mathsf{pp}, \tau) \leftarrow \mathsf{Sim}(1^\lambda), \\ (\mathsf{pp}, \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell}, \mathsf{tr}) & (\{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}), \\ & \{(\mathsf{pp}, \mathcal{S}^{(i)}, \mathbf{u}^{(i)}, \mathbf{w}^{(i)}) \in \mathcal{R}^{(i)}\}_{i=1}^{\ell}, \\ & \mathsf{tr} \leftarrow \mathsf{Sim}(\mathsf{pp}, \{\mathcal{S}^{(i)}, \mathbf{u}^{(i)}\}_{i=1}^{\ell}, \tau) \end{array}\right\}.$$

Definition 11 (Non-interactive)**.** *A generic folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is non-interactive if the interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of a single message from $\mathcal{P}$ to $\mathcal{V}$. This single message is denoted as $\mathcal{P}$'s output and as $\mathcal{V}$'s input.*

Definition 12 (Public coin)**.** *A generic folding scheme $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is called public coin if all the messages sent from $\mathcal{V}$ to $\mathcal{P}$ are sampled from a uniform distribution.*

Appendix B. Details for Generic Folding Schemes

We first present two special sound protocols for committed CCS and atomic CCS relations separately to illustrate how to run "early stopping" SuperSpartan for them. According to the workflow in Figure 2, one can first assume running the special sound protocol for each instance independently and then applying aggregation techniques for their intermediate steps, e.g., sum-check protocols and polynomial evaluations.

B.1. Special Sound Protocol for Committed CCS

This part describes special sound protocols for the committed CCS relation $\mathcal{R}_{\mathsf{CCCS}}$. The basic idea is to commit the special sound protocol in SuperSpartan [21]. Unlike the general-purpose protocol described in Protostar [15], the special sound protocol we used is specified for concrete CCS relations because the relation itself is already expressive enough. Note that Protostar also covers CCS relations with their special sound protocol to manifest expressiveness. While their protocols are not based on sum-check protocols. The performance of the final scheme is still restricted by the accumulation scheme they proposed.

We instantiate a protocol with a series of interactions between two parties $(\mathcal{P}, \mathcal{V})$, checking the validity of relation $\mathcal{R}_{\mathsf{CCCS}}$ by running sum-check protocols on the target multivariate polynomials. The running steps of the protocol are given in Π_{CCCS} below. Specifically, the prover wants to convince that Equation (2) holds for all $\boldsymbol{x} \in B_{s_x}$. To check this equation with sum-check protocol, a trick is introduced in Spartan [17]: if multiply each value with a corresponding term $\widetilde{eq}(\boldsymbol{\alpha}, \boldsymbol{x})$ with random chosen $\boldsymbol{\alpha}$, then their sum equals to zero only when all values equal to zero with high probability. Formally, denote $\widetilde{F}(\boldsymbol{x})$ as

$$\widetilde{F}(\boldsymbol{x}) = \sum_{i\in[q]} c_i \left(\prod_{j\in S_i} \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}_j(\boldsymbol{x},\boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}) \right) \right),$$

denote $\widetilde{Q}(\boldsymbol{t})$ as

$$\widetilde{Q}(\boldsymbol{t}) = \sum_{\boldsymbol{x}\in B_{s_x}} \widetilde{F}(\boldsymbol{x}) \cdot \widetilde{eq}(\boldsymbol{t},\boldsymbol{x}),$$

where $\widetilde{eq}(\boldsymbol{t},\boldsymbol{x}) = \prod_{i=1}^{s_x}(t_i \cdot x_i + (1-t_i)\cdot(1-x_i))$. Note that $Q(\boldsymbol{t})$ is a multivariate polynomial evaluates to $\widetilde{F}(\boldsymbol{t})$ for all $\boldsymbol{t} \in B_{s_x}$. Therefore, $Q(\boldsymbol{t})$ is a zero-polynomial if and only if $\widetilde{F}(\boldsymbol{x})$ evaluates to zero everywhere on $\boldsymbol{x} \in B_{s_x}$ (that is, the CCS relation is satisfied). To check whether $Q(\boldsymbol{t})$ is a zero-polynomial, it is sufficient to query its value on a random input $\boldsymbol{t} = \boldsymbol{\alpha}$ with an acceptable soundness error.

Lemma 3. $\Pr_{\boldsymbol{\alpha}}\{Q(\alpha) = 0 \mid \exists \boldsymbol{x} \in B_{s_x}\ s.t.\ \widetilde{F}(\boldsymbol{x}) \neq 0\} \leq \log m/|\mathbb{F}|$.

Proof. Refers to the proof of Lemma 4.3 in Spartan [17]. □

As a result, the prover runs the first sum-check protocol at step 4 on the polynomial $f(\boldsymbol{x})$ with the randomness $\boldsymbol{\alpha}, \boldsymbol{r}_x$ given by the verifier at step 1 and 2, where $\mathrm{sum}_x = 0$. To evaluate $f(\boldsymbol{r}_x)$, the verifier needs to know the value of $\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}_j(\boldsymbol{x},\boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y})$ for all $j \in [t]$, which can be reduced to another sum-check problem. Thus, the prover makes t separate claims to the sums on $\boldsymbol{y} \in B_{s_y}$ to the verifier at step 5. The verifier checks two facts accordingly: (1) c_x in sum-check#1 is consistent with the above claims (step 6), and (2) the t claims are valid.

Special Sound Protocol $\Pi_{\mathsf{CCCS}} = (\mathcal{P}, \mathcal{V})$ for $\mathcal{R}_{\mathsf{CCCS}}$

1. $\mathcal{V}$: Sample $\boldsymbol{\alpha} \leftarrow\!\!\$\ \mathbb{F}^{s_x}$ and send to $\mathcal{P}$.
2. $\mathcal{V}$: Sample $\boldsymbol{r}_x \leftarrow\!\!\$\ \mathbb{F}^{s_x}$.
3. $\mathcal{P}$: Compute $\tilde{z}(\boldsymbol{y}) = \widetilde{(\mathbf{w}, 1, \mathbf{io})}$.
4. **Sum-check#1.** Run $c_x \leftarrow \Pi_{\mathsf{sc}}(f, s_x, d+1, \mathrm{sum}_x, \boldsymbol{r}_x)$ where: $f(\boldsymbol{x}) =$
$$\widetilde{eq}(\boldsymbol{\alpha},\boldsymbol{x}) \cdot \left(\sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}_j(\boldsymbol{x},\boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}) \right) \right).$$
5. $\mathcal{P}$: Compute $\{\sigma_j\}_{j\in[t]}$ and send to $\mathcal{V}$, where for all $j \in [t]$:
$$\sigma_j = \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}_j(\boldsymbol{r}_x,\boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}).$$
6. $\mathcal{V}$: Compute $e \leftarrow \widetilde{eq}(\boldsymbol{\alpha},\boldsymbol{r}_x)$, and abort if:
$$c_x \neq e \cdot \sum_{i\in[q]} c_i \cdot \prod_{j\in S_i} \sigma_j.$$
7. $\mathcal{V}$: Sample $\delta \leftarrow\!\!\$\ \mathbb{F}$, and send to $\mathcal{P}$.
8. $\mathcal{V}$: Sample $\boldsymbol{r}_y \leftarrow\!\!\$\ \mathbb{F}^{s_y}$.
9. **Sum-check#2.** Run $c_y \leftarrow \Pi_{\mathsf{sc}}(g, s_y, 2, \mathrm{sum}_y, \boldsymbol{r}_y)$ where:
$$g(\boldsymbol{y}) = \sum_{j\in[t]} \delta^j \cdot \widetilde{M}_j(\boldsymbol{r}_x,\boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}).$$
10. $\mathcal{P}$: Compute $\epsilon, \{\theta_j\}_{j\in[t]}$ and send to $\mathcal{V}$, where:
$$\epsilon = \tilde{z}(\boldsymbol{r}_y),\ \theta_j = \widetilde{M}_j(\boldsymbol{r}_x,\boldsymbol{r}_y), j \in [t]$$
11. $\mathcal{V}$: Abort if:
$$c_y \neq \sum_{j\in[t]} \delta^j \cdot \theta_j \cdot \epsilon.$$
12. $\mathcal{P}$: Open the witness $\widetilde{\mathbf{w}}$.
13. $\mathcal{V}$: Check that
$$(1)\ \mathsf{Commit}(\mathsf{pp}, \widetilde{\mathbf{w}}) = C$$
$$(2)\ \epsilon = \tilde{z}(\boldsymbol{r}_y),\ \theta_j = \widetilde{M}_j(\boldsymbol{r}_x,\boldsymbol{r}_y).$$

The first fact is verified directly at step 6. For the second fact, a naive approach is to run t more times the sum-check protocol in parallel for t claims. A more elegant solution aggregates these claims by linear combination with weights $[\delta^1, ..., \delta^t]$ generated from a random δ. Consequently, the prover and verifier can run the sum-check protocol only once on the aggregated multi-variate polynomial $g(\boldsymbol{y})$ at step 9 with the randomness $\boldsymbol{r}_y$ and $\mathrm{sum}_y = \sum_{j\in[t]} \delta^j \cdot \sigma_j$. Likewisely, the prover makes claims to t evaluations $\{M_j(\boldsymbol{r}_x,\boldsymbol{r}_y)\}_{j=1}^t$ and one evaluation $\tilde{z}(\boldsymbol{r}_y)$ at steps 10. The verifier checks accordingly by running step 11 and computing the evaluations at step 13.

The security properties of the protocol Π_{CCCS} are guaranteed as follows:

- Completeness. Π_{CCCS} satisfies perfect completeness.
- Knowledge Soundness. Π_{CCCS} is a knowledge sound protocol for $\mathcal{R}_{\mathsf{CCCS}}$ if the commitment scheme $\mathsf{Commit}()$ satisfies the binding property. To prove it, let $\mathsf{Ext}_{\mathsf{CCCS}}$ be the PPT extractor for the protocol Π_{CCCS}. By rewinding the malicious prover $\mathcal{P}^*$ twice with different challenges ρ, ρ', $\mathsf{Ext}_{\mathsf{CCCS}}$ can compute a witness $\mathbf{w}'$ satisfying: (1) $\mathsf{Commit}(\mathsf{pp}, \widetilde{\mathbf{w}}') = C$ guaranteed by the binding property of commitment scheme and (2) the CCS relation guaranteed by the soundness of sum-check protocol [32] and Schwartz-Zippel lemma. By applying the union bound, we claim that the soundness error of Π_{CCCS} is at most $O(d \cdot \log m + t + \log n)/|\mathbb{F}|$.

B.2. Main Protocol

Construction 3 (Folding scheme for two instances)**.** *Let* $\mathsf{PC} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *denote an additively homomorphic polynomial commitment scheme for multilinear polynomials. The generator and the encoder are defined as follows.*

$\mathcal{G}(1^\lambda \to \mathsf{pp})$:

```
1 : Sample size bounds $m, n, N, l, t, q, d \in \mathbb{N}$ with $n = 2 \cdot (l+1)$.
2 : $\mathsf{pp}_{\mathsf{PC}} \leftarrow \mathsf{Gen}(1^\lambda, \log n - 1)$.
3 : Output $(m, n, N, l, t, q, d, \mathsf{pp}_{\mathsf{PC}})$.
```

```
$\mathcal{K}(\mathsf{pp}, \mathcal{S}, \mathcal{S}') \to (\mathsf{pk}, \mathsf{vk})$ :
1 : Parse $\mathcal{S}$ to obtain $\{\widetilde{M}_j\}_{j\in[t]}$.
2 : Parse $\mathcal{S}'$ to obtain $\{\widetilde{M}'_j\}_{j\in[t]}, \{S'_i\}_{i\in[q]}, \{c'_i\}_{i\in[q]}$.
3 : $\mathsf{pk} \leftarrow (\mathsf{pp}, (\{\widetilde{M}_j\}_{j\in[t]}, \{\widetilde{M}'_j\}_{j\in[t]}, \{S'_i\}_{i\in[q]}, \{c'_i\}_{i\in[q]}))$.
4 : $\mathsf{vk} \leftarrow \perp$.
5 : Output $(\mathsf{pk}, \mathsf{vk})$.
```

To distinguish, we mark the parts corresponding to the committed CCS instance in blue text. The verifier $\mathcal{V}$ takes as inputs an atomic CCS instance $(C, v_0, \mathbf{io}, \boldsymbol{r}_x, \boldsymbol{r}_y, \{v_j\}_{j\in[t]}, v_z)$ and a committed CCS instance $(C', \mathbf{io}')$. The prover $\mathcal{P}$, in addition to the two instances, takes witnesses $\widetilde{\mathbf{w}}$ and $\widetilde{\mathbf{w}'}$. Let $s_x = \log m$, $s_y = \log n$, $\tilde{z} = \widetilde{(\mathbf{w}, v_0, \mathbf{io})}$, and $\tilde{z}' = \widetilde{(\mathbf{w}', 1, \mathbf{io}')}$. The prover and the verifier proceed as follows.

1. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\gamma \leftarrow\$ \mathbb{F}$, $\boldsymbol{\alpha} \leftarrow\$ \mathbb{F}^{s_x}$, and sends them to $\mathcal{P}$.
2. $\mathcal{V}$: Sample $\boldsymbol{r}'_x \leftarrow\$ \mathbb{F}^{s_x}$.
3. $\mathcal{P}$: Compute $\tilde{z}(\boldsymbol{y}) = (\widetilde{\mathbf{w}}, v_0, \mathbf{io})$, $\tilde{z}'(\boldsymbol{y}) = (\widetilde{\mathbf{w}}', 1, \mathbf{io}')$.
4. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol#1 $c_x \leftarrow \Pi_{\mathrm{sc}}(f, s_x, d+1, \mathsf{sum}_x, \boldsymbol{r}'_x)$, where $\widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{x}), \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y})$ in $L_j(\boldsymbol{x})$ are used for updating challenge vectors:

$$\mathsf{sum}_x := \sum_{j\in[t]} \gamma^j \cdot v_j,$$

$$f(\boldsymbol{x}) := \left(\sum_{j\in[t]} \gamma^j \cdot L_j(\boldsymbol{x})\right) + \gamma^t \cdot Q(\boldsymbol{x}),$$

$$L_j(\boldsymbol{x}) := \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{x}) \cdot \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y})\right), \ j \in [t],$$

$$Q(\boldsymbol{x}) := \widetilde{eq}(\boldsymbol{\alpha}, \boldsymbol{x}) \cdot \left(\sum_{i\in[q]} c'_i \cdot \prod_{j\in S_i} \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}'_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \tilde{z}'(\boldsymbol{y})\right)\right),$$

5. $\mathcal{P} \to \mathcal{V}$: $(\{\sigma_j\}_{j\in[t]}, \{\sigma'_j\}_{j\in[t]})$, where:

$$\sigma_j = \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{y}), \forall j \in [t],$$
$$\sigma'_j = \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}'_j(\boldsymbol{r}'_x, \boldsymbol{y}) \cdot \tilde{z}'(\boldsymbol{y}), \forall j \in [t].$$

6. $\mathcal{V}$: Compute $e_1 \leftarrow \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{r}'_x)$ and $e_2 \leftarrow \widetilde{eq}(\boldsymbol{\alpha}, \boldsymbol{r}'_x)$, and abort if:

$$c_x \neq \left(\sum_{j\in[t]} \gamma^j \cdot e_1 \cdot \sigma_j\right) + \left(\gamma^t \cdot e_2 \cdot \sum_{i\in[q]} c'_i \cdot \prod_{j\in S_i} \sigma'_j\right).$$

7. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\delta \leftarrow\$ \mathbb{F}$, and sends it to $\mathcal{P}$.
8. $\mathcal{V}$: Sample $\boldsymbol{r}'_y \leftarrow\$ \mathbb{F}^{s_y}$.
9. $\mathcal{V} \leftrightarrow \mathcal{P}$: Run the sum-check protocol#2 $c_y \leftarrow \Pi_{\mathrm{sc}}(g, s_y, 2, \mathsf{sum}_y, \boldsymbol{r}'_y)$, where:

$$\mathsf{sum}_y := \sum_{j\in[t]} \delta^j \cdot \sigma_j + \delta^{t+1} \cdot v_z + \delta^{t+1} \cdot \sum_{j\in[t]} \delta^j \cdot \sigma'_j,$$
$$g(\boldsymbol{y}) := \sum_{j\in[t]} \delta^j \cdot R_j(\boldsymbol{y}) + \delta^{t+1} \cdot S(\boldsymbol{y}) + \delta^{t+1} \cdot \sum_{j\in[t]} \delta^j \cdot T_j(\boldsymbol{y}),$$
$$R_j(\boldsymbol{y}) = \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{y}), \ \forall j \in [t],$$
$$T_j(\boldsymbol{y}) = \widetilde{M}'_j(\boldsymbol{r}'_x, \boldsymbol{y}) \cdot \tilde{z}'(\boldsymbol{y}), \ \forall j \in [t],$$
$$S(\boldsymbol{y}) = \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}).$$

10. $\mathcal{P} \to \mathcal{V}$: $(\epsilon, \epsilon', \{\theta_j\}_{j\in[t]}, \{\theta'_j\}_{j\in[t]})$, where:

$$\epsilon = \tilde{z}(\boldsymbol{r}'_y), \quad \theta_j = \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{r}'_y), \ \forall j \in [t],$$
$$\epsilon' = \tilde{z}'(\boldsymbol{r}'_y), \quad \theta'_j = \widetilde{M}'_j(\boldsymbol{r}'_x, \boldsymbol{r}'_y), \ \forall j \in [t].$$

11. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\eta \leftarrow\$ \mathbb{F}$ and sends it to $\mathcal{P}$.
12. $\mathcal{V}$: Compute $e_3 \leftarrow \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{r}'_y)$, and abort if:

$$c_y \neq \sum_{j\in[t]} \delta^j \cdot e_3 \cdot \theta_j + \delta^{t+1} \cdot e_3 \cdot \epsilon + \delta^{t+1} \cdot \sum_{j\in[t]} \delta^j \cdot \theta'_j \cdot \epsilon'$$

13. $\mathcal{V}, \mathcal{P}$: Output the folded atomic CCS structure $\mathcal{S}^*$ containing
$$M^*_j = M_j + \eta \cdot M'_j$$
for all $j \in [t]$ and its instance $(C^*, v^*_0, \mathbf{io}^*, \boldsymbol{r}^*_x, \boldsymbol{r}^*_y, \{v^*_j\}_{j\in[t]}, v^*_z)$, where $\boldsymbol{r}^*_x = \boldsymbol{r}'_x, \boldsymbol{r}^*_y = \boldsymbol{r}'_y$, and for all $j \in [t]$:
$$C^* \leftarrow C + \eta \cdot C',$$
$$\mathsf{io}^* \leftarrow \mathsf{io} + \eta \cdot \mathsf{io}',$$
$$v^*_0 \leftarrow v_0 + \eta \cdot 1,$$
$$v^*_j \leftarrow \theta_j + \eta \cdot \theta'_j,$$
$$v^*_z \leftarrow \epsilon + \eta \cdot \epsilon'.$$
14. $\mathcal{P}$: Output the folded witness $\mathbf{w} + \eta \cdot \mathbf{w}'$.

Concretely, we provide a folding scheme for two instances in specific relations $\mathcal{R}$ and $\mathcal{R}'$, where $\mathcal{R}$ and $\mathcal{R}'$ are ACCS and CCCS relations with different structures $\mathcal{S}$ and $\mathcal{S}'$ respectively. We assume $\mathcal{S}$ and $\mathcal{S}'$ share the same size bounds m, n, N, l, t, but contain different multilinear polynomials, $\{\widetilde{M}_j\}_{j\in[t]}$ and $\{\widetilde{M}'_j\}_{j\in[t]}$. The security of this folding scheme is discussed below.

Theorem 4. *(Folding scheme for committed CCS). Construction 3 is a public coin folding scheme for $(\mathcal{R}, \mathcal{R}')$ with perfect completeness and knowledge soundness.*

Proof sketch. It is trivial to prove that Construction 3 satisfies completeness. For the knowledge soundness, we prove that Construction 3 is knowledge sound if the commitment scheme Commit() satisfies the binding property. Concretely, if there exists an adversary $\mathcal{A}$ that succeeds in producing valid proof with non-negligible probability, we show that a polynomial time extractor Ext that outputs the witness with non-negligible probability can be constructed. According to the conclusion in [38], given a $2^{s_x+s_y}$-tree of transcripts, we can either find a pair of commitments breaking the binding property or construct an expected extractor outputting the witness.

B.3. Security Proofs for Folding Scheme

In this section, we present the formal security proofs of our generic foldings scheme, including perfect completeness, knowledge soundness, and honest verifier zero-knowledge. For the former two properties, we mainly refer to the proof of the HyperNova [14].

Lemma 4. *(Perfect Completeness). Construction 3 satisfies perfect completeness.*

Proof. Consider public parameters $\mathsf{pp} = (m, n, N, l, t, q, d, \mathsf{pp}_{\mathsf{PC}}) \leftarrow \mathcal{G}(1^\lambda)$ and let $s_x = \log m$ and $s_y = \log n$. Consider arbitrary structures

$$\begin{aligned}\mathcal{S} &= \{\widetilde{M_j}\}_{j\in[t]} \leftarrow \mathcal{A}(\mathsf{pp}),\\ \mathcal{S}' &= \{\widetilde{M'_j}\}_{j\in[t]}, \{S'_i\}_{i\in[q]}, \{c'_i\}_{i\in[q]} \leftarrow \mathcal{A}(\mathsf{pp}).\end{aligned}$$

Consider prover and verifier key $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathcal{S}, \mathcal{S}')$. Suppose the prover and verifier are provided an atomic CCS instance

$$(C, v_0, \mathsf{io}, \boldsymbol{r}_x, \boldsymbol{r}_y, \{v_j\}_{j\in[t]}, v_z),$$

and a committed CCS instance

$$(C', \mathsf{io}').$$

1. Sum-check protocol#1: suppose the prover is additionally provided the corresponding satisfying witnesses $\widetilde{\mathbf{w}}$ and $\widetilde{\mathbf{w}}'$. Since the input atomic CCS instance-witness pair is satisfying, we have, for $\tilde{z} = \widetilde{(\mathbf{w}, v_0, \mathbf{io})}$, that

$$\begin{aligned} v_j &= \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{r}_x, \boldsymbol{y}) \\ &= \sum_{\boldsymbol{x}\in B_{s_x}} \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{x}) \cdot \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{x}, \boldsymbol{y}) \right) \\ &= \sum_{\boldsymbol{x}\in B_{s_x}} L_j(\boldsymbol{x}), \forall j \in [t]. \end{aligned}$$

Moreover, since the input committed CCS instance-witness pair is satisfying, we have, for $\tilde{z}'(\boldsymbol{y}) = \widetilde{(\mathbf{w}', 1, \mathbf{io}')}(\boldsymbol{y})$, that

$$0 = \sum_{i\in[q]} c'_i \cdot \prod_{j\in S'_i} \left(\sum_{y\in B_{s_y}} \widetilde{M}'_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \tilde{z}'(\boldsymbol{y}) \right), \forall \boldsymbol{x} \in B_{s_x}.$$

Treating the right-hand side of the above equation as a polynomial in $\boldsymbol{x}$, because it is multilinear and vanishes on all $\boldsymbol{x} \in B_{s_x}$, we have that it must be the zero polynomial. Therefore, we have, for $\boldsymbol{\alpha}$ sampled by the verifier, that

$$\begin{aligned} 0 &= \sum_{i\in[q]} c'_i \cdot \prod_{j\in S'_i} \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}'_j(\boldsymbol{\alpha}, \boldsymbol{y}) \cdot \tilde{z}'(\boldsymbol{y}) \right) \\ &= \sum_{\boldsymbol{x}\in B_{s_x}} \widetilde{eq}(\boldsymbol{\alpha}, \boldsymbol{x}) \cdot \left(\sum_{i\in[q]} c'_i \prod_{j\in S'_i} \left(\sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}'_j(\boldsymbol{x}, \boldsymbol{y}) \cdot \tilde{z}'(\boldsymbol{y}) \right) \right) \\ &= \sum_{\boldsymbol{x}\in B_{s_x}} Q(\boldsymbol{x}). \end{aligned}$$

For γ sampled by the verifier, by linearity, we have that

$$\begin{aligned} \sum_{j\in[t]} \gamma^j \cdot v_j &= \sum_{\boldsymbol{x}\in B_{s_x}} \left(\left(\sum_{j\in[t]} \gamma^j \cdot L_j(\boldsymbol{x}) \right) + \gamma^{t+1} \cdot Q(\boldsymbol{x}) \right) \\ &= \sum_{\boldsymbol{x}\in B_{s_x}} f(\boldsymbol{x}). \end{aligned}$$

Therefore, by the perfect completeness of the sum-check protocol, we have for $e_1 = \widetilde{eq}(\boldsymbol{r}_x, \boldsymbol{r}'_x), e_2 = \widetilde{eq}(\boldsymbol{\alpha}, \boldsymbol{r}'_x)$ and

$$\begin{aligned} \sigma_j &= \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{y}), \forall j \in [t], \\ \sigma'_j &= \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}'_j(\boldsymbol{r}'_x, \boldsymbol{y}) \cdot \tilde{z}'(\boldsymbol{y}), \forall j \in [t], \end{aligned}$$

that

$$\begin{aligned} c_x &= f(\boldsymbol{r}'_x) \\ &= \left(\sum_{j\in[t]} \gamma^j \cdot L_j(\boldsymbol{r}'_x) \right) + \gamma^{t+1} \cdot Q(\boldsymbol{r}'_x) \\ &= \left(\sum_{j\in[t]} \gamma^j \cdot e_1 \cdot \sigma_j \right) + \gamma^{t+1} \cdot e_2 \cdot \sum_{i\in[q]} c'_i \cdot \prod_{j\in S'_i} \sigma'_j. \end{aligned}$$

2. Sum-check protocol#2: According to the results of sum-check protocol#1, for $\tilde{z}(\boldsymbol{x}), \tilde{z}'(\boldsymbol{x})$ and sampled $\boldsymbol{r}'_x$, we have

$$\begin{aligned} \sigma_j &= \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{y}) \\ &= \sum_{\boldsymbol{y}\in B_{s_y}} R_j(\boldsymbol{y}), \forall j \in [t], \\ v_z &= \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y, \boldsymbol{y}) \cdot \tilde{z}(\boldsymbol{y}) \\ \sigma'_j &= \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}'_j(\boldsymbol{r}'_x, \boldsymbol{y}) \cdot \tilde{z}'(\boldsymbol{y}), \\ &= \sum_{\boldsymbol{y}\in B_{s_y}} T_j(\boldsymbol{y}), \forall j \in [t]. \end{aligned}$$

For δ sampled by the verifier, by linearity, we have that

$$\begin{aligned} &\sum_{j\in[t]} \delta^j \cdot \sigma_j + \delta^{t+1} \cdot v_z + \delta^{t+1} \cdot \sum_{j\in[t]} \delta^j \cdot \sigma'_j \\ =\ &\sum_{j\in[t]} \delta^j \cdot R_j(\boldsymbol{y}) + \delta^{t+1} \cdot S(\boldsymbol{y}) + \delta^{t+1} \cdot \sum_{j\in[t]} \delta^j \cdot T_j(\boldsymbol{y}) \\ =\ &\sum_{j\in[t]} g(\boldsymbol{y}). \end{aligned}$$

Therefore, by the perfect completeness of the sum-check protocol, we have for

$$\begin{aligned} &\epsilon = \tilde{z}(\boldsymbol{r}'_y), \\ &\epsilon' = \tilde{z}'(\boldsymbol{r}'_y), \\ &\theta_j = \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{r}'_y), \forall j \in [t], \\ &\theta'_j = \widetilde{M}'_j(\boldsymbol{r}'_x, \boldsymbol{r}'_y), \forall j \in [t], \end{aligned}$$

that

$$\begin{aligned} c_y &= g(\boldsymbol{r}'_y) \\ &= \sum_{j\in[t]} \delta^j \cdot R_j(\boldsymbol{r}'_y) + \delta^{t+1}\cdot S(\boldsymbol{y}) + \delta^{t+1}\cdot \sum_{j\in[t]} \delta^j \cdot T_j(\boldsymbol{r}'_y) \\ &= \sum_{j\in[t]} \delta^j \cdot e_3 \cdot \theta_j + \delta^{t+1}\cdot \epsilon + \delta^{t+1}\cdot \sum_{j\in[t]} \delta^j \cdot \theta_j \cdot \epsilon'. \end{aligned}$$

The above two steps imply that the verifier will not abort. Now, consider the atomic CCS instance obtained from $\mathcal{R}'$ as

$$(C', 1, \mathsf{io}', \boldsymbol{r}'_x, \boldsymbol{r}'_y, \{\theta'_j\}_{j\in[t]}, \epsilon').$$

By the precondition that the committed CCS instance $(C', \mathbf{io}')$ is satisfied by $\widetilde{\mathbf{w}}'$ and by the definition of $\{\theta'_j\}_{j\in[t]}, \epsilon'$ we have that this new atomic CCS instance is satisfied by the witness $\widetilde{\mathbf{w}}'$.

Therefore, for random η sampled by the verifier, and folded structure $\mathcal{S}^*$ with $\{M^*_j = M_j + \eta \cdot M'_j\}_{j=1}^t$, folded instance $C^* = C + \eta \cdot C'$, $v^*_0 = v_0 + \eta \cdot 1$, $\mathbf{io}^* = \mathbf{io} + \eta \cdot \mathbf{io}'$, $v^*_j = \theta_j + \eta \cdot \theta'_j$, $v^*_z = \epsilon_j + \eta \cdot \epsilon'_j$, we have that the output folded atomic CCS instance

$$(C^*, v^*_0, \mathsf{io}^*, \boldsymbol{r}'_x, \boldsymbol{r}'_y, \{v^*_j\}_{j\in[t]}, v^*_z).$$

is satisfied by the witness $\widetilde{\mathbf{w}}^* \leftarrow \widetilde{\mathbf{w}} + \eta \cdot \widetilde{\mathbf{w}}'$ under the structure $\mathcal{S}^*$ by the linearity and the additive homomorphism property of the commitment scheme. □

Lemma 5. *(Knowledge Soundness). Construction 3 satisfies knowledge soundness.*

Proof. Consider an adversary $\mathcal{A}$ that adaptively picks the structures and instances, and a malicious prover $\mathcal{P}^*$ that succeeds with probability ϵ. Let $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$. Suppose on input pp and random tape η, the adversary $\mathcal{A}$ picks two structures

$$\begin{aligned} &\mathcal{S} = \{\widetilde{M_j}\}_{j\in[t]}, \\ &\mathcal{S}' = \{\widetilde{M'_j}\}_{j\in[t]}, \{S'_i\}_{i\in[q]}, \{c'_i\}_{i\in[q]}. \end{aligned}$$

a new committed CCS instance

$$\mathbf{u} = (C, v_0, \mathsf{io}, \boldsymbol{r}_x, \boldsymbol{r}_y, \{v_j\}_{j\in[t]}, v_z),$$

and committed CCS instance

$$\mathbf{u}' = (C', \mathsf{io}'),$$

and some auxiliary state st.
1. Extraction Algorithm: we construct an expected-polynomial time extractor Ext that succeeds with probability $\epsilon - \mathrm{negl}(\lambda)$ in obtaining satisfying witnesses for the original instances as follows.

```
Ext(pp, ρ):
1 :  Obtain the output tuple from $\mathcal{A}$:
         $(\mathcal{S}, \mathcal{S}', \mathbf{u}, \mathbf{u}', \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \rho)$.
2 :  Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathcal{S}, \mathcal{S}')$.
3 :  Run the folding interaction#1
         $(\mathcal{S}^*_1, \mathbf{u}^*_1, \widetilde{\mathbf{w}}^*_1) \leftarrow \langle \mathcal{P}^*, \mathcal{V}\rangle((\mathsf{pk}, \mathsf{vk}), \mathcal{S}, \mathcal{S}', \mathbf{u}, \mathbf{u}', \mathsf{st})$
4 :  *once* with the final verifier challenge $\eta_1 \leftarrow\!\!\$\ \mathbb{F}$.
5 :  Abort if $(\mathsf{pp}, \mathcal{S}^*_1, \mathbf{u}^*_1, \widetilde{\mathbf{w}}^*_1) \notin \mathcal{R}_{\mathsf{ACCS}}$.
6 :  Run the folding interaction#2
         $(\mathcal{S}^*_2, \mathbf{u}^*_2, \widetilde{\mathbf{w}}^*_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V}\rangle((\mathsf{pk}, \mathsf{vk}), \mathcal{S}, \mathcal{S}', \mathbf{u}, \mathbf{u}', \mathsf{st}))$
7 :  with a different verifier's final challenge $\eta_2 \leftarrow\!\!\$\ \mathbb{F}$ while maintaining
     the same prior randomness. Keep doing so until
         $(\mathsf{pp}, \mathcal{S}^*_2, \mathbf{u}^*_2, \widetilde{\mathbf{w}}^*_2) \in \mathcal{R}_{\mathsf{ACCS}}$.
8 :  Abort if $\eta_1 = \eta_2$ or $\mathcal{S}^*_1 \neq \mathcal{S}^*_2$.
9 :  Interpolating points $(\eta_1, \widetilde{\mathbf{w}}^*_1)$ and $(\eta_2, \widetilde{\mathbf{w}}^*_2)$, retrieve the witness
     polynomials $\widetilde{\mathbf{w}}$ and $\widetilde{\mathbf{w}}'$ such that for $i \in \{1, 2\}$
         $\widetilde{\mathbf{w}} + \eta_i \cdot \widetilde{\mathbf{w}}' = \widetilde{\mathbf{w}}^*_i$.
10 : Output $(\widetilde{\mathbf{w}}, \widetilde{\mathbf{w}}')$.
```

We first demonstrate that the extractor Ext runs in expected polynomial time. Observe that Ext runs the folding interaction#1 once, and if it does not abort, keeps rerunning the folding interaction#2 until $\mathcal{P}^*$ succeeds. Let W denote the event that the extractor does not abort at step 4, and $\bar{W}$ denotes that the event W does not happen. Define the number of folding interactions Ext runs in total as a variable X (i.e., number of rewinds). We can calculate its expectation as

$$\begin{aligned} \mathbb{E}[X] &= \Pr[W] \cdot (1 + \frac{1}{\Pr[\langle \mathcal{P}^*, \mathcal{V}\rangle \text{ succeeds}]}) + \Pr[\bar{W}] \cdot 1 \\ &= \epsilon \cdot (1 + \frac{1}{\epsilon}) + (1 - \epsilon) \cdot 1 = 2. \end{aligned}$$

Therefore, we have the extractor run in the expected polynomial time.
2. Advantage Analysis: We now analyze Ext's success probability. We must demonstrate that Ext succeeds in producing $\widetilde{\mathbf{w}}$ and $\widetilde{\mathbf{w}}'$ such that

$$(\mathsf{pp}, \mathcal{S}, \mathbf{u}, \widetilde{\mathbf{w}}) \in \mathcal{R}_{\mathsf{ACCS}} \text{ and } (\mathsf{pp}, \mathcal{S}', \mathbf{u}', \widetilde{\mathbf{w}}') \in \mathcal{R}_{\mathsf{CCCS}}$$

, with probability $\epsilon - \mathrm{negl}(\lambda)$.

To do so, we first show that the extractor successfully produces *some* output (i.e., does not abort) in less than $\sqrt[3]{|\mathbb{F}|}$ rewinding steps with probability $\epsilon - \mathrm{negl}(\lambda)$. Indeed, by the malicious prover's success probability, we have that the extractor does not abort at step (4) with probability ϵ. Given that the extractor does not abort at step (4), by Markov's inequality, we have that the extractor rewinds more than $\sqrt[3]{|\mathbb{F}|}$ times with probability

$$\Pr[X \geq \sqrt[3]{|\mathbb{F}|}] \leq \frac{\mathbb{E}[X]}{\sqrt[3]{|\mathbb{F}|}} = \frac{2}{\sqrt[3]{|\mathbb{F}|}},$$

where X is the random variable of the number of running folding interactions. Thus, the probability that the extractor does not abort at step (4) and requires less than $\sqrt[3]{|\mathbb{F}|}$ rewinds is $(1-2/\sqrt[3]{|\mathbb{F}|})\cdot\epsilon$.

Now, suppose that the extractor does not abort at step (4) and requires less than $\sqrt[3]{|\mathbb{F}|}$ rewinds. This ensures that the extractor tests at most $\sqrt[3]{|\mathbb{F}|}$ values for η. Since the challenges are sampled uniformly in random form $|\mathbb{F}|$, the probability that $\rho^{(1)}\neq\rho^{(2)}$ is $1-\sqrt[3]{|\mathbb{F}|}^2/|\mathbb{F}|$. Therefore, assuming $\sqrt[3]{|\mathbb{F}|}^2\geq 2$, we have that the probability the extractor successfully produces some output under $\sqrt[3]{|\mathbb{F}|}$ rewinding steps is

$$\begin{aligned}
&\Pr[X<\sqrt[3]{|\mathbb{F}|}]\cdot\Pr[\rho^{(1)}\neq\rho^{(2)}]\\
=\;&(1-\frac{2}{\sqrt[3]{|\mathbb{F}|}})\cdot\epsilon\cdot(1-\frac{\sqrt[3]{|\mathbb{F}|}^2}{|\mathbb{F}|})\\
=\;&(1-\frac{2}{\sqrt[3]{|\mathbb{F}|}}-\frac{\sqrt[3]{|\mathbb{F}|}^2}{|\mathbb{F}|}+\frac{2}{|\mathbb{F}|})\\
=\;&\epsilon-\mathrm{negl}(\lambda).
\end{aligned}$$

Next, if the extractor does not abort, we show that the extractor succeeds in producing satisfying witnesses with probability $1-\mathrm{negl}(\lambda)$. This brings the overall extractor success probability to $(\epsilon-\mathrm{negl}(\lambda))\cdot(1-\mathrm{negl}(\lambda))$.

For $i\in\{1,2\}$, let $\mathbf{u}_i^*=(C_i^*,v_{0,i}^*,\mathbf{io}_i^*,r_{x,i}^*,v_{1,i}^*,...,v_{t,i}^*,v_{z,i}^*)$. We first show that the retrieved polynomials are valid openings to the corresponding commitments in the instance. For $i\in\{1,2\}$, since $\widetilde{\mathbf{w}}_i^*$ is a satisfying witness, by construction,

$$\begin{aligned}
&\mathsf{Commit}(\mathsf{pp},\widetilde{\mathbf{w}})+\eta_i\cdot\mathsf{Commit}(\mathsf{pp},\widetilde{\mathbf{w}}')\\
=\;&\mathsf{Commit}(\mathsf{pp},\widetilde{\mathbf{w}}+\eta_i\cdot\widetilde{\mathbf{w}}')\\
=\;&\mathsf{Commit}(\mathsf{pp},\widetilde{\mathbf{w}}_i^*)\\
=\;&C_i^*\\
=\;&C+\eta_i\cdot C'.
\end{aligned}$$

Interpolating, we have that

$$\begin{aligned}
\mathsf{Commit}(\mathsf{pp},\widetilde{\mathbf{w}})&=C, &(6)\\
\mathsf{Commit}(\mathsf{pp},\widetilde{\mathbf{w}}')&=C'. &(7)
\end{aligned}$$

Next, we must argue that $\widetilde{\mathbf{w}}$ and $\widetilde{\mathbf{w}}'$ satisfy the remainder of the instances $(\mathcal{S},\mathbf{u})$ and $(\mathcal{S}',\varphi')$ respectively.

Consider $\{\theta_j\}_{j\in[t]},\{\theta_j'\}_{j\in[t]}$ and ϵ,ϵ' sent by the prover which by the extractor's construction are identical across all executions of the interaction. By the verifier's computation we have that for $i\in\{1,2\}$ and all $j\in[t]$

$$\begin{aligned}
v_{j,i}&=\theta_j+\eta_i\cdot\theta_j', &(8)\\
v_{z,i}&=\epsilon+\eta_i\cdot\epsilon'. &(9)
\end{aligned}$$

Now, because $\widetilde{\mathbf{w}}_i^*$ is a satisfying witness, for $i\in\{1,2\}$ we have for all $j\in[t]$ that

$$\begin{aligned}
v_{j,i}&=\widetilde{M}_{j,i}^*(\boldsymbol{r}_x',\boldsymbol{r}_y'),\\
v_{z,i}&=\tilde{z}_i^*(\boldsymbol{r}_y'),
\end{aligned}$$

where $\widetilde{M}_{j,i}^*=\widetilde{M}_j+\eta_i\cdot\widetilde{M}_j$, $\tilde{z}_i^*=\widetilde{(\mathbf{w}_i^*,v_{0,i}^*,\mathbf{io}_i^*)}=\tilde{z}+\eta\cdot\tilde{z}'$.

Meanwhile, according to equations (8) and (9), for $i\in\{1,2\}$ and $j\in[t]$, we have

$$\begin{aligned}
\theta_j+\eta_i\cdot\theta_j'=v_{j,i}&=\widetilde{M}_j(\boldsymbol{r}_x',\boldsymbol{r}_y')+\eta_i\cdot\widetilde{M}_j'(\boldsymbol{r}_x',\boldsymbol{r}_y'),\\
\epsilon+\eta_i\cdot\epsilon'=v_{z,i}&=\tilde{z}(\boldsymbol{r}_y')+\eta_i\cdot\tilde{z}'(\boldsymbol{r}_y'),
\end{aligned}$$

where $\tilde{z}=\widetilde{(\mathbf{w}_i,v_{0,i},\mathbf{io}_i)}$ and $\tilde{z}'=\widetilde{(\mathbf{w}',1,\mathbf{io}')}$. Interpolating, we have that, for all $j\in[t]$

$$\begin{aligned}
&\theta_j=\widetilde{M}_j(\boldsymbol{r}_x',\boldsymbol{r}_y'),\\
&\theta_j'=\widetilde{M}_j'(\boldsymbol{r}_x',\boldsymbol{r}_y'),\\
&\epsilon=\tilde{z}(\boldsymbol{r}_y'),\\
&\epsilon'=\tilde{z}'(\boldsymbol{r}_y').
\end{aligned}$$

Thus, because the verifier does not abort at step 11, we have that

$$\begin{aligned}
c_y&=\sum_{j\in[t]}\delta^j\cdot e_3\cdot\theta_j+\delta^{t+1}\cdot e_3\cdot\epsilon+\delta^{t+1}\cdot\sum_{j\in[t]}\delta^j\cdot\theta_j'\cdot\epsilon'\\
&=\sum_{j\in[t]}\delta^j\cdot\widetilde{eq}(\boldsymbol{r}_y,\boldsymbol{r}_y')\cdot\theta_j+\delta^{t+1}\cdot\widetilde{eq}(\boldsymbol{r}_y,\boldsymbol{r}_y')\cdot\epsilon\\
&\quad+\delta^{t+1}\cdot\sum_{j\in[t]}\delta^j\cdot\theta_j'\cdot\epsilon'\\
&=\sum_{j\in[t]}\delta^j\cdot\widetilde{eq}(\boldsymbol{r}_y,\boldsymbol{r}_y')\cdot\widetilde{M}_j(\boldsymbol{r}_x',\boldsymbol{r}_y')+\delta^{t+1}\cdot\widetilde{eq}(\boldsymbol{r}_y,\boldsymbol{r}_y')\cdot\tilde{z}(\boldsymbol{r}_y')\\
&\quad+\delta^{t+1}\cdot\sum_{j\in[t]}\delta^j\cdot\widetilde{M}_j'(\boldsymbol{r}_x',\boldsymbol{r}_y')\cdot\tilde{z}'(\boldsymbol{r}_y')\\
&=\sum_{j\in[t]}\delta^j\cdot R_j(\boldsymbol{r}_y')+\delta^{t+1}\cdot S(\boldsymbol{r}_y')+\delta^{t+1}\cdot\sum_{j\in[t]}\delta^j\cdot T_j(\boldsymbol{r}_y')\\
&=g(\boldsymbol{r}_y'),
\end{aligned}$$

by the soundness of the sum-check protocol#2, this implies that with probability $1-O(d\cdot s_y)/|\mathbb{F}|=1-\mathrm{negl}(\lambda)$ over the choice of $\boldsymbol{r}_y'$,

$$\begin{aligned}
&\sum_{j\in[t]}\delta^j\cdot\sigma_j+\delta^{t+1}\cdot v_z+\delta^{t+1}\cdot\sum_{j\in[t]}\delta^j\cdot\sigma_j'\\
=\;&\sum_{\boldsymbol{y}\in B_{s_y}}g(\boldsymbol{y})\\
=\;&\sum_{\boldsymbol{y}\in B_{s_y}}\left(\sum_{j\in[t]}\delta^j\cdot R_j(\boldsymbol{y})+\delta^{t+1}\cdot S(\boldsymbol{y})+\delta^{t+1}\cdot\sum_{j\in[t]}\delta^j\cdot T_j(\boldsymbol{y})\right)\\
=\;&\sum_{j\in[t]}\delta^j\cdot\sum_{\boldsymbol{y}\in B_{s_y}}R_j(\boldsymbol{y})+\delta^{t+1}\cdot\sum_{\boldsymbol{y}\in B_{s_y}}S(\boldsymbol{y})\\
&+\delta^{t+1}\cdot\sum_{j\in[t]}\delta^j\cdot\sum_{\boldsymbol{y}\in B_{s_y}}T_j(\boldsymbol{y}).
\end{aligned}$$

By the Schwartz-Zippel lemma [Sch80], this implies that with probability $1-O(t)/|\mathbb{F}|=1-\mathrm{negl}(\lambda)$ over the choice of δ, for all $j \in [t]$, we have

$$\begin{aligned}
\sigma_j &= \sum_{\boldsymbol{y}\in B_{s_y}} R_j(\boldsymbol{y}) = \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y,\boldsymbol{y})\cdot \widetilde{M}_j(\boldsymbol{r}'_x,\boldsymbol{y}),\\
v_z &= \sum_{\boldsymbol{y}\in B_{s_y}} S(\boldsymbol{y}) = \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{eq}(\boldsymbol{r}_y,\boldsymbol{y})\cdot \tilde{z}(\boldsymbol{y}),\\
\sigma'_j &= \sum_{\boldsymbol{y}\in B_{s_y}} T_j(\boldsymbol{y}) = \sum_{\boldsymbol{y}\in B_{s_y}} \widetilde{M}'_j(\boldsymbol{r}'_x,\boldsymbol{y})\cdot \tilde{z}'(\boldsymbol{y}).
\end{aligned}$$

Thus, because the verifier does not abort at step 5, we have that

$$\begin{aligned}
& c_x\\
=& \left(\sum_{j\in[t]}\gamma^j\cdot e_1\cdot\sigma_j\right) + \left(\gamma^{t+1}\cdot e_2\cdot\sum_{i\in[q]}c'_i\cdot\prod_{j\in S_i}\sigma_j\right)\\
=& \left(\sum_{j\in[t]}\gamma^j\cdot\widetilde{eq}(\boldsymbol{r}_x,\boldsymbol{r}'_x)\cdot\sigma_j\right)\\
+& \left(\gamma^{t+1}\cdot\widetilde{eq}(\boldsymbol{\alpha},\boldsymbol{r}'_x)\cdot\sum_{i\in[q]}c'_i\cdot\prod_{j\in S_i}\theta_j\right)\\
=& \left(\sum_{j\in[t]}\gamma^j\cdot\widetilde{eq}(\boldsymbol{r}_x,\boldsymbol{r}'_x)\cdot\sum_{\boldsymbol{y}\in B_{s_y}}\widetilde{eq}(\boldsymbol{r}_y,\boldsymbol{y})\cdot\widetilde{M}_j(\boldsymbol{r}'_x,\boldsymbol{y})\right)\\
+& \left(\gamma^{t+1}\cdot\widetilde{eq}(\boldsymbol{\alpha},\boldsymbol{r}'_x)\cdot\sum_{i\in[q]}c_i\cdot\prod_{j\in S_i}\sum_{\boldsymbol{y}\in B_{s_y}}\widetilde{M}'_j(\boldsymbol{r}'_x,\boldsymbol{y})\cdot\tilde{z}'(\boldsymbol{y})\right)\\
=& \sum_{j\in[t]}\gamma^j\cdot L_j(\boldsymbol{r}'_x)+\gamma^{t+1}\cdot Q(\boldsymbol{r}'_x)\\
=& f(\boldsymbol{r}'_x),
\end{aligned}$$

by the soundness of the sum-check protocol#1, this implies that with probability $1-O(d\cdot s_x)/|\mathbb{F}| = 1-\mathrm{negl}(\lambda)$ over the choice of $\boldsymbol{r}'_x$,

$$\begin{aligned}
& \sum_{j\in[t]}\gamma^j\cdot v_j+\gamma^{t+1}\cdot 0\\
=& \sum_{\boldsymbol{x}\in B_{s_x}} f(x)\\
=& \sum_{\boldsymbol{x}\in B_{s_x}}\left(\left(\sum_{j\in[t]}\gamma^j\cdot L_j(\boldsymbol{x})\right)+\gamma^{t+1}\cdot Q(\boldsymbol{x})\right)\\
=& \sum_{\boldsymbol{x}\in B_{s_x}}\gamma^j\cdot\left(\sum_{j\in[t]}L_j(\boldsymbol{x})\right)+\gamma^{t+1}\cdot\sum_{\boldsymbol{x}\in B_{s_x}}Q(x).
\end{aligned}$$

By the Schwartz-Zippel lemma [Sch80], this implies that with probability $1-O(t)/|\mathbb{F}| = 1-\mathrm{negl}(\lambda)$ over the choice of γ, for all $j\in[t]$, we have

$$v_j = \sum_{\boldsymbol{x}\in B_{s_x}} L_j(\boldsymbol{x}),\ \text{and}\ 0 = \sum_{\boldsymbol{x}\in B_{s_x}} Q(x).$$

Therefore,

$$\begin{aligned}
v_j &= \sum_{\boldsymbol{x}\in B_{s_x}} L_j(\boldsymbol{x})\\
&= \sum_{\boldsymbol{x}\in B_{s_x}}\widetilde{eq}(\boldsymbol{r}_x,\boldsymbol{x})\cdot\left(\sum_{\boldsymbol{y}\in B_s}\widetilde{eq}(\boldsymbol{r}_y,\boldsymbol{y})\cdot\widetilde{M}_j(\boldsymbol{x},\boldsymbol{y})\right)\\
&= \widetilde{M}_j(\boldsymbol{r}_x,\boldsymbol{r}_y).
\end{aligned}$$

This implies that $\widetilde{\mathbf{w}}$ is a satisfying witness to $(\mathcal{S},\mathbf{u})$. Finally, we have that

$$\begin{aligned}
0 &= \sum_{\boldsymbol{x}\in B_{s_x}} Q(x)\\
&= \sum_{\boldsymbol{x}\in B_{s_x}}\widetilde{eq}(\boldsymbol{\alpha},\boldsymbol{x})\cdot\left(\sum_{i\in[q]}c'_i\cdot\prod_{j\in S_i}\left(\sum_{\boldsymbol{y}\in B_{s_y}}\widetilde{M}'_j(\boldsymbol{x},\boldsymbol{y})\cdot\tilde{z}'(\boldsymbol{y})\right)\right)\\
&= \sum_{i\in[q]}c'_i\cdot\prod_{j\in S_i}\left(\sum_{\boldsymbol{y}\in B_{s_y}}\widetilde{M}'_j(\boldsymbol{\alpha},\boldsymbol{y})\cdot\tilde{z}'(\boldsymbol{y})\right).
\end{aligned}$$

By the Schwartz-Zippel lemma, this implies that with probability $1-s_x/|\mathbb{F}| = 1-\mathrm{negl}(\lambda)$ over the choice of $\boldsymbol{\alpha}$, we have that for all $\boldsymbol{x}\in B_{s_x}$

$$0 = \sum_{i\in[q]}c'_i\cdot\prod_{j\in S_i}\left(\sum_{\boldsymbol{y}\in B_{s_y}}\widetilde{M}'_j(\boldsymbol{x},\boldsymbol{y})\cdot\tilde{z}'(\boldsymbol{y})\right).$$

This implies that $\widetilde{\mathbf{w}}'$ is a satisfying witness to $(\mathcal{S}',\mathbf{u}')$. Thus, if the extractor does not abort, it succeeds in producing satisfying witness $\widetilde{\mathbf{w}},\widetilde{\mathbf{w}}'$ with probability $1-\mathrm{negl}(\lambda)$. □

Lemma 6. *(Honest Verifier Zero Knowledge). Construction 3 satisfies honest verifier zero knowledge.*

Proof. Consider an adversary $\mathcal{A}$ that adaptively picks the structures and instances. Let $\mathsf{pp}\leftarrow\mathcal{G}(1^\lambda)$. Suppose on input pp, the adversary $\mathcal{A}$ picks two structures

$$\begin{aligned}
&\mathcal{S} = \{\widetilde{M}_j\}_{j\in[t]},\\
&\mathcal{S}' = \{\widetilde{M}'_j\}_{j\in[t]},\{S'_i\}_{i\in[q]},\{c'_i\}_{i\in[q]},
\end{aligned}$$

a new committed CCS instance-witness pair

$$(\mathbf{u},\mathbf{w}) = (C, v_0, \mathsf{io}, \boldsymbol{r}_x, \boldsymbol{r}_y, \{v_j\}_{j\in[t]}, v_z, \mathbf{w}),$$

and committed CCS instance-witness pair

$$(\mathbf{u}',\mathbf{w}') = (C', \mathsf{io}'.\mathbf{w}).$$

With the keys generated by $(\mathsf{pk},\mathsf{vk})\leftarrow\mathcal{K}(\mathsf{pp},\mathcal{S},\mathcal{S}')$, the non-deterministic function trace produces an interaction transcript tr between honest $\mathcal{P}$ and $\mathcal{V}$ of Π_{fold} on input $((\mathsf{pk},\mathsf{vk}),(\mathbf{u},\mathbf{u}'),(\mathbf{w},\mathbf{w}'))$.

Next, we construct a PPT simulator Sim producing the trace $\hat{\mathsf{tr}}$ with indistinguishable distribution from tr with the input of $(\mathsf{pp},\{(\mathcal{S},\mathcal{S}'),(\mathbf{u},\mathbf{u}'),\rho)$.

To begin with, the simulator inputs a random challenge $\hat{\eta}$ to aggregate the structures and instances accordingly to obtain the folded structure $\hat{S}^*$ containing

$$\hat{M_j}^* = \hat{\eta} \cdot M_j + \hat{\eta}^2 \cdot M'_j$$

for all $j \in [t]$, and part of the folded instance $\hat{\mathbf{u}}^*$ containing

$$\begin{aligned} \hat{v_0}^* &\leftarrow \hat{\eta} \cdot v_0 + \hat{\eta}^2 \cdot 1, \\ \hat{\mathsf{io}}^* &\leftarrow \hat{\eta} \cdot \mathsf{io} + \hat{\eta}^2 \cdot \mathsf{io}'. \\ \hat{v_j}^* &\leftarrow \hat{\eta} \cdot v_j + \hat{\eta}^2 \cdot v'_j \ \forall j \in [t], \end{aligned}$$

To simulate the trace $\hat{\mathsf{tr}}$, the simulator samples a random vector in $\mathbb{F}n - l - 1$ as $\hat{\mathbf{w}}^*$, and compute the commitment on the random witness $\boldsymbol{w}$ in the dummy instance as

$$\hat{C}'' = \mathsf{Commit}(\mathsf{pp}, \hat{\mathbf{w}}^*) - \hat{\eta} \cdot C - \hat{\eta}^2 \cdot C'.$$

The commitment $\hat{C}^* = \mathsf{Commit}(\mathsf{pp}, \hat{\mathbf{w}}^*)$ is added to the instance $\hat{\mathbf{u}}^*$.

By sampling another random value as $\hat{v_z}^*$, the simulator computes the value $\hat{\epsilon}''$ for the claim on $\hat{v_z}''$ of the dummy instance as

$$\hat{\epsilon}'' = \hat{v_z}^* - \hat{\eta} \cdot \hat{\epsilon} - \hat{\eta}^2 \cdot \hat{\epsilon}',$$

where $\hat{\epsilon} = v_z, \hat{\epsilon}' = v'_z$. The $\hat{v_z}^*$ is then added to the instance $\hat{\mathbf{u}}^*$.

Denote $\hat{\theta_j} = v_j, \hat{\theta_j}' = v'_j, \hat{\theta_j}'' = \perp$ Now, we have obtained the claims on matrices and instance-witness pairs for three instances as follows

$$\begin{aligned} &\hat{\epsilon} = \tilde{z}(\boldsymbol{r}'_y), \ \hat{\epsilon}' = \tilde{z}'(\boldsymbol{r}'_y), \ \hat{\epsilon}'' = \tilde{z}'(\boldsymbol{r}'_y), \\ &\hat{\theta_j} = \widetilde{M}_j(\boldsymbol{r}'_x, \boldsymbol{r}'_y), \forall j \in [t], \ \hat{\theta_j}' = \widetilde{M}'_j(\boldsymbol{r}'_x, \boldsymbol{r}'_y), \forall j \in [t]. \end{aligned}$$

Note that the matrices for making instance can be set equal to either $\{M_j\}_{j\in[t]}$ or $\{M'_j\}_{j\in[t]}$. The above values are indistinguishable from those in tr.

According to the conclusion given by Chiesa et al. in [35], the Sim can invoke another efficient simulator $\mathsf{Sim}_{\mathsf{sc}}$ to simulate an indistinguishable trace tr_2 for sum-check#2 based on the claims above.

By running the similar process as above, the Sim can simulate another indistinguishable trace tr_1 for sum-check#1 based on the claims given in tr_2.

Finally, the Sim outputs a valid trace $\hat{\mathsf{tr}}$ constructed from $\hat{\epsilon}, \hat{\epsilon}', \hat{\epsilon}''$, $\hat{\theta_j}, \hat{\theta_j}', \hat{\theta_j}''$ and $\mathsf{tr}_1, \mathsf{tr}_2$. Obviously, the Sim can be executed in polynomial time. *1. Extraction Algorithm*: we construct an expected-polynomial time extractor Ext that succeeds with probability $\epsilon - \mathrm{negl}(\lambda)$ in obtaining satisfying witnesses for the original instances as follows.

```
Ext(pp, $\rho$):
1 : Obtain the output tuple from $\mathcal{A}$:
        $(\mathcal{S}, \mathcal{S}', \mathbf{u}, \mathbf{u}', \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \rho)$.
2 : Compute $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, \mathcal{S}, \mathcal{S}')$.
3 : Run the folding interaction#1
        $(\mathcal{S}^*_1, \mathbf{u}^*_1, \widetilde{\mathbf{w}}^*_1) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \mathcal{S}, \mathcal{S}', \mathbf{u}, \mathbf{u}', \mathsf{st})$
    once with the final verifier challenge $\eta_1 \leftarrow\$ \mathbb{F}$.
4 : Abort if $(\mathsf{pp}, \mathcal{S}^*_1, \mathbf{u}^*_1, \widetilde{\mathbf{w}}^*_1) \notin \mathcal{R}_{\mathsf{ACCS}}$.
5 : Run the folding interaction#2
        $(\mathcal{S}^*_2, \mathbf{u}^*_2, \widetilde{\mathbf{w}}^*_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \mathcal{S}, \mathcal{S}', \mathbf{u}, \mathbf{u}', \mathsf{st}))$
    with a different verifier's final challenge $\eta_2 \leftarrow\$ \mathbb{F}$ while maintaining
    the same prior randomness.
6 : Keep doing so until $(\mathsf{pp}, \mathcal{S}^*_2, \mathbf{u}^*_2, \widetilde{\mathbf{w}}^*_2) \in \mathcal{R}_{\mathsf{ACCS}}$.
7 : Abort if $\eta_1 = \eta_2$ or $\mathcal{S}^*_1 \neq \mathcal{S}^*_2$.
8 : Interpolating points $(\eta_1, \widetilde{\mathbf{w}}^*_1)$ and $(\eta_2, \widetilde{\mathbf{w}}^*_2)$, retrieve the witness
    polynomials $\widetilde{\mathbf{w}}$ and $\widetilde{\mathbf{w}}'$ such that for $i \in \{1, 2\}$
        $\widetilde{\mathbf{w}} + \eta_i \cdot \widetilde{\mathbf{w}}' = \widetilde{\mathbf{w}}^*_i$
9 : Output $(\widetilde{\mathbf{w}}, \widetilde{\mathbf{w}}')$.
```

□

Appendix C. Security proofs for PCD

We refer to the security proofs of completeness and knowledge soundness to [27].

Lemma 7 (Perfect Completeness)**.** *Construction 2 satisfies perfect completeness.*

Proof. For public parameter pp, consider arbitrary adversarially chosen messages $(\varphi, z, z_{\mathsf{loc}}, \{z_i, \Pi_i\}_{k\in[s]})$ satisfying

$$\begin{aligned} &\varphi \in \mathsf{F}; \varphi(z, z_{\mathsf{loc}}, \{z_k\}_{k\in[s]}) = 1; \\ &\forall k \in [s], z_k = \perp \text{ or } \mathcal{V}(\mathsf{vk}, z_k, \Pi_k) = 1, \end{aligned}$$

such that the perfect completeness precondition is satisfied. We show that given $\Pi \leftarrow \mathcal{P}(\mathsf{pk}, z, z_{\mathsf{loc}}, \{z_k, \Pi_k\}_{k\in[s]})$, the verifier algorithm passes, i.e., $\mathcal{V}(\mathsf{vk}, z, \Pi) = 1$ with probability 1.

Specifically, there are two cases:

- If $z_k = \perp$ for all $k \in [s]$, the prover runs the algorithm honestly, and the compliance $\varphi(z, z_{\mathsf{loc}}, z_1, ..., z_s)$ holds by the preconditions. The circuit $\mathcal{R}_0$ can be constructed accordingly and a satisfied $\mathcal{R}_{\mathsf{CCCS}}$ instance is as

$$(\mathsf{s}_0, \mathbf{u}_0, \mathbf{w}_0) \leftarrow \mathsf{trace}(R_0, (h, (z, z_{\mathsf{loc}}, \{z_k, \mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathsf{vk}'))$$

  Then the prover sets $(\mathbf{U}, \mathbf{W}, \mathsf{pf})$ accordingly to $(\mathbf{u}_0, \mathbf{w}_0, \perp)$ and computes $h = \mathsf{Hash}(\mathsf{vk}'_{mathsffs}, z, \mathbf{U})$. And the circuit $\mathcal{R}_1$ can be constructed accordingly and a satisfied $\mathcal{R}_{\mathsf{CCCS}}$ instance is as

$$(\mathsf{s}, \mathbf{u}, \mathbf{w}) \leftarrow \mathsf{trace}(R_1, (h, (\{\mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathbf{u}_0, \mathsf{vk}', \mathbf{U}, \Pi)).$$

Besides, $\mathbf{u}.\mathbf{io} = \mathsf{H}(\mathsf{vk}', z, \mathbf{U})$. As a result, $\mathcal{V}(\mathsf{vk}, z, \Pi) = 1$ with probability 1.

- If $\exists k \in [s]$ such that $z_k \neq \bot$, by the perfect completeness precondition, $\{\mathbf{U}_k, \mathbf{W}_k\}_{k\in[s]}$ are satisfied $\mathcal{R}_{\mathsf{ACCS}}$ instance-witness pairs, $\{\mathbf{u}_k, \mathbf{w}_k\}_{k\in[s]}$ are satisfied $\mathcal{R}_{\mathsf{CCCS}}$ instance-witness pairs, and $\mathbf{u}_k.\mathbf{io} = \mathsf{H}(\mathsf{vk}', z_k, \mathbf{U}_k)$. The prover runs the algorithm honestly, and the compliance $\varphi(z, z_{\mathsf{loc}}, z_1, ..., z_s)$ holds by the preconditions. The circuit $\mathcal{R}_0$ can be constructed accordingly and a satisfied $\mathcal{R}_{\mathsf{CCCS}}$ instance is as

$$(\mathsf{s}_0, \mathbf{u}_0, \mathbf{w}_0) \leftarrow \mathsf{trace}(R_0, (h, (z, z_{\mathsf{loc}}, \{z_k, \mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathsf{vk}'))$$

Then, the prover runs generic folding scheme for $\{\mathsf{S}_k, \mathbf{U}_k, \mathbf{W}_k\}_{k\in[s]}$, $\{\mathsf{s}_k, \mathbf{u}_k, \mathbf{w}_k\}_{k=0}^{s}$. By the perfect completeness of the generic folding scheme, we have that $(\mathbf{U}, \mathbf{W})$ is a satisfied $\mathcal{R}_{\mathsf{CCCS}}$ instance-witness pair. The circuit $\mathcal{R}_1$ can be constructed accordingly and a satisfied $\mathcal{R}_{\mathsf{CCCS}}$ instance is as

$$(\mathsf{s}, \mathbf{u}, \mathbf{w}) \leftarrow \mathsf{trace}(R_1, (h, (\{z_k, \mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathbf{u}_0, \mathsf{vk}', \mathbf{U}, \Pi)).$$

Besides, $\mathbf{u}.\mathbf{io} = \mathsf{H}(\mathsf{vk}', z, \mathbf{U})$. As a result, $\mathcal{V}(\mathsf{vk}, z, \Pi) = 1$ with probability 1.

In conclusion, we show that Construction 2 has perfect completeness. □

Lemma 8 (Knowledge Soundness). *Construction 2 satisfies knowledge soundness.*

Proof. Given a fixed set Z, $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ and auxiliary input $\mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp})$, the polynomial time adversary $\mathcal{P}^*$ succeeds in producing valid transcript $(\varphi, \circ, \Pi, \mathsf{ao})$ with non-negligible probability ϵ. We aim to show that it is feasible to construct an extractor $\mathsf{Ext}_{\mathcal{P}^*}$ on input $(\mathsf{pp}, \mathsf{ai})$, succeeds in outputting $(\varphi, \mathsf{T}, \mathsf{ao})$ with probability $\epsilon - \mathrm{negl}(\lambda)$, where $\varphi \in \mathsf{F}$, $(\mathsf{pp}, \mathsf{ai}, \varphi, \mathsf{ao}) \in Z$ and T is φ-compliant.

According to [12], it is convenient to assume the transcript T as a d-depth tree, where d is the depth of φ. Among the tree T, each node u with local data $z(u)_{\mathsf{loc}}$ has a unique outgoing edge labelled with $z^{(u)}$ and a proof $\Pi^{(u)}$ for the correctness of $z^{(u)}$. The extractor $\mathsf{Ext}_{\mathcal{P}^*}$ is constructed inductively by constructing a sequence of extractors $\mathsf{Ext}_0, ..., \mathsf{Ext}_d$. For $i \in 0, ..., d$, Ext_i outputs a $(i+1)$-depth tree T_i. Basically, we define $\mathsf{Ext}_0(\mathsf{pp}, \mathsf{ai})$ runs $(\varphi, \circ, \Pi, \mathsf{ao}) \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{ai})$ and outputs $(\varphi, \mathsf{T}_0, \mathsf{ao})$, where T_0 contains only one node labeled with $(\circ, \Pi)$.

Then assume we already have extractor Ext_{i-1}. To construct Ext_i, an adversary $\mathcal{P}^*_{i-1}$ for the zero-knowledge non-interactive generic folding scheme needs to be constructed first.

$\mathcal{P}^*_{i-1}(\mathsf{pp}, \mathsf{ai}, \rho)$:

```
1 : Compute $(\varphi, \mathsf{T}_{i-1}, \mathsf{ao}) \leftarrow \mathsf{Ext}_{i-1}(\mathsf{pp}, \mathsf{ai})$. If $\mathsf{T}_{i-1}$ is not a tree of depth $i$, abort.
2 : For each node $u \in L_{\mathsf{T}_{i-1}}(i)$, denote its label as $(z^{(u)}, \Pi_{(u)})$.
3 : Parse $\Pi^{(u)}$ as $((\mathbf{U}^{(u)}, \mathbf{W}^{(u)}), (\mathbf{u}^{(u)}, \mathbf{w}^{(u)}))$.
4 : Obtain $(\{\mathbf{U}_k^{(u)}, \mathbf{u}_k^{(u)}, z_j^{(u)}\}_{k\in[s]}, \mathbf{u}_0, \mathsf{pf}^{(u)})$ from $\mathbf{w}^{(u)}$.
5 : Let $L_{i-1} := \{u \in L_{\mathsf{T}_{i-1}}(i) \mid \exists k \in [s], z_k^{(u)} \neq \bot\}$.
6 : Output
    $\left(\left\{\{\mathbf{U}_k^{(u)}, \mathbf{u}_k^{(u)}\}_{k\in[s]}, \mathbf{u}_0^{(u)}, \mathbf{U}^{(u)}, \mathbf{W}^{(u)}, \mathsf{pf}^{(u)}\right\}_{u\in L_{i-1}}, (\varphi, \mathsf{T}_{i-1}, \mathsf{ao})\right)$.
```

where $L_{\mathsf{T}_{i-1}}(i)$ denotes the set of nodes of T at depth i. According to the knowledge soundness of the generic folding scheme, we can construct another extractor $\mathsf{Ext}_{\mathcal{P}^*_{i-1}}$. On input $v \in L_{i-1}$, $\mathsf{Ext}_{\mathcal{P}^*_{i-1}}$ outputs $\{\mathbf{W}_k^{(u)}, \mathbf{w}_k^{(u)}\}_{k\in[s]}$ and $\mathbf{w}_0^{(u)}$ with non-negligible probability, where $\{\mathbf{U}_k^{(u)}, \mathbf{W}_k^{(u)}\}_{k\in[s]}$ are satisfied atomic CCS instance-witness pairs and $\{\mathbf{u}_k^{(u)}, \mathbf{w}_k^{(u)}\}_{k=0}^{s}$ are satisfied committed CCS instance-witness pairs.

Based on $\mathcal{P}^*_{i-1}, \mathsf{Ext}_{\mathcal{P}^*_{i-1}}$, we can further construct Ext_i as follows.

$(\varphi, \mathsf{T}_i, \mathsf{ao}) \leftarrow \mathsf{Ext}_i(\mathsf{pp}, \mathsf{ai})$:

```
1 : Run $\leftarrow \mathsf{Ext}_{\mathcal{P}^*_{i-1}}(\mathsf{pp}, \mathsf{ai}, \rho)$ to obtain
    $\left(\left\{\{\mathbf{U}_k^{(u)}, \mathbf{W}_k^{(u)}\}_{k\in[s]}, \{\mathbf{u}_k^{(u)}, \mathbf{w}_k^{(u)}\}_{k=0}^{s}\right\}_{u\in L_{i-1}}, (\varphi, \mathsf{T}_{i-1}, \mathsf{ao})\right)$
    If $\mathsf{T}_{i-1}$ is not a tree of depth $i$, abort.
2 : Retrieve $\{\mathbf{w}^{(u)}\}_{u\in L_{\mathsf{T}_{i-1}}(i)}$ from the internal state of $\mathcal{P}^*_{i-1}$ and obtain $z_{\mathsf{loc}}^{(u)}, \{z_k^{(u)}\}_{k\in[s]}$ from $\mathbf{w}^{(u)}$.
3 : Append $z_{\mathsf{loc}}^{(u)}$ to the label of $u \in L_{\mathsf{T}_{i-1}(i)}$.
4 : For each node $u \in L_{i-1}$, let $L_u := \{k \in [s] \mid z_k^{(u)} \neq \bot\}$.
5 : Construct $\mathsf{T}_i$ of depth $i+1$ from $\mathsf{T}_{i-1}$ by adding, for each node $u \in L_{i-1}$, $(z_k^{(u)}, \Pi_k^{(u)})$ to the label of its child $k \in L_u$, where
        $\Pi_k^{(u)} = \left((\mathbf{U}_k^{(u)}, \mathbf{W}_k^{(u)}), (\mathbf{u}_k^{(u)}, \mathbf{w}_k^{(u)})\right)$.
6 : Output $(\varphi, \mathsf{T}_i, \mathsf{ao})$.
```

We claim that for $i \in \{0, 1, ..., d\}$, the extractor $\mathsf{Ext}_i(\mathsf{pp}, \mathsf{ai})$ outputs $(\varphi, \mathsf{T}_i, \mathsf{ao})$ in expected polynomial time such that with probability $\epsilon - \mathrm{negl}(\lambda)$, the following conditions hold

- $\varphi \in \mathsf{F}$, $(\mathsf{pp}, \mathsf{ai}, \varphi, \circ(\mathsf{T}_i), \mathsf{ao}) \in Z$;
- T_i is φ-compliant up to depth i;
- for all $u \in L_{\mathsf{T}_i}(i+1)$, $\mathcal{V}(\mathsf{vk}, z^{(u)}, \Pi^{(u)}) = 1$.

The correctness of the above claim can be proved by induction.

- (Base case.) Since $\mathsf{Ext}_0(\mathsf{pp}, \mathsf{ai})$ runs $(\varphi, \circ, \Pi, \mathsf{ao}) \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{ai})$, it satisfies the conditions above.
- (Inductive hypothesis.) Assume that the extractor Ext_{i-1} satisfies the abovementioned conditions.

- (Inductive step.) Based on the hypothesis, we show that Ext_i also satisfies the conditions by the following discussion.

The inductive hypothesis ensures that Ext_{i-1} satisfies with probability $\epsilon-\mathrm{negl}(\lambda)$, that $\varphi \in \mathsf{F}$, $(\mathsf{pp}, \mathsf{ai}, \varphi, \circ(\mathsf{T}_{i-1}), \mathsf{ao}) \in Z$, T_{i-1} is φ-compliant up to the depth $i-1$, and for all $u \in L_{\mathsf{T}_{i-1}}(i)$, $\mathcal{V}(\mathsf{vk}, z^{(u)}, \Pi^{(u)}) = 1$. By the correctness of algorithm $\mathcal{V}$, we have

(1) $\{(\mathbf{U}^{(u)}, \mathbf{W}^{(u)}), (\mathbf{u}^{(u)}, \mathbf{w}^{(u)})\}_{u \in L_{\mathsf{T}_{i-1}(i)}}$ are satisfied instance-witness pairs.

Since T_{i-1} is φ-compliant, by the construction of $\mathcal{R}_0, \mathcal{R}_1$ and hash function Hash, we have

(2) for $u \in L_{\mathsf{T}_{i-1}}(i), \varphi(z^{(u)}, z^{(u)}_{\mathsf{loc}}, z^{(u)}_1, ..., z^{(u)}_s)$ accepts;

(3) for $u \in L_{i-1}$, $\mathbf{U}^{(u)} = \mathsf{zk\text{-}NIFS}.\mathcal{V}'(\mathsf{vk}', \{\mathbf{U}^{(u)}_k\}_{k\in[s]}, \{\mathbf{u}^{(u)}_k\}^s_{k=0}, \mathsf{pf}^{(u)})$;

(4) for $u \in L_{i-1}$, $\mathbf{u}^{(u)}_k.\mathbf{io} = \mathsf{Hash}(\mathsf{vk}', z^{(u)}_k, \mathbf{U}^{(u)}_k)\ \forall k \in [s]$.

Concretely, (2) implies that T_i is φ-compliant up to depth i and $\varphi \in \mathsf{F}$,
$(\mathsf{pp}, \mathsf{ai}, \varphi, \circ(\mathsf{T}_i), \mathsf{ao}) \in Z$. (1) and (3) imply that there exists efficient construction of $\mathcal{P}^*_{i-1}$ that succeeds in producing folded pairs $\{\mathbf{U}^{(u)}, \mathbf{W}^{(u)}\}_{u \in L_{i-1}}$ with probability $\epsilon - \mathrm{negl}(\lambda)$. Then there exists an efficient extractor $\mathsf{Ext}_{\mathcal{P}^*_{i-1}}$ outputting $\{\{\mathbf{W}^{(u)}_k\}_{k\in[s]}, \{\mathbf{w}^{(u)}_k\}^s_{k=0}\}_{u \in L_{i-1}}$ guaranteed by the knowledge soundness of generic foldings scheme. (1)-(4) imply that $\mathcal{V}(\mathsf{vk}, z^{(u)}, \Pi^{(u)}) = 1$ holds for all $u \in L_{(T)_i}(i+1)$. Therefore, the hypothesis for Ext_i also holds.

In conclusion, we prove that Construction 2 is knowledge-sound. □

Lemma 9 (Zero Knowledge)**.** *Construction 2 satisfies zero knowledge.*

Proof. We prove that PCD is zero knowledge by constructing a probabilistic polynomial-time simulator Sim as

```
Sim(1^λ):
1 : Compute $(\mathsf{pp}_{\mathsf{fs}}, \tau_{\mathsf{fs}}) \leftarrow \mathsf{Sim}_{\mathsf{fs}}(1^\lambda)$.
2 : Output $(\mathsf{pp} = \mathsf{pp}_{\mathsf{fs}}, \tau = \tau_{\mathsf{fs}})$.
```

```
Sim(pp, φ, z, τ):
1 : Obtain $\{\mathsf{S}_k, \mathbf{U}_k\}_{k\in[s]}$, $\{\mathsf{s}_k, \mathbf{u}_k\}^s_{k=0}$ from public $\mathcal{R}_1$.
2 : Compute $(\mathsf{S}, \mathbf{U}, \mathbf{W}, \mathsf{pf})$ by running
        $\mathsf{Sim}_{\mathsf{fs}}(\mathsf{pp}_{\mathsf{fs}}, \{\mathsf{S}^{(k)}, \mathbf{U}^{(k)}\}_{k\in[s]}, \{\mathsf{s}^{(k)}, \mathbf{u}^{(k)}\}^s_{k=0}, \tau)$.
3 : Compute $h \leftarrow \mathsf{Hash}(\mathsf{vk}'_{\mathsf{fs}}, z, \mathbf{U})$.
4 : Output $(\mathsf{s}, \mathbf{u}, \mathbf{w}) \leftarrow \mathsf{trace}(\mathcal{R}_1, (h, (\{\mathbf{U}_k, \mathbf{u}_k\}_{k\in[s]}, \mathbf{u}_0, \mathsf{vk}'_{\mathsf{fs}}, \mathbf{U}, \mathsf{pf})))$.
```

□