SnarkFold: Efficient SNARK Proof Aggregation from Split Incrementally Verifiable Computation

Xun Liu, Shang Gao*, Tianyu Zheng, and Bin Xiao

Department of Computing, The Hong Kong Polytechnic University
compxun.liu@connect.polyu.hk
shanggao@polyu.edu.hk
tian-yu.zheng@connect.polyu.hk
csbxiao@polyu.edu.hk

Abstract. The succinct non-interactive argument of knowledge (SNARK) technique is widely used in blockchain systems to replace the costly on-chain computation with the verification of a succinct proof. However, when dealing with multiple proofs, most existing applications require each proof to be independently verified, resulting in a heavy load on nodes and high transaction fees for users. To improve the efficiency of verifying multiple proofs, we introduce SnarkFold, a universal SNARK-proof aggregation scheme based on incrementally verifiable computation (IVC). Unlike previous proof aggregation approaches based on inner product arguments, which have a logarithmic proof size and verification cost, SnarkFold achieves constant verification time and proof size. One core technical advance in SnarkFold, of independent interest, is the "split IVC": rather than using one running instance to fold/accumulate the computation, we employ two (or more) running instances of different types in the recursive circuit to avoid transferring into the same structure. This distinguishing feature is particularly well-suited for proof aggregation scenarios, as constructing arithmetic circuits for pairings can be expensive. We further demonstrate how to fold Groth16 proofs with our SnarkFold. With some further optimizations, SnarkFold achieves the highest efficiency among all approaches.

Keywords: Succinct Non-interactive Argument of Knowledge, Incrementally Verifiable Computation, Proof Aggregation.

1 Introduction

The succinct non-interactive arguments of knowledge (SNARK) is a crucial cryptographic technique that allows a prover to convince a verifier of the correctness of a specific statement in a non-interactive way. This is achieved by the prover constructing a succinct proof that the verifier can efficiently verify. The zero-knowledge version of SNARK, known as zk-SNARK, further guarantees that the proofs reveal no additional information beyond the validity of the prover's statement.

* Shang Gao is the corresponding author of this paper.

In recent years, (zk-)SNARKs have gained great attention in real-world applications, leading to the development of many new constructions and implementations [10] [8] [15] [25] [4] [1] [19]. For example, in zk-Rollups (zero-knowledge rollups), a layer-2 network must provide a SNARK proof to the underlying layer-1 blockchain to show the validity of off-chain transactions [22]. The more general version, zk-EVM (zero-knowledge Ethereum virtual machine), further employs SNARK to demonstrate the correct execution of smart contracts [29]. Additionally, zk-SNARK has been extensively used in private scenarios such as anonymous cryptocurrencies [18] [11]. They can enhance blockchain privacy by allowing users to prove the validity of a transaction without disclosing sensitive information such as the transaction amount, account address, and account balance.

Challenges. Although (zk-)SNARKs are promising for enhancing scalability and privacy, SNARK-based applications pose significant challenges for both the prover and the verifier, as generating and verifying individual proofs can be time-consuming, thereby reducing the system's throughput. This issue is more pronounced in blockchain systems like Ethereum [24], where users (provers) are required to pay transaction fees to the blockchain miners (verifiers) based on the computational and storage resource usage of operations (the cost of proof verification). For instance, in December 2023, a deposit operation in TornadoCash (a smart contract for anonymous transactions on Ethereum) required 80 USD worth of transaction fee, calculated based on the Ethereum exchange rate.

A SNARK-proof aggregation scheme (hereafter referred to as proof aggregation) can be utilized to mitigate these challenges. It compacts multiple proofs $\pi_1, ..., \pi_n$ into a single aggregated proof π, and generates an aggregation proof π_{AGG} to validate the aggregation process. The validity of π and π_{AGG} implies the validity of all individual proofs. This aggregation scheme requires the verification of π and π_{AGG} to be much simpler than checking $\pi_1, ..., \pi_n$ individually, thus reducing both verification time and user costs. Proof aggregation schemes have been applied in many applications, such as SNARKBlock [17] and SnarkPack [9].

One approach to achieve proof aggregation is through a generalized inner product argument (GIPA) [7] [9], which transforms the verification of n SNARK proofs into an inner product form and further employs Bulletproofs-like compression [4] and a KZG commitment [12] to reduce the proof size and verification cost to $O(\log n)$. However, due to the costly operations in bilinear groups (e.g., for pairing-based SNARKs such as Groth16 [10]), the verification overhead of the aggregated proof remains substantial for real-world applications such as TornadoCash.

Motivations. One major objective of this paper is to reduce the size and the verification cost of an aggregation proof. To achieve this, we resort to another cryptographic primitive known as incrementally verifiable computation (IVC) [23], which provides an efficient framework to generate proof for a "long-repeated" computation. Notably, the size of an IVC proof is independent of the number of iterations, making the framework ideal for proof aggregation scenarios.

Originally, IVC is constructed using recursive SNARK [23], which requires the IVC prover (recursive circuit) to show the correctness of the previous step in the current step (i.e., the current proof needs to complete a full verification relation of the previous proof). Recently, state-of-the-art IVC has been developed from the folding/accumulation[1] schemes [14] [6] [5] [28] [13], which defers the costly verification to the final step. The recursive circuit only folds instances and generates a non-interactive argument (NARK) instance to indicate the folding is done correctly, which will be folded in the subsequent iteration.

In a folding scheme, the correctness of the folded instance implies the correctness of all input instances. This motivates us to incorporate folding in proof aggregation, regarding folding as the aggregation function that takes each π_i as an input. In each iteration, the recursive circuit folds (aggregates) proofs into one, and the final folded proof is regarded as the π_{AGG}. When further constructed into IVC, we can ensure a constant size and verification time for π_{AGG}.

However, previous works have limitations when implementing proof aggregation directly. The new proof/instance must have the same form as the instance generated by the recursive circuit, enabling them to be folded into one. For instance, Nova [14] necessitates that all instances be relaxed R1CS (rank-1 constraint system) instances, while BCMS20 [6] requires them to be in bulletproofs/KZG forms. Consequently, in proof aggregation, these methods require new proofs to be transformed into relaxed R1CS instances, or need to use a bulletproofs/Plonk proof (instead of a simple NARK) to demonstrate that the folding has been correctly executed. These transformations introduce a non-negligible overhead into the system.

Contributions. We introduce SnarkFold, a new proof aggregation scheme that improves the state-of-the-art by providing constant size aggregated proofs and constant time verification. The core of SnarkFold is a new folding-based IVC, called *split IVC*, which employs two (or more) instances of different types in the recursive circuit to avoid costly transformations.

We summarize the contributions of this paper as follows.

- We introduce a new structure for constructing IVC from folding schemes, named split IVC. The recursive circuit uses two (or more) running instances to fold instances of different types, eliminating the costly transformations.
- We adopt our split IVC in proof aggregation and further propose SnarkFold, a new system that achieves constant proof size and verification cost when folding multiple (zk-)SNARK proofs.
- To demonstrate the application of SnarkFold, we provide a detailed construction for Groth16 proofs aggregation. By introducing a new "augmented relaxed Groth16 proof relation", we further reduce the prover's cost, making our scheme the most efficient among state-of-the-art approaches.

The structure of this paper is as follows. We first introduce the challenge of verifying multiple SNARK proofs in blockchain systems and emphasize the

[1] Precisely, folding and accumulation (*not* to be the cryptographic accumulator) are techniques with slight differences. In this paper, we use the term "folding" to represent both of techniques.

motivation for an IVC-based aggregation scheme to enhance efficiency (Section 1). We highlight the limitations of existing approaches in Section 2. To solve these, we present SnarkFold, a novel proof aggregation scheme that advances the state-of-the-art by enabling constant-size aggregated proof and constant-time verification. The core building block of SnarkFold is a generic split IVC (Section 4), an IVC framework that is more suitable for proof aggregation purposes. Finally, we introduce the real-world applications of SnarkFold and instantiate it to Groth16 (Section 5).

2 Related Work

We review existing work and techniques for proof aggregation.

(Non-)Membership proof aggregation. Some work focuses on designing efficient proof aggregations with cryptography accumulators for specific proofs, such as (non-)membership proofs. The RSA accumulator is a widely used implementation of this concept. Boneh et al. [3] provide both membership and non-membership proofs in an RSA accumulator. However, this approach is limited to aggregating elements within a prime domain and cannot be directly applied to arbitrary elements. To overcome this limitation, Srinivasan et al. [20] propose a zero-knowledge (non-)membership proof aggregation in the bilinear pairing group settings. Although this approach offers constant proof size and verification time, the bilinear pairing group elements in the trusted setup are significantly large (18 MB for 2^{17} elements). Furthermore, they are not suitable for aggregating general-purpose SNARK proofs such as Groth16 and Plonk.

Generalized inner pairing product argument. Bünz et al. [7] extend the inner-product argument in bulletproofs to pairing-based languages, enabling the aggregation of Groth16 proofs. This approach converts the verification of an aggregated proof into the verification of a generalized inner product argument, which can utilize bulletproofs folding to reduce the proof size to a logarithmic scale. Furthermore, they regard the compressed argument as a polynomial and employ the KZG commitment to reduce the verification cost further. SnarkPack, an optimization of Bünz et al.'s scheme, is designed to aggregate Groth16 proofs and reuses the public parameters from the trusted setup. Both Bünz et al.'s scheme and SnarkPack achieve $O(\log n)$ proof size and $O(\log n)$ verification time.

Incrementally Verifiable Computation. IVC enables a prover to prove the correctness of a "long-repeated" computation, such as the repetition of function F with m as the input, $F(F(...F(m)...))$, with a succinct proof. Traditionally, IVC is constructed using a recursive composition of SNARKs, which requires the recursive circuit to implement a full verification logic for the proof generated in the previous step [2] [23]. Recent research presents a new IVC construction based on the folding technique [6] [5] [14], which allows multiple instances to be folded into a single one. The validity of the folded instance implies the correctness of all instances. This eliminates the need for costly verification from the recursive circuit: the recursive circuit simply folds multiple instances into a folded "running instance" and outputs an "accumulated instance" to demonstrate that F

and folding are executed correctly. By verifying the correctness of the running instance and the accumulated instance in the final step, the verifier can ensure the accuracy of all iterations.

3 Preliminaries

3.1 Notion

This work considers the security parameter as λ. $\mathsf{negl}(\lambda)$ denotes a negligible function in λ. Let $\mathbb{Z}_p$ denote the prime field for a large prime p. We use $r \leftarrow\!\!\$\, \mathbb{Z}_p$ to denote sampling r from $\mathbb{Z}_p$ uniformly at random and $x \leftarrow a$ to denote variable assignment of a to x. For simplicity, $(a_1, ..., a_n)$ is denoted as $(a_i)_{i=1}^n$.
Relations. For a nondeterministic polynomial (NP) relation $\mathcal{R}$, we define it over *public parameters* pp (e.g., the groups and fields), *structure* s (e.g., R1CS coefficient matrices), *instance* u (i.e., the public inputs), and *witness* w (i.e., the secret) tuples, $(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}$.
Bilinear groups. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e)$ be a type III bilinear group of prime order p where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is the bilinear map. Let $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ be the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We define group elements $[x]_1$ as $g^x \in \mathbb{G}_1$, $[x]_2$ as $h^x \in \mathbb{G}_2$, and $[x]_T$ as $e(g, h)^x \in \mathbb{G}_T$.

3.2 (zk-)SNARK and Proof Aggregation

Let $\mathcal{R}$ be a NP relation. A SNARK is described by four algorithms, ($\mathsf{SNARK.G}$ for public parameter generator, $\mathsf{SNARK.K}$ for key generator, $\mathsf{SNARK.P}$ for prover, $\mathsf{SNARK.V}$ for verifier), that work in a non-interactive way:

- $\mathsf{pp} \leftarrow \mathsf{SNARK.G}(1^\lambda)$: On input security parameter λ, sample public parameters pp.
- $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{SNARK.K}(\mathsf{pp}, \mathsf{s})$: Taking the public parameters pp and a structure s representing common structure among instances, generate the prover's proving key pk and the verifier's verification key vk.
- $\pi \leftarrow \mathsf{SNARK.P}(\mathsf{pk}, u, w)$: Given a instance-witness pair (u, w), output a succinct proof π with pk proving that $(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R}$.
- $0/1 \leftarrow \mathsf{SNARK.V}(\mathsf{vk}, u, \pi)$: Given an isntance u and the corresponding proof π, output either 1 by accepting the proof or 0 by rejecting it.

A SNARK satisfies **completeness**, **knowledge soundness**, and **succinctness** properties, which are formally defined in the Appendix A.1. zk-SNARK is a variant of SNARK with **zero-knowledge** property: the proof π reveals nothing about w.

Given a SNARK system and n proofs $\pi_1, ..., \pi_n$. We call AGG a proof aggregation scheme with three functions ($\mathsf{AGG.G}$, $\mathsf{AGG.K}$, $\mathsf{AGG.P}$, $\mathsf{AGG.V}$):

- $\mathsf{pp} \leftarrow \mathsf{AGG.G}(1^\lambda)$: On input of the security parameter λ, sample public parameters pp.

– $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AGG.K}(\mathsf{pp}, \mathsf{s}')$: Taking the public parameters pp and a structure s', generate the prover's proving key pk and the verifier's verification key vk for the aggregation scheme.
– $((u, \pi), \pi_{\mathsf{AGG}}) \leftarrow \mathsf{AGG.P}(\mathsf{pk}, (u_i, \pi_i)_{i=1}^n)$: Given n instance-proof pairs, output an aggregated proof-instance pair (u, π) and an aggregation proof π_{AGG} that shows the correctness of the aggregation process.
– $0/1 \leftarrow \mathsf{AGG.V}(\mathsf{vk}, n, u, \pi_{\mathsf{AGG}})$: Given an aggregated instance u and the corresponding aggregation proof u_{AGG}, output either 1 by accepting the aggregation or 0 by rejecting.

Note that the aggregation proof π_{AGG} only ensures the correct execution of the aggregation process. In real-world applications, the verifier needs to further verify (u, π) to ascertain the validity of all (u_i, π_i) pairs. A proof aggregation satisfies **perfect completeness** and **knowledge soundness**, which are formally defined in Appendix A.2.

3.3 IVC and Folding Scheme

IVC allows efficient verification on repeated computation of F, i.e., $F(z_{i-1}, \omega_{i-1}) = z_i$ at step i, where ω_{i-1} is an auxiliary input and z_{i-1} is the output in step $i-1$. IVC is constructed by a tuple of PPT algorithms ($\mathsf{IVC.G}$, $\mathsf{IVC.K}$, $\mathsf{IVC.P}$, $\mathsf{IVC.V}$) with the following interface:

– $\mathsf{pp} \leftarrow \mathsf{IVC.G}(1^\lambda)$: Given a security parameter λ, sample public parameters pp.
– $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F)$: Given the public parameter pp and the function F, generate a proving key pk and a verification key vk.
– $\pi_i \leftarrow \mathsf{IVC.P}(\mathsf{pk}, i, z_0, z_i, z_{i-1}, \omega_{i-1}, \pi_{i-1})$: Taking a counter of current step i, an initial input z_0, two claimed outputs of the current and previous steps z_{i-1} and z_i, an auxiliary input ω_{i-1}, and an proof π_{i-1} attesting to z_{i-1} is correct, output a proof π_i for $z_i = F(z_{i-1}, \omega_{i-1})$ with pk.
– $0/1 \leftarrow \mathsf{IVC.V}(\mathsf{vk}, i, z_0, z_i, \pi_i)$: On the input of a counter of current step i, an initial input z_0, a claimed output of the i-th iteration z_i, and an proof π_i attesting to z_i, output 1 if π_i is a valid proof and 0 otherwise with vk.

An IVC scheme satisfies **perfect completeness**, **knowledge soundness**, and **succinctness**, which are formally defined in the Appendix A.3.

One approach to achieve IVC is from folding schemes, which allow the prover and verifier to transform the task of verifying two (or more) instances of relation $\mathcal{R}$ into the task of verifying a single instance in $\mathcal{R}$. A folding scheme consists of four algorithms ($\mathsf{Fold.G}$, $\mathsf{Fold.K}$, $\mathsf{Fold.P}$, $\mathsf{Fold.V}$) with the following interface:

– $\mathsf{pp} \leftarrow \mathsf{Fold.G}(1^\lambda)$: Given a security parameter λ, sample public parameters pp.
– $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Fold.K}(\mathsf{pp}, \mathsf{s})$: Given the public input pp and a common structure s between instances to be folded, output a prover key pk and a verifier key vk.
– $(u, w) \leftarrow \mathsf{Fold.P}(\mathsf{pk}, (u_1, w_1), (u_2, w_2))$: Given two instance-witness pairs (u_1, w_1) and (u_2, w_2), generate a new folded instance-witness pair (u, w) of the same size.

– $u \leftarrow \mathsf{Fold.V}(\mathsf{vk}, u_1, u_2)$: Given the instance u_1 and u_2, outputs a new folded instance u.

The above algorithms can also be generalized to multiple folding (e.g., $(u, w) \leftarrow \mathsf{Fold.P}(\mathsf{pk}, (u_i, w_i)_{i=1}^n)$). A folding scheme satisfies **perfect completeness** and **knowledge soundness**, which are formally defined in Appendix A.4.

3.4 Groth16 Backgroud

Groth16 [10] is a pairing-based efficient zkSNARK, which has been widely adopted in various cryptographic applications [26] [16]. We briefly present the algorithms of $\mathsf{SNARK}_{\mathsf{Gro16}}$ (the details of each parameter are presented in Appendix B.1).

– $\mathsf{pp} \leftarrow \mathsf{SNARK}_{\mathsf{Gro16}}.\mathsf{G}(1^\lambda)$: Based on λ, sample a type III bilinear group ($\mathbb{G}_1$, $\mathbb{G}_2, \mathbb{G}_T, p, e$) with $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$ as generators. Output $\mathsf{pp} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g, h)$.
– $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{SNARK}_{\mathsf{Gro16}}.\mathsf{K}(\mathsf{pp}, \mathsf{s})$: Generate a common reference string crs which contains the necessary group elements for the prover and verifier. Output $\mathsf{pk} = \mathsf{crs}$ and $\mathsf{vk} = \mathsf{crs}$.
– $\pi \leftarrow \mathsf{SNARK}_{\mathsf{Gro16}}.\mathsf{P}(\mathsf{pk}, u, w)$: Given a R1CS instance-witness pair (u, w), output a Groth16 proof $\pi = (A, B, C)$ consists of three group elements $A, C \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.
– $0/1 \leftarrow \mathsf{SNARK}_{\mathsf{Gro16}}.\mathsf{V}(\mathsf{vk}, u, \pi)$: Compute $H \in \mathbb{G}_1$ based on vk (i.e., crs) and check $e(A, B) = e(C, [\delta]_2) \cdot e(H, [\gamma]_2) \cdot D$.

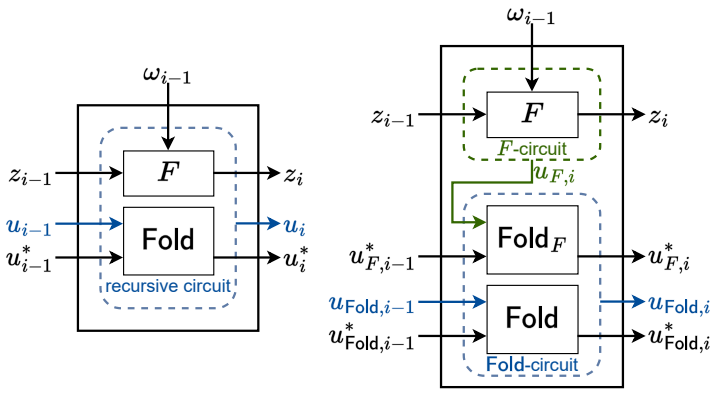
4 Split IVC

We introduce a novel IVC construction derived from folding schemes named split IVC. It utilizes multiple running instances during its recursive computation, thereby avoiding costly transformations between relations. First, we provide a brief overview of our techniques. Then, we present the details of circuits in the split IVC. Lastly, we formally describe the construction of split IVC.

4.1 Technique Overview

We begin by revisiting the existing IVC construction from folding schemes, as illustrated in Figure 1a. At step i, the prover computes an incremental step of F, $z_i = F(z_{i-1}, \omega_{i-1})$, and folds the running instance u^*_{i-1} and the accumulated instance u_{i-1} from the previous step into a new running instance u^*_i, $u^*_i = \mathsf{Fold}(u_{i-1}, u^*_{i-1})$. Additionally, it uses a recursive circuit to generate a new claim (as a new accumulated instance u_i) to show the above computation has been correctly executed.[2]

[2] Precisely, the prover needs to run $\mathsf{Fold.P}$ to fold both instances and witnesses. The recursive circuit shows $\mathsf{Fold.V}$ is correctly executed, and the corresponding claim should also include a witness. The folded witness and the witness of the claim will serve as an IVC proof. We omit witness in the description here for simplicity.



(a) Traditional IVC. (b) Split IVC.

Fig. 1: The IVC structure from a folding scheme in each iteration. The witness part is omitted for simplicity.

A significant issue in existing IVC constructions is the costly conversion of F into the same form as u_i, e.g., in Nova [14], all computations are required to be rewritten into a relaxed R1CS relation. However, for some operations such as elliptic curve pairing, these transformations can be highly costly. As pointed out in [21], a single pairing operation requires up to 25 million gates, which takes a compilation time of 4.2 hours.

To address this problem, we introduce a novel construction called *split IVC*: instead of using a single running instance in the recursive computation, we employ two (or more) instances of different types, as depicted in Figure 1b. Specifically, we use an "F-friendly" instance $u_{F,i}$ to represent the correct computation of F (F-circuit[3]) and fold it with the running F-instance $u^*_{F,i-1}$ from the previous step to a new running F-instance $u^*_{F,i}$. Additionally, we also maintain a "Fold-friendly" running instance $u^*_{\mathsf{Fold},i}$, which is derived by folding the running Fold-instance $u^*_{\mathsf{Fold},i-1}$ and accumulated Fold-instance $u_{\mathsf{Fold},i-1}$ from the previous step. The "Fold-circuit" will show the correct execution of the two foldings (Fold_F and Fold), outputting a claim as a new accumulated Fold-instance $u_{\mathsf{Fold},i}$.

A key advantage of our split IVC is its ability to bypass the costly transformation. For example, when F is a pairing relation of (A_i, B_i) such that $e(A_i, h) = e(B_i, C)$ for some $C \in \mathbb{G}_2$, we can use an *algebraic instance* $(A_i, B_i) \in (\mathbb{G}_1, \mathbb{G}_1)$ to represent F, incurring no additional cost. To fold two F-instances, (A_1, B_1) and (A_2, B_2), Fold_F simply computes $A^* = A_1 \cdot A_2^r$ and $B^* = B_1 \cdot B_2^r$ (r donates a random challenge from $\mathbb{Z}_p$). Consequently, the Fold-circuit only needs to transfer the correct execution of two $\mathbb{G}_1$ operations (and Fold) into a $u_{\mathsf{Fold},i}$ instance (e.g., *relaxed R1CS*), without the need for the costly transformation of pairing.

[3] We slightly abuse the term "circuit" here since some instances (relations) may not be constructed from the circuit.

4.2 Circuit Design

To describe our techniques more precisely, we include the witness part. In our construction of the split IVC, the F-circuit simply transforms the correct execution of F into an F-friendly instance-witness pair $(u_{F,i}, w_{F,i})$. The Fold-circuit takes two F-friendly instances ($u_{F,i}$ and $u^*_{F,i-1}$) and two Fold-friendly instances ($u_{\mathsf{Fold},i-1}$ and $u^*_{\mathsf{Fold},i-1}$), and derives two running instances $u^*_{F,i}$ and $u^*_{\mathsf{Fold},i}$ as follows:

$$\begin{aligned} u^*_{F,i} &\leftarrow \mathsf{Fold}_F.\mathsf{V}(\mathsf{vk}_F, u_{F,i}, u^*_{F,i-1}), \\ u^*_{\mathsf{Fold},i} &\leftarrow \mathsf{Fold}.\mathsf{V}(\mathsf{vk}_{\mathsf{Fold}}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1}). \end{aligned} \tag{1}$$

Additionally, the Fold-circuit will generate an instance-witness pair $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$ to show the correct execution of Equation (1).

The prover will fold both instance-witness pairs locally:

$$\begin{aligned} (u^*_{F,i}, w^*_{F,i}) &\leftarrow \mathsf{Fold}_F.\mathsf{P}\big(\mathsf{pk}_F, (u_{F,i}, w_{F,i}), (u^*_{F,i-1}, w^*_{F,i-1})\big), \\ (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i}) &\leftarrow \mathsf{Fold}.\mathsf{P}\big(\mathsf{pk}_{\mathsf{Fold}}, (u_{\mathsf{Fold},i-1}, w_{\mathsf{Fold},i-1}), \\ &\qquad (u^*_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1})\big). \end{aligned}$$

The above construction has a subtle issue: since $u^*_{F,i}$ and $u^*_{\mathsf{Fold},i}$ are outputs of the Fold-circuit, they must be included in $u_{\mathsf{Fold},i}$ as a part of public input (i.e., $u^*_{F,i}, u^*_{\mathsf{Fold},i} \in u_{\mathsf{Fold},i}$). This implies that the structures of $u^*_{\mathsf{Fold},i}$ and $u_{\mathsf{Fold},i}$ are different and cannot be folded in the next iteration. To address this inconsistency, we adopt the same idea as Nova, modifying the Fold-circuit to output a collision-resistant hash of its inputs and outputs [14], i.e., $u_{\mathsf{Fold},i}.h = \mathrm{Hash}(\mathsf{vk}, i, u^*_{F,i}, u^*_{\mathsf{Fold},i})$. Consequently, the Fold-circuit outputs a statement that there exists $\big((i, u_{F,i}, u^*_{F,i-1}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1}), (u^*_{F,i},\ u^*_{\mathsf{Fold},i})\big)$ such that the following equations hold:

$$\begin{aligned} u^*_{F,i} &= \mathsf{Fold}_F.\mathsf{V}(\mathsf{vk}_F, u_{F,i}, u^*_{F,i-1}), \\ u^*_{\mathsf{Fold},i} &= \mathsf{Fold}.\mathsf{V}(\mathsf{vk}_{\mathsf{Fold}}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1}), \\ u_{\mathsf{Fold},i-1}.h &= \mathrm{Hash}(\mathsf{vk}, i-1, u^*_{F,i-1}, u^*_{\mathsf{Fold},i-1}), \\ u_{\mathsf{Fold},i}.h &= \mathrm{Hash}(\mathsf{vk}, i, u^*_{F,i}, u^*_{\mathsf{Fold},i}). \end{aligned} \tag{2}$$

Accordingly, the Fold-circuit outputs $u_{\mathsf{Fold},i}.h$. The details of Fold-circuit is in Figure 2.

Generalize to multiple instances. The idea of using two running instances can be generalized to multiple instances. Specifically, we can take k_1 instances to encode the correct execution of F and k_2 instances to encode the folding.[4] Accordingly, the Fold_F folds k_1-many $u_{F,i}$ and k_1-many $u^*_{F,i-1}$ into k_1-many $u^*_{F,i}$, and the Fold folds k_2-many $u_{\mathsf{Fold},i-1}$ and k_2-many $u^*_{\mathsf{Fold},i-1}$ into k_2-many $u^*_{\mathsf{Fold},i}$, as depicted in Figure 3. The claim generated by the Fold-circuit is similar

[4] In these scenarios, it may require some "glue" proof to link all instances. But for some (reusable) relations, the glue proof can be avoided with [27].

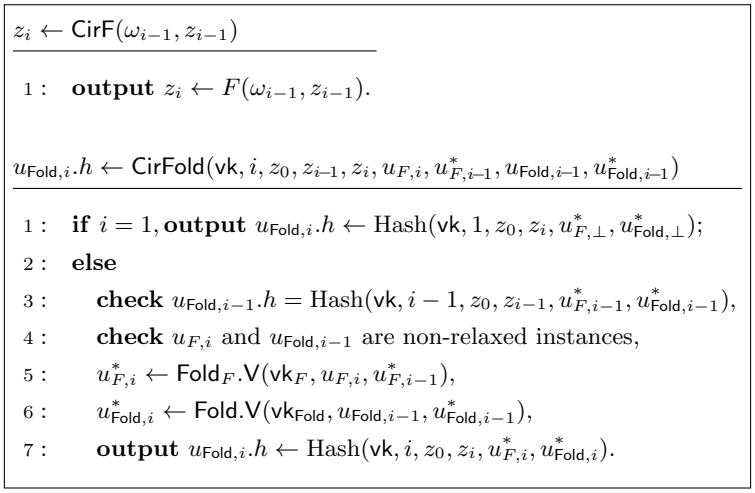


Fig. 2: The logic of CirF and CirFold. (z_0, z_i) is put into CirFold to ensure F is originated from z_0 and avoid the hash relation in CirF.

to Equation (2) by regarding u as a set of multiple instances. The output only contains one element (i.e., the output of the hash function). This could yield an optimization on all commitment-based folding schemes, including Nova, HyperNova, ProtoStar, and KiloNova. The recursive circuit can output two instances: one circuit instance (e.g., relaxed R1CS) to show the correctness of hash functions ($u_i.h$ and the Fiat-Shamir transform) and one algebraic instance to show commitments are folded correctly (group operations).

4.3 Construction of Split IVC

Circuits. Let $(u^*_{F,\perp}, w^*_{F,\perp})$ and $(u^*_{\mathsf{Fold},\perp}, w^*_{\mathsf{Fold},\perp})$ be trivially satisfying instances-witness pairs. We give a formal description of the F-circuit (CirF) Fold-circuit (CirFold) of split IVC.

To ensure the computation originated from z_0, we should use the hash function again in CirF and check $h_i = \mathrm{Hash}(i, z_0, z_i)$. However, here we use a trick to put z_0 and z_i in CirFold instead of CirF, thereby avoiding the hash relation in CirF. We formally described the logic of circuits in Figure 2.

CirF and CirFold can be expressed as a F-friendly instance with structure s_F and a Fold-friendly instance with structure $\mathsf{s}_{\mathsf{Fold}}$, respectively. Let $(u, w) \leftarrow \mathsf{tr}(\mathsf{Circuit}, \mathrm{inputs})$ denotes a satisfying instance-witness pair for the execution of Circuit.

IVC Proofs. In step i, the ICV prover computes $(u^*_{F,i}, w^*_{F,i})$, $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$ and $(u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})$ pairs. Since $u^*_{F,i}$, $u_{\mathsf{Fold},i}$, and $u^*_{\mathsf{Fold},i}$ together attest the correctness of all i steps of F (indirectly attests $u^*_{F,i}$ which implies the correctness of the i-th step of F), the IVC proof is $\pi_i = \big((u^*_{F,i}, w^*_{F,i}), (u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i}), (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})\big)$.

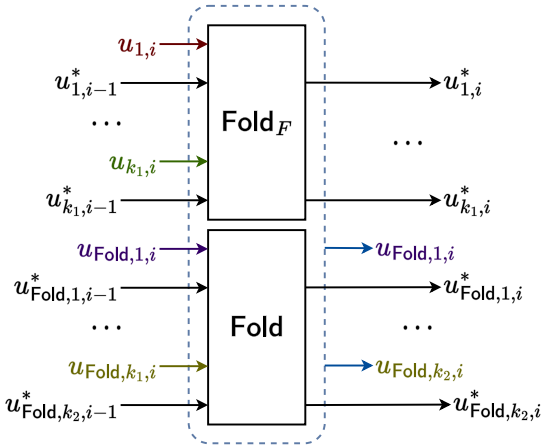


Fig. 3: Fold-circuit implemented with multiple instances.

Construction. We formally describe our split IVC scheme in Figure 4. If the original SNARK requires relaxation in folding, we need to check $u_{\mathsf{Fold},i}$ is a non-relaxed instance in the last step of IVC.V. Otherwise, this final step can be avoided.

Theorem 1. *The construction of split IVC in Figure 4 satisfies perfect completeness, knowledge soundness, and succinctness if the* Fold_F *and* Fold *have perfect completeness and knowledge soundness.*

Proof Sketch. The succinctness trivially holds since $(u^*_{F,i}, w^*_{F,i})$, $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$, and $(u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})$ are three instance-witness pairs. We prove the perfect completeness by induction. Specifically, we assume that the proof π_{i-1}, generated by IVC.P, is valid at step $i-1$. Then, we demonstrate that the proof π_i for step i is also valid. For the knowledge soundness, we use the recursive extraction technique described in [6], which allows us to construct an extractor $\mathcal{E}_i$ that outputs $\big((z_i, ..., z_{n-1}), (\omega_i, ..., \omega_{n-1}), \pi_{i-1}\big)$. To achieve so, we use *inductive hypothesis* to assume that the existence of an extractor $\mathcal{E}_i$. We then use the outputs of $\mathcal{E}_i$ and the extractors of folding schemes to construct $\mathcal{E}_{i-1}$. The formal proof is provided in Appendix C.2.

5 SnarkFold

We present our proof aggregation scheme, SnarkFold, which is built from split IVC. We also construct a Groth16 folding scheme for real-world applications.

5.1 SnarkFold from Split IVC

The core idea of SnarkFold is to view each instance-proof pair (u_i, π_i) as the instance-proof pair $(u_{F,i}, w_{F,i})$ derived from F-circuit in split IVC, i.e., π_i as

$\mathsf{pp} \leftarrow \mathsf{IVC.G}(1^\lambda)$

1 : $\mathsf{pp}_F \leftarrow \mathsf{Fold}_F.\mathsf{G}(1^\lambda), \quad \mathsf{pp}_{\mathsf{Fold}} \leftarrow \mathsf{Fold.G}(1^\lambda),$
2 : **output** $\mathsf{pp} \leftarrow (\mathsf{pp}_F, \mathsf{pp}_{\mathsf{Fold}})$.

$(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F)$

1 : $(\mathsf{pk}_F, \mathsf{vk}_F) \leftarrow \mathsf{Fold}_F.\mathsf{K}(\mathsf{pp}_F, \mathsf{s}_F),$
2 : $(\mathsf{pk}_{\mathsf{Fold}}, \mathsf{vk}_{\mathsf{Fold}}) \leftarrow \mathsf{Fold}_{\mathsf{Fold}}.\mathsf{K}(\mathsf{pp}_{\mathsf{Fold}}, \mathsf{s}_{\mathsf{Fold}}),$
3 : **output** $(\mathsf{pk}, \mathsf{vk}) \leftarrow \big((F, \mathsf{pk}_F, \mathsf{pk}_{\mathsf{Fold}}), (F, \mathsf{vk}_F, \mathsf{vk}_{\mathsf{Fold}})\big)$.

$\pi_i \leftarrow \mathsf{IVC.P}(\mathsf{pk}, i, z_0, z_{i-1}, \omega_{i-1}, \pi_{i-1})$

1 : **parse** π_{i-1} **as** $\big((u^*_{F,i-1}, w^*_{F,i-1}), (u_{\mathsf{Fold},i-1}, w_{\mathsf{Fold},i-1}), (u^*_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1})\big)$,
2 : **if** $i = 1$
3 : $\quad (u^*_{F,i}, w^*_{F,i}) \leftarrow (u^*_{F,\perp}, w^*_{F,\perp}),\ (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i}) \leftarrow (u^*_{\mathsf{Fold},\perp}, w^*_{\mathsf{Fold},\perp});$
4 : **else**
5 : $\quad (u_{F,i}, w_{F,i}) \leftarrow \mathsf{tr}\big(\mathsf{CirF}, (\omega_{i-1}, z_{i-1})\big),$
6 : $\quad (u^*_{F,i}, w^*_{F,i}) \leftarrow \mathsf{Fold}_F.\mathsf{P}\big(\mathsf{pk}_F, (u_{F,i}, w_{F,i}), (u^*_{F,i-1}, w^*_{F,i-1})\big),$
7 : $\quad (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i}) \leftarrow \mathsf{Fold.P}\big(\mathsf{pk}_{\mathsf{Fold}}, (u_{\mathsf{Fold},i-1}, w_{\mathsf{Fold},i-1}), (u^*_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1})\big);$
8 : $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i}) \leftarrow \mathsf{tr}\big(\mathsf{CirFold}, (\mathsf{vk}, i, z_0, z_{i-1}, z_i, u_{F,i}, u^*_{F,i-1}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1})\big),$
9 : **output** $\pi_i \leftarrow \big((u^*_{F,i}, w^*_{F,i}), (u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i}), (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})\big)$.

$0/1 \leftarrow \mathsf{IVC.V}(\mathsf{vk}, i, z_0, z_i, \pi_i)$

1 : **if** $i = 0$, **check** $z_i = z_0$;
2 : **else**
3 : $\quad$ **parse** π_i **as** $\big((u^*_{F,i}, w^*_{F,i}), (u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i}), (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})\big)$,
4 : $\quad$ **check** $u_{\mathsf{Fold},i}.h = \mathrm{Hash}(\mathsf{vk}, i, z_0, z_i, u^*_{F,i}, u^*_{\mathsf{Fold},i})$,
5 : $\quad$ **check** $w^*_{F,i}, w_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i}$ are satisfying witnesses to $u^*_{F,i}, u_{\mathsf{Fold},i}, u^*_{\mathsf{Fold},i}$ respectively,
6 : $\quad$ **check** $u_{\mathsf{Fold},i}$ is a non-relaxed instance.

Fig. 4: The split IVC scheme.

a secret. Using a folding scheme for the SNARK proof, denoted as $\mathsf{Fold}_{\mathsf{SNARK}}$, the (u_i, π_i) pair is folded with the previously aggregated instance-proof pair (u^*_{i-1}, π^*_{i-1}) to generate a new aggregated pair (u^*_i, π^*_i). Furthermore, we employ a Fold-circuit (referred to as $\mathsf{CirFold}'$ in Figure 5) to ensure the instance part is folded correctly, which is a simplified version of $\mathsf{CirFold}$ in split IVC without z_0, z_{i-1}, z_i.

$u_{\mathsf{Fold},i}.h \leftarrow \mathsf{CirFold}'(\mathsf{vk}, i, u_i, u^*_{i-1}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1})$

1 : **if** $i = 1$, **output** $u_{\mathsf{Fold},i}.h \leftarrow \mathrm{Hash}(\mathsf{vk}, 1, u^*_\perp, u^*_{\mathsf{Fold},\perp})$;
2 : **else**
3 : **check** $u_{\mathsf{Fold},i-1}.h = \mathrm{Hash}(\mathsf{vk}, i-1, u^*_{i-1}, u^*_{\mathsf{Fold},i-1})$,
4 : **check** u_i and $u_{\mathsf{Fold},i-1}$ are non-relaxed instances,
5 : $u^*_i \leftarrow \mathsf{Fold}_{\mathsf{SNARK}}.\mathsf{V}(\mathsf{vk}_{\mathsf{SF}}, u_i, u^*_{i-1})$,
6 : $u^*_{\mathsf{Fold},i} \leftarrow \mathsf{Fold}.\mathsf{V}(\mathsf{vk}_{\mathsf{Fold}}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1})$,
7 : **output** $u_{\mathsf{Fold},i}.h \leftarrow \mathrm{Hash}(\mathsf{vk}, i, u^*_i, u^*_{\mathsf{Fold},i})$.

Fig. 5: The logic of $\mathsf{CirFold}'$ in proof aggregation.

In a straightforward approach, the resulting aggregated instance-proof pair is $(u, \pi) = (u^*_n, \pi^*_n)$ and the aggregation proof includes (u^*_n, π^*_n), $(u_{\mathsf{Fold},n}, w_{\mathsf{Fold},n})$, and $(u^*_{\mathsf{Fold},n}, w^*_{\mathsf{Fold},n})$. To further improve efficiency, the prover can fold $(u_{\mathsf{Fold},n}, w_{\mathsf{Fold},n})$ and $(u^*_{\mathsf{Fold},n}, w^*_{\mathsf{Fold},n})$ into (u', w') and employ a SNARK for Fold-circuit (denoted as SNARK', which can be different from the SNARK to be aggregated) to produce a proof π' showing the knowledge of w'. Accordingly, the aggregation proof π_{AGG} becomes $(u^*_n, u_{\mathsf{Fold},n}, u^*_{\mathsf{Fold},n}, \pi')$. Since $u^*_n = u$, this part can be further omitted in π_{AGG}, i.e., $\pi_{\mathsf{AGG}} = (u_{\mathsf{Fold},n}, u^*_{\mathsf{Fold},n}, \pi')$. In verification, u is input into the hash function to check $u_{\mathsf{Fold},n}.h$. The aggregation verifier locally folds $u_{\mathsf{Fold},n}$ and $u^*_{\mathsf{Fold},n}$ to obtain u' and run $\mathsf{SNARK}'.\mathsf{V}$ to check only *one* proof π'.[5]

The above construction folds one proof in each iteration. If $\mathsf{Fold}_{\mathsf{SNARK}}$ can handle multiple pairs, the proof aggregation could be implemented more efficiently by folding multiple proofs in one step. For simplicity, we only consider the case of one proof.

Let $\mathsf{Fold}_{\mathsf{SNARK}}$ be a folding scheme for SNARK proofs with proving key $\mathsf{pk}_{\mathsf{SF}}$ and verification key $\mathsf{vk}_{\mathsf{SF}}$, SNARK' be a SNARK for $\mathsf{CirFold}'$ with structure s', $(u^*_\perp, \pi^*_\perp)$ be a trivial satisfying instance-proof pair, and $(u^*_{\mathsf{Fold},\perp}, w^*_{\mathsf{Fold},\perp})$ be a trivially satisfying instances-witness pair. We formally describe the construction of SnarkFold in Figure 6.

[5] In real-world applications, the verifier needs to further check (u^*_n, π^*_n) to ensure the correctness of all instance-proof pairs. We omit this check here as the aggregation verifier only cares about the validity of the aggregation process.

$\mathsf{pp} \leftarrow \mathsf{AGG.G}(1^\lambda)$

1 : $\mathsf{pp}_{\mathsf{IVC}} \leftarrow \mathsf{IVC.G}(1^\lambda), \quad \mathsf{pp}_{\mathsf{SNARK}'} \leftarrow \mathsf{SNARK}'.\mathsf{G}(1^\lambda),$
2 : **output** $\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{IVC}}, \mathsf{pp}_{\mathsf{SNARK}'})$.

$(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AGG.K}(\mathsf{pp}, \mathsf{s}')$

1 : $(\mathsf{pk}_{\mathsf{IVC}}, \mathsf{vk}_{\mathsf{IVC}}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}_{\mathsf{IVC}}, \emptyset)$,
2 : $(\mathsf{pk}_{\mathsf{SNARK}'}, \mathsf{vk}_{\mathsf{SNARK}'}) \leftarrow \mathsf{SNARK}'.\mathsf{K}(\mathsf{pp}_{\mathsf{SNARK}'}, \mathsf{s}')$,
3 : **output** $(\mathsf{pk}, \mathsf{vk}) \leftarrow \big((\mathsf{s}', \mathsf{pk}_{\mathsf{IVC}}, \mathsf{pk}_{\mathsf{SNARK}'}), (\mathsf{s}', \mathsf{vk}_{\mathsf{IVC}}, \mathsf{vk}_{\mathsf{SNARK}'})\big)$.

$((u, \pi), \pi_{\mathsf{AGG}}) \leftarrow \mathsf{AGG.P}(\mathsf{pk}, (u_i, \pi_i)_{i=1}^n)$

1 : $(u_0^*, \pi_0^*) \leftarrow (u_\perp^*, \pi_\perp^*)$,
2 : $(u_{\mathsf{Fold},0}^*, w_{\mathsf{Fold},0}^*) \leftarrow (u_{\mathsf{Fold},\perp}^*, w_{\mathsf{Fold},\perp}^*)$,
3 : **for** $i = 1$ **to** n
4 : $\quad (u_i^*, \pi_i^*) \leftarrow \mathsf{Fold}_{\mathsf{SNARK}}.\mathsf{P}\big(\mathsf{pk}_{\mathsf{SF}}, (u_i, \pi_i), (u_{i-1}^*, \pi_{i-1}^*)\big)$,
5 : $\quad (u_{\mathsf{Fold},i}^*, w_{\mathsf{Fold},i}^*) \leftarrow \mathsf{Fold.P}\big(\mathsf{pk}_{\mathsf{Fold}}, (u_{\mathsf{Fold},i-1}, w_{\mathsf{Fold},i-1}), (u_{\mathsf{Fold},i-1}^*, w_{\mathsf{Fold},i-1}^*)\big)$;
6 : $\quad (u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i}) \leftarrow \mathsf{tr}\big(\mathsf{CirFold}', (\mathsf{vk}, i, u_i, u_{i-1}^*, u_{\mathsf{Fold},i-1}, u_{\mathsf{Fold},i-1}^*)\big)$,
7 : $(u, \pi) \leftarrow (u_n^*, \pi_n^*)$,
8 : $(u', w') \leftarrow \mathsf{Fold.P}\big(\mathsf{pk}_{\mathsf{Fold}}, (u_{\mathsf{Fold},n}, w_{\mathsf{Fold},n}), (u_{\mathsf{Fold},n}^*, w_{\mathsf{Fold},n}^*)\big)$,
9 : $\pi' \leftarrow \mathsf{SNARK}'.\mathsf{P}(\mathsf{pk}_{\mathsf{SNARK}'}, u', w')$,
10 : $\pi_{\mathsf{AGG}} \leftarrow (u_{\mathsf{Fold},n}, u_{\mathsf{Fold},n}^*, \pi')$,
11 : **output** $\big((u, \pi), \pi_{\mathsf{AGG}}\big)$.

$0/1 \leftarrow \mathsf{AGG.V}(\mathsf{vk}, n, u, \pi_{\mathsf{AGG}})$

1 : **parse** π_{AGG} **as** $(u_{\mathsf{Fold},n}, u_{\mathsf{Fold},n}^*, \pi')$,
2 : **check** $u_{\mathsf{Fold},n}.h = \mathrm{Hash}(\mathsf{vk}_{\mathsf{IVC}}, n, u, u_{\mathsf{Fold},n}^*)$,
3 : $u' \leftarrow \mathsf{Fold.V}\big(\mathsf{vk}_{\mathsf{Fold}}, u_{\mathsf{Fold},n}, u_{\mathsf{Fold},n}^*\big)$,
4 : **check** $\mathsf{SNARK}'.\mathsf{V}(\mathsf{vk}_{\mathsf{SNARK}'}, u', \pi') = 1$,
5 : **check** $u_{\mathsf{Fold},n}$ is a non-relaxed instance.

Fig. 6: The SnarkFold scheme.

Theorem 2. *The construction of SnarkFold proof aggregation in Figure 6 satisfies perfect completeness and knowledge soundness if* IVC *and* SNARK' *has perfect completeness and knowledge soundness.*

Proof Sketch. The perfect completeness is directly inferred from the perfect completeness of IVC and SNARK'.

The knowledge soundness is also implied by the knowledge soundness of IVC and SNARK', composing the two extractors: by running the extractor of SNARK', we can extract a valid w'; by further running the extractor of IVC with different w', we can obtain all proofs $(\pi_i)_{i=1}^n$ such that $(u_i, \pi_i)_{i=1}^n$ are valid instance-proof pairs.

5.2 Folding Scheme for Groth16

To adopt SnarkPack in Groth16 aggregation, we need a folding scheme for Groth16. A straightforward approach is to perform a random linear combination (e.g., $A^* = A_1 \cdot A_2^r$), but this leads to inconsistencies in folded verification due to cross-terms (i.e., $e(A^*, B^*) \neq e(C^*, [\delta]) \cdot e(H^*, [\gamma]_2) \cdot D^*$).

First attempt. We introduce our initial attempt at the folding scheme for Groth16. Specifically, we propose a variant of the Groth16 relation, called "relaxed" Groth16, which introduces some additional elements to ensure that the folded instance-proof pair is satisfiable.

Definition 1 (Relaxed Groth16 Proof Relation). *Given the structure with* $([\delta]_2, [\gamma]_2, D) \in (\mathbb{G}_2, \mathbb{G}_2, \mathbb{G}_T)$*, a relaxed Groth16 proof relation consists of an instance* $(\mu, H, E) \in (\mathbb{Z}_p, \mathbb{G}_1, \mathbb{G}_T)$ *and a proof* $(A, B, C) \in (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_1)$*, such that*

$$e(A, B) \cdot e(C, [\delta]_2)^{-\mu} \cdot e(H, [\gamma]_2)^{-\mu} \cdot D^{-\mu^2} = E.$$

The relaxed relation introduces two additional elements, E and μ. Specifically, E is used to absorb the cross-term generated by folding, and μ is used to absorb additional factors. For traditional (non-relaxed) Groth16 proof relations, $E = [0]_T$ and $\mu = 1$.

We describe the folding scheme for relaxed Groth16 in more details. Given two instance-proof pairs, $(u_1, \pi_1) = \big((\mu_1, H_1, E_1), (A_1, B_1, C_1)\big)$ and $(u_2, \pi_2) = \big((\mu_2, H_2, E_2), (A_2, B_2, C_2)\big)$, the prover $\mathcal{P}$ and verifier $\mathcal{V}$ engage in the following protocol:

1. $\mathcal{P} \to \mathcal{V}$: Compute and send the cross-item T:

$$\begin{aligned} T \leftarrow & e(A_1, B_2) \cdot e(A_2, B_1) \cdot e(C_1^{-\mu_2} C_2^{-\mu_1}, [\delta]_2) \\ & \cdot e(H_1^{-\mu_2} H_2^{-\mu_1}, [\gamma]_2) \cdot D^{-2\mu_1\mu_2}. \end{aligned} \tag{3}$$

2. $\mathcal{V} \to \mathcal{P}$: Sample and send a challenge $r \leftarrow\!\!\$\ \mathbb{Z}_p$.
3. $\mathcal{P}$ and $\mathcal{V}$ Compute the folded relaxed Groth16 instance (μ^*, H^*, E^*):

$$\mu^* \leftarrow \mu_1 + r \cdot \mu_2, \quad H^* \leftarrow H_1 \cdot H_2^r, \quad E^* \leftarrow E_1 \cdot T^r \cdot E_2^{r^2}.$$

4. $\mathcal{P}$ Compute the folded relaxed Groth16 proof (A^*, B^*, C^*):

$$A^* \leftarrow A_1 \cdot A_2^r, \quad B^* \leftarrow B_1 \cdot B_2^r, \quad C^* \leftarrow C_1 \cdot C_2^r.$$

Clearly, $\big((\mu^*, H^*, E^*), (A^*, B^*, C^*)\big)$ is a satisfying pair since

$$\begin{aligned} & e(A^*, B^*) \cdot e(C^*, [\delta]_2)^{-\mu^*} \cdot e(H^*, [\gamma]_2)^{-\mu^*} \cdot D^{-\mu^{*2}} \\ =& e(A_1, B_1) \cdot e(C_1, [\delta]_2)^{-\mu_1} \cdot e(H_1, [\gamma]_2)^{-\mu_1} \cdot D^{-\mu_1^2} \\ & \cdot \big(e(A_1, B_2) \cdot e(A_2, B_1) \cdot e(C_1^{-\mu_2} C_2^{-\mu_1}, [\delta]_2) \\ & \quad \cdot e(H_1^{-\mu_2} H_2^{-\mu_1}, [\gamma]_2) \cdot D^{-2\mu_1\mu_2}\big)^r \\ & \cdot \Big(e(A_2, B_2) \cdot e(C_2, [\delta]_2)^{-\mu_2} \cdot e(H_2, [\gamma]_2)^{-\mu_2} \cdot D^{-\mu_2^2}\Big)^{r^2} \\ =& E_1 \cdot T^r \cdot E_2^{r^2} = E^*. \end{aligned}$$

Augmented relaxed Groth16 proof relation. In the above construction, $\mathcal{P}$ is required to compute *four* pairings for T at step 2 in each iteration. Although the pairing is performed locally (not in the recursive circuit), it introduces a non-negligible cost. To improve efficiency, $\mathcal{P}$ can compute and send $T' = e(A_1, B_2) \cdot e(A_2, B_1)$, $R = C_1^{-\mu_2} C_2^{-\mu_1}$, $S = H_1^{-\mu_2} H_2^{-\mu_1}$, and $\kappa = -2\mu_1\mu_2$ with *two* pairings in step 1. Consequently, the E^* in step 3 becomes $E^* = E_1 \cdot \left(T' \cdot e(R, [\delta]_2) \cdot e(S, [\gamma]_2) \cdot D^{\kappa}\right)^r \cdot E_2^{r^2}$. This may not seem helpful since computing E^* requires two additional pairings, but we observe that R and S (and κ) can further be *folded* in each iteration, and the pairing operation (and $\mathbb{G}_T$ operation) can be deferred to the final step. This indicates we only need $2n+2$ pairings for folding n proofs instead of $4n$ pairings.

To achieve this optimization, we formally introduce the concept of an "augmented relaxed Groth16 proof relation".

Definition 2 (Augmented Relaxed Groth16 Proof Relation). *Given the structure with $([\delta]_2, [\gamma]_2, D) \in (\mathbb{G}_2, \mathbb{G}_2, \mathbb{G}_T)$, an augmented relaxed Groth16 proof relation consists of an instance $(\mu, H, E, R, S, \kappa) \in (\mathbb{Z}_p, \mathbb{G}_1, \mathbb{G}_T, \mathbb{G}_1, \mathbb{G}_1, \mathbb{Z}_p)$ and a proof $(A, B, C) \in (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_1)$, such that*

$$\begin{aligned} & e(A, B) \cdot e(C, [\delta]_2)^{-\mu} \cdot e(H, [\gamma]_2)^{-\mu} \cdot D^{-\mu^2} \\ =& E \cdot e(R, [\delta]_2) \cdot e(S, [\gamma]_2) \cdot D^{\kappa}. \end{aligned} \tag{4}$$

For traditional Groth16 proof relations, we can simply set $\mu = 1$, $E = [0]_T$, $R = [0]_1$, $S = [0]_1$, and $\kappa = 0$. We formally describe the construction of our non-interactive folding scheme (by adopting the Fiat-Shamir transform) for augmented relaxed Groth16 ($\mathsf{Fold}_{\mathsf{Gro16}}$) in Figure 7. $\mathsf{Fold}_{\mathsf{Gro16}}.\mathsf{P}$ and $\mathsf{Fold}_{\mathsf{Gro16}}.\mathsf{V}$ are slightly adjusted to take an auxiliary T to assist the Fiat-Shamir transformation, as with [14].

Theorem 3. *The construction of the folding scheme for Groth16 in Figure 7 satisfies perfect completeness and knowledge soundness under the random oracle model.*

Proof Sketch. The perfect soundness holds trivially. For the knowledge soundness, we prove the *interactive* version of the protocol via the forking lemma in Lemma 1, which implies the knowledge soundness of the non-interactive construction under the Fiat-Shamir heuristic in the random oracle model. We present a formal proof in Appendix C.3.

6 Evaluation

We provide a comparison of the communication and time complexity of Snarkpack and a TIPP-based approach for Groth16 proof aggregation. The Fold-circuit is expressed as a relaxed R1CS relation. Let LC and RC donate the cost of local computations and recursive circuit computations, respectively. P be a pairing operation, H be a hash function, and RO be a random oracle query (can be

$\mathsf{pp} \leftarrow \mathsf{Fold}_{\mathsf{Gro16}}.\mathsf{G}(1^\lambda)$

1 : **output** $\mathsf{pp} \leftarrow \perp$.

$(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Fold}_{\mathsf{Gro16}}.\mathsf{K}(\mathsf{pp}, ([\delta]_2, [\gamma]_2, D))$

1 : **output** $\mathsf{pk} = \mathsf{vk} \leftarrow \big(\mathsf{pp}, ([\delta]_2, [\gamma]_2, D)\big)$.

$((u^*, \pi^*), T) \leftarrow \mathsf{Fold}_{\mathsf{Gro16}}.\mathsf{P}(\mathsf{pk}, (u_1, \pi_1), (u_2, \pi_2))$

1 : **parse** (u_1, π_1) **as** $\big((\mu_1, H_1, E_1, R_1, S_1), (A_1, B_1, C_1)\big)$,
2 : **parse** (u_2, π_2) **as** $\big((\mu_2, H_2, E_2, , R_2, S_2), (A_2, B_2, C_2)\big)$,
3 : $T' \leftarrow e(A_1, B_2) \cdot e(A_2, B_1)$,
4 : $R \leftarrow C_1^{-\mu_2} C_2^{-\mu_1}, \quad S \leftarrow H_1^{-\mu_2} H_2^{-\mu_1}, \quad \kappa \leftarrow -2\mu_1\mu_2$,
5 : $T \leftarrow (T', R, S, \kappa)$,
6 : $r \leftarrow \mathrm{Hash}(\mathsf{pk}, u_1, \pi_1, u_2, \pi_2, T)$,
7 : $\mu^* \leftarrow \mu_1 + r \cdot \mu_2, \quad H^* \leftarrow H_1 \cdot H_2^r, \quad E^* \leftarrow E_1 \cdot (T')^r \cdot E_2^{r^2}$,
8 : $R^* \leftarrow R_1 \cdot R^r \cdot R_2^{r^2}, \quad S^* \leftarrow S_1 \cdot S^r \cdot S_2^{r^2}, \quad \kappa^* \leftarrow \kappa_1 + r \cdot \kappa + r^2 \cdot \kappa_2$,
9 : $A^* \leftarrow A_1 \cdot A_2^r, \quad B^* \leftarrow B_1 \cdot B_2^r, \quad C^* \leftarrow C_1 \cdot C_2^r$,
10 : $(u^*, \pi^*) \leftarrow \big((\mu^*, H^*, E^*, R^*, S^*, \kappa^*), (A^*, B^*, C^*)\big)$,
11 : **return** $\big((u^*, \pi^*), T\big)$.

$u^* \leftarrow \mathsf{Fold}_{\mathsf{Gro16}}.\mathsf{V}(\mathsf{vk}, u_1, u_2, T)$

1 : $r \leftarrow \mathrm{Hash}(\mathsf{pk}, u_1, \pi_1, u_2, \pi_2, T)$,
2 : **parse** T **as** (T', R, S),
3 : $\mu^* \leftarrow \mu_1 + r \cdot \mu_2, \quad H^* \leftarrow H_1 \cdot H_2^r, \quad E^* \leftarrow E_1 \cdot (T')^r \cdot E_2^{r^2}$,
4 : $R^* \leftarrow R_1 \cdot R^r \cdot R_2^{r^2}, \quad S^* \leftarrow S_1 \cdot S^r \cdot S_2^{r^2}, \quad \kappa^* \leftarrow \kappa_1 + r \cdot \kappa + r^2 \cdot \kappa_2$,
5 : **return** $u^* \leftarrow (\mu^*, H^*, E^*, R^*, S^*, \kappa^*)$.

Fig. 7: The Folding Scheme for Groth16.

	Proof Size	**Proving Time**		**Verification Time**	
TIPP-based [7]	6 $\mathbb{G}_1$, 6 $\mathbb{G}_2$, $(12\log n)$ $\mathbb{G}_T$	LC	$(18+\ell)n$ P, $10n$ $\mathbb{G}_1$, $8n$ $\mathbb{G}_2$, $(6n+\ell n-\ell)$ $\mathbb{G}_T$	LC	$16P$, $n\ell$ $\mathbb{G}_1$, $12\log n$ $\mathbb{G}_T$
SnarkFold	6 $\mathbb{G}_1$, 1 $\mathbb{G}_2$, 4 $\mathbb{Z}_p$	LC	$2n$ P, n H, $11n$ $\mathbb{G}_1$, n $\mathbb{G}_2$, n $\mathbb{G}_T$, $(2m-\ell)$ $\mathbb{Z}_p$	LC	3 P, 3 H, 2 $\mathbb{G}_1$, 1 RO
		RC	$5n$ $\mathbb{G}_1$, $2n$ H, $2n$ RO, n $\mathbb{G}_T$	RC	-

Table 1: Efficiency comparisons for proof aggregation schemes. "LC" is for the cost of local computation, and "RC" is for the cost of the recursive circuit. P indicates pairing operation, H indicates hash function, and RO indicates random oracle query. ℓ is the size of Groth16 instance, and m is the total size of Groth16 instance and witness (as in Appendix B.1).

implemented with hash functions in applications, refer to Nova's Lemma 4 for more details). The result is depicted in Table 1.

Proof size. The aggregated proof contains of two relaxed R1CS instance $u_{\mathsf{Fold,n}}$, $u^*_{\mathsf{Fold,n}}$, including 4 elements in $\mathbb{G}_1$ and 4 elements in $\mathbb{Z}_p$. Additionally, it contains a Groth16 proof π', which have 2 element in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$. Therefore, the proof π_{AGG} has a total of 6 elements in $\mathbb{G}_1$, 1 element in $\mathbb{G}_2$, and 4 elements in $\mathbb{Z}_p$.

Prover time. The aggregation prover ($\mathsf{AGG.P}$) performs n iterations of $\mathsf{Fold}_{\mathsf{SNARK}}.P$, $\mathsf{Fold.P}$ and $\mathsf{tr}(\mathsf{CirFold}')$. $\mathsf{Fold}_{\mathsf{SNARK}}.P$ involves $2n$ pairings, n hash computations, $9n$ exponentiations in $\mathbb{G}_1$, n exponentiations in $\mathbb{G}_2$, and n exponentiations in $\mathbb{G}_T$. The recursive circuit $\mathsf{Fold.P}$ requires $2n$ exponentiations in $\mathbb{G}_1$ and $2m-\ell$ scalar multiplications in $\mathbb{Z}_p$. $\mathsf{tr}(\mathsf{CirFold}')$ requires the cost of $5n\mathbb{G}_1+2n\mathsf{H}+2n\mathsf{RO}+n\mathbb{G}_T$ in the recursive circuit.

Verification time. The aggregation verifier ($\mathsf{AGG.V}$) first calls a hash function to verify $u_{\mathsf{Fold}.h}$. Then, it performs $\mathsf{Fold.V}$ with the cost of $2\mathbb{G}_1 + 2\mathsf{H} + 1\mathsf{RO}$). $\mathsf{AGG.V}$ additionally performs a Groth16 verification for proof u', π' with the cost of 3 pairings (D can be precomputed).

7 Conclusion

In conclusion, we propose a novel proof aggregation scheme called SnarkFold that significantly improves the efficiency of verifying multiple SNARK proofs. Our SnarkFold is based on incrementally verifiable computation and employs a new folding-based IVC called split IVC, which avoids costly transformations in recursive circuit. SnarkFold achieves constant verification time and proof size, making it a valuable tool for reducing user transaction fees in blockchain applications. We also provide detailed aggregation designs for Groth16 proofs, a widely used zk-SNARK in real-world applications.

References

1. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. Aurora: Transparent succinct arguments for r1cs. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 103–128. Springer, 2019.
2. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *Advances in Cryptology - CRYPTO*, 2014.
3. Dan Boneh, Benedikt Bünz, and Ben 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*, pages 561–586. Springer, 2019.
4. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018.
5. Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas 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*, pages 681–710. Springer, 2021.
6. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. *Cryptology ePrint Archive*, 2020.
7. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*, pages 65–97. Springer, 2021.
8. Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.
9. Nicolas Gailly, Mary Maller, and Anca Nitulescu. Snarkpack: Practical snark aggregation. In *International Conference on Financial Cryptography and Data Security*, pages 203–229. Springer, 2022.
10. Jens Groth. On the size of pairing-based non-interactive arguments. 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*, pages 305–326. Springer, 2016.
11. Daira Hopwood, Sean Bowe, Taylor Hornby, Nathan Wilcox, et al. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 4(220):32, 2016.
12. Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.
13. Abhiram Kothapalli and Srinath Setty. Supernova: Proving universal machine executions without universal circuits. *Cryptology ePrint Archive*, 2022.

14. Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pages 359–388. Springer, 2022.
15. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.
16. Alexandre Miranda Pinto. An introduction to the use of zk-snarks in blockchains. In *Mathematical Research for Blockchain Economy: 1st International Conference MARBLE 2019, Santorini, Greece*, pages 233–249. Springer, 2020.
17. Michael Rosenberg, Mary Maller, and Ian Miers. Snarkblock: Federated anonymous blocklisting from hidden common input aggregate proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 948–965. IEEE, 2022.
18. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.
19. Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Annual International Cryptology Conference*, pages 704–737. Springer, 2020.
20. Shravan Srinivasan, Ioanna Karantaidou, Foteini Baldimtsi, and Charalampos Papamanthou. Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2719–2733, 2022.
21. Yi Sun. zkpairing, 2023. https://github.com/yi-sun/circom-pairingbenchmarks.
22. Scroll Tech. Scroll tech, 2023. https://github.com/scroll-tech.
23. Paul 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*, pages 1–18. Springer, 2008.
24. Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.
25. Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn 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*, pages 733–764. Springer, 2019.
26. Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3003–3017, 2022.
27. Min Zhang, Yu Chen, Chuanzhou Yao, and Zhichao Wang. Sigma protocols from verifiable secret sharing and their applications. 2023.
28. Tianyu Zheng, Shang Gao, Yu Guo, and Bin Xiao. Kilonova: Non-uniform pcd with zero-knowledge property from generic folding schemes. *Cryptology ePrint Archive*, 2023.
29. Polygon. Polygon zkevm, 2023. https://github.com/0xpolygonhermez.

A Formal Definitions

A.1 (zk-)SNARK

Definition 3 (Perfect Completeness). *A SNARK satisfies perfect completeness if for any PPT adversary $\mathcal{A}$*

$$\Pr\left[\mathsf{SNARK.V}(\mathsf{vk},u,\pi)=1 \;\middle|\; \begin{array}{l}\mathsf{pp}\leftarrow\mathsf{SNARK.G}(1^\lambda),\\ (\mathsf{s},(u,w))\leftarrow\mathcal{A}(\mathsf{pp}),\\ (\mathsf{pp},\mathsf{s},u,w)\in\mathcal{R},\\ (\mathsf{pk},\mathsf{vk})\leftarrow\mathsf{SNARK.K}(\mathsf{pp},\mathsf{s}),\\ \pi\leftarrow\mathsf{SNARK.P}(\mathsf{pk},u,w)\end{array}\right]=1.$$

Definition 4 (Knowledge Soundness). *A SNARK satisfies knowledge soundness if for all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ such that for any randomness ρ*

$$\Pr\left[\begin{array}{l}\mathsf{SNARK.V}(\mathsf{vk},u,\pi)=1,\\ (\mathsf{pp},\mathsf{s},u,w)\notin\mathcal{R}\end{array}\;\middle|\; \begin{array}{l}\mathsf{pp}\leftarrow\mathsf{SNARK.G}(1^\lambda),\\ (\mathsf{s},(u,w))\leftarrow\mathcal{A}(\mathsf{pp};\rho),\\ (\mathsf{pk},\mathsf{vk})\leftarrow\mathsf{SNARK.K}(\mathsf{pp},\mathsf{s}),\\ w\leftarrow\mathcal{E}(\mathsf{pp};\rho)\end{array}\right]=\mathsf{negl}(\lambda).$$

Definition 5 (Succinctness). *A SNARK system is succinct if the size of the SNARK proof π is poly-logarithmic in the size of the witness w.*

Definition 6 (Zero-Knowledge). *A zk-SNARK satisfies zero-knowledge if there exists PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$*

$$\left\{(\mathsf{pp},\mathsf{s},u,\pi)\;\middle|\; \begin{array}{l}\mathsf{pp}\leftarrow\mathsf{SNARK.G}(1^\lambda),\\ (\mathsf{s},(u,w))\leftarrow\mathcal{A}(\mathsf{pp}),\\ (\mathsf{pp},\mathsf{s},u,w)\in\mathcal{R},\\ (\mathsf{pk},\mathsf{vk})\leftarrow\mathsf{SNARK.K}(\mathsf{pp},\mathsf{s}),\\ \pi\leftarrow\mathsf{SNARK.P}(\mathsf{pk},u,w)\end{array}\right\}\approx$$
$$\left\{(\mathsf{pp},\mathsf{s},u,\pi)\;\middle|\; \begin{array}{l}(\mathsf{pp},\tau)\leftarrow\mathcal{S}(1^\lambda),\\ (\mathsf{s},(u,w))\leftarrow\mathcal{A}(\mathsf{pp}),\\ (\mathsf{pp},\mathsf{s},u,w)\in\mathcal{R},\\ (\mathsf{pk},\mathsf{vk})\leftarrow\mathsf{SNARK.K}(\mathsf{pp},\mathsf{s}),\\ \pi\leftarrow\mathcal{S}(\mathsf{pp},u,\tau)\end{array}\right\}.$$

A.2 Proof Aggregation

Definition 7 (Perfect Completeness). *A proof aggregation scheme for* SNARK *(with proving key* $\mathsf{pk}_{\mathsf{SNARK}}$ *and verification key* $\mathsf{pk}_{\mathsf{SNARK}}$*) satisfies perfect completeness if for any PPT adversary $\mathcal{A}$*

$$\Pr\left[\begin{array}{l}\mathsf{AGG.V}(\mathsf{vk},n,u,\pi_{\mathsf{AGG}})=1,\\ \mathsf{SNARK.V}(\mathsf{vk}_{\mathsf{SNARK}},u,\pi)=1\end{array}\;\middle|\; \begin{array}{l}\mathsf{pp}\leftarrow\mathsf{AGG.G}(1^\lambda),\\ (\mathsf{s}',(u_i,\pi_i)_{i=1}^n)\leftarrow\mathcal{A}(\mathsf{pp}),\\ \mathsf{SNARK.V}(\mathsf{vk}_{\mathsf{SNARK}},u_i,\pi_i)=1,\forall i\in\{1,...,n\},\\ (\mathsf{pk},\mathsf{vk})\leftarrow\mathsf{AGG.K}(\mathsf{pp},\mathsf{s}'),\\ ((u,\pi),\pi_{\mathsf{AGG}})\leftarrow\mathsf{AGG.P}\big(\mathsf{pk},(u_i,\pi_i)_{i=1}^n\big)\end{array}\right]=1.$$

Definition 8 (Knowledge Soundness). *A proof aggregation satisfies knowledge soundness if for all PPT adversaries $\mathcal{A}$ and any randomness ρ*

$$\Pr\left[\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathsf{AGG.G}(1^\lambda), \\ \mathsf{AGG.V}(\mathsf{vk}, n, u, \pi_{\mathsf{AGG}}) = 1, & \mathsf{s} \leftarrow \mathcal{A}(\mathsf{pp}; \rho), \\ \mathsf{SNARK.V}(\mathsf{vk}_{\mathsf{SNARK}}, u, \pi) = 1, & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AGG.K}(\mathsf{pp}, \mathsf{s}'), \\ \exists i \ s.t \ \mathsf{SNARK.V}(\mathsf{vk}_{\mathsf{SNARK}}, u_i, \pi_i) = 0 & ((u_i)_{i=1}^n, (u, \pi), \pi_{\mathsf{AGG}}) \leftarrow \\ & \qquad \mathcal{A}(\mathsf{pp}, \mathsf{pk}; \rho), \\ & (\pi_i)_{i=1}^n \leftarrow \mathcal{E}(\mathsf{pp}, u; \rho) \end{array}\right] = \mathsf{negl}(\lambda).$$

A.3 Incrementally Verifiable Computation

Definition 9 (Perfect Completeness). *An IVC satisfies perfect completeness if for any PPT adversary $\mathcal{A}$*

$$\Pr\left[\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathsf{IVC.G}(1^\lambda), \\ \mathsf{IVC.V}(\mathsf{vk}, i, & (F, (i, z_0, z_i, z_{i-1}, \omega_{i-1}, \pi_{i-1})) \leftarrow \mathcal{A}(\mathsf{pp}), \\ z_0, z_i, \pi_i) = 1 & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F), \\ & z_i = F(z_{i-1}, \omega_{i-1}), \\ & \mathsf{IVC.V}(\mathsf{vk}, i-1, z_0, z_{i-1}, \pi_{i-1}) = 1, \\ & \pi_i \leftarrow \mathsf{IVC.P}(\mathsf{pk}, i, z_0, z_i, z_{i-1}, \omega_{i-1}, \pi_{i-1}) \end{array}\right] = 1.$$

Definition 10 (Knowledge Soundness). *An IVC satisfies knowledge soundness if for any constant $n \in \mathbb{N}$ and all expected PPT adversaries $\mathcal{A}$ there exists an expected PPT extractor $\mathcal{E}$ such that for any randomness ρ*

$$\Pr\left[\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathsf{IVC.G}(1^\lambda), \\ z_n \neq z, & (F, (z_0, z, \pi)) \leftarrow \mathcal{A}(\mathsf{pp}; \rho), \\ \mathsf{IVC.V}(\mathsf{vk}, n, & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F), \\ z_0, z, \pi) = 1 & (\omega_0, ..., \omega_{n-1}) \leftarrow \mathcal{E}(\mathsf{pp}, z_0, z; \rho), \\ & z_i = F(z_{i-1}, \omega_{i-1}) \quad \forall i \in \{1, ..., n\} \end{array}\right] = \mathsf{negl}(\lambda).$$

Definition 11 (Succinctness). *An IVC is succinct if the size of the IVC proof π_n does not grow with the number of iterations n.*

A.4 Folding Scheme

Consider a folding scheme between the prover $\mathcal{P}$ and verifier $\mathcal{V}$. Let $(u, w) \leftarrow \langle \mathcal{P}(\mathsf{pk}, w_1, w_2), \mathcal{V}(\mathsf{vk}) \rangle (u_1, u_2)$ denote the verifier's output instance u and the prover's output witness w from the interaction of the folding scheme on instance-witnesses pairs (w_1, u_1) and (w_2, u_2), prover key pk, and verifier key vk. The following definitions can also be generalized to multiple folding.

Definition 12 (Perfect Completeness). *A folding scheme satisfies perfect completeness if for any PPT adversary $\mathcal{A}$*

$$\Pr\left[\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathsf{Fold.G}(1^\lambda), \\ (\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R} & (\mathsf{s}, (u_1, w_1), (u_2, w_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ & (\mathsf{pp}, \mathsf{s}, u_1, w_1), (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}, \\ & (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Fold.K}(\mathsf{pp}, \mathsf{s}), \\ & (u, w) \leftarrow \langle \mathcal{P}(\mathsf{pk}, w_1, w_2), \mathcal{V}(\mathsf{vk}) \rangle (u_1, u_2) \end{array}\right] = 1.$$

Definition 13 (Knowledge Soundness). *A folding scheme satisfies knowledge soundness if for all expected PPT adversaries $\mathcal{A}$, there exists an expected PPT extractor $\mathcal{E}$ such that for any randomness ρ*

$$\Pr\left[(\mathsf{pp},\mathsf{s},u,w)\in\mathcal{R}\;\middle|\;\begin{array}{l}\mathsf{pp}\leftarrow\mathsf{Fold.G}(1^\lambda),\\(\mathsf{s},(u_1,u_2))\leftarrow\mathcal{A}(\mathsf{pp};\rho),\\(\mathsf{pk},\mathsf{vk})\leftarrow\mathsf{Fold.K}(\mathsf{pp},\mathsf{s}),\\(u,w)\leftarrow\langle\mathcal{A}(\mathsf{pk};\rho),\mathcal{V}(\mathsf{vk})\rangle(u_1,u_2)\end{array}\right]-$$
$$\Pr\left[\begin{array}{l}(\mathsf{pp},\mathsf{s},u_1,w_1)\in\mathcal{R},\\(\mathsf{pp},\mathsf{s},u_2,w_2)\in\mathcal{R}\end{array}\;\middle|\;\begin{array}{l}\mathsf{pp}\leftarrow\mathsf{Fold.G}(1^\lambda),\\(\mathsf{s},(u_1,u_2))\leftarrow\mathcal{A}(\mathsf{pp};\rho),\\(w_1,w_2)\leftarrow\mathcal{E}(\mathsf{pp};\rho)\end{array}\right]\leq\mathsf{negl}(\lambda).$$

B Details of zk-SNARKs

B.1 Groth16

Groth16 describes the circuit constraints as a quadratic arithmetic program $\mathcal{R}$, which is defined over the public parameters $\mathsf{pp}=(\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,p,e,g,h)$, structure $\big(\ell,(u_i(X),v_i(X),w_i(X))_{i=1}^m,t(X)\big)$, instance $(a_1,...,a_\ell)\in\mathbb{Z}_p^\ell$, and witness $(a_{\ell+1},...,a_m)\in\mathbb{Z}_p^{m-\ell}$ with $a_0=1$, such that

$$\sum_{i=0}^m a_iu_i(X)\cdot\sum_{i=0}^m a_iv_i(X)=\sum_{i=0}^m a_iw_i(X)+h(X)t(X),$$

for some $(n-2)$-degree quotient polynomial $h(X)$, where n is the degree of $t(X)$.

In $\mathsf{SNARK}_{\mathsf{Gro16}}.\mathsf{K}$, crs is set as follows with randomly sampled $\alpha,\beta,\gamma,\delta,x\leftarrow\mathbb{Z}_p^*$:

$$\Bigg([\alpha]_1,[\beta]_1,[\beta]_2,[\gamma]_2,[\delta]_1,[\delta]_2,\Big([x^i]_1,[x^i]_2\Big)_{i=0}^{n-1},\Big(\Big[\frac{x^it(x)}{\delta}\Big]_1\Big)_{i=0}^{n-1},$$
$$\Big(\Big[\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}\Big]_1\Big)_{i=0}^{\ell},\Big(\Big[\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\Big]_1\Big)_{i=\ell+1}^{m}\Bigg).$$

In $\mathsf{SNARK}_{\mathsf{Gro16}}.\mathsf{P}$, A,B,C is computed as follows with randomly sampled $r,s\leftarrow\mathbb{Z}_p$:

$$A=\left[\alpha+\sum_{i=0}^m a_iu_i(x)+r\delta\right]_1,\quad B=\left[\beta+\sum_{i=0}^m a_iv_i(x)+s\delta\right]_2,$$
$$C=\Bigg[\frac{\sum_{i=\ell+1}^m a_i\big(\beta u_i(x)+\alpha v_i(x)+w_i(x)\big)+h(x)t(x)}{\delta}$$
$$+s\Big(\alpha+\sum_{i=0}^m a_iu_i(x)\Big)+r\Big(\beta+\sum_{i=0}^m a_iv_i(x)\Big)+rs\delta\Bigg]_1.$$

In $\mathsf{SNARK}_{\mathsf{Gro16}}.\mathsf{V}$, H and D is computed as follows:

$$H = \left[\frac{\sum_{i=0}^{\ell} a_i\big(\beta u_i(x) + \alpha v_i(x) + w_i(x)\big)}{\gamma}\right]_1,$$
$$D = e\big([\alpha]_1, [\beta]_2\big).$$

C Formal Proofs

C.1 Lemmas

Lemma 1 (Forking Lemma for Folding Schemes [14]). *Consider a $(2k+1)$-move folding scheme $\mathsf{Fold} = (\mathsf{G}; \mathsf{K}; \mathsf{P}; \mathsf{V})$. Fold satisfies knowledge soundness if there exists a PPT $\mathcal{E}$ such that for all input instance pairs $(u_1; u_2)$, outputs satisfying witnesses $(w_1; w_2)$ with probability $1 - \mathsf{negl}(\lambda)$, given public parameters pp, a structure s, and an $(n_1, ..., n_k)$-tree of accepting transcripts and the corresponding folded instance-witness pairs $(u; w)$. This tree comprises n_1 transcripts (and the corresponding instance-witness pairs) with fresh randomness in the verifier's first message, and for each such transcript, n_2 transcripts (and the corresponding instance-witness pairs) with fresh randomness in the verifier's second message; etc., for a total of $\prod_{i=1}^{k} n_i$ leaves bounded by $\mathsf{poly}(\lambda)$.*

C.2 Proof of Theorem 1

Perfect completeness. Consider (F, i, z_0, z_i) and the corresponding inputs (z_{i-1}, ω_{i-1}) such that $z_i = F(z_{i-1}, \omega_{i-1})$. Let $\mathsf{pp} \leftarrow \mathsf{IVC.G}(1^\lambda)$ and $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F)$. Given a proof π_{i-1} such that $\mathsf{IVC.V}(\mathsf{vk}, i, z_0, z_{i-1}, \pi_{i-1}) = 1$, we prove $\pi_i \leftarrow$
$\mathsf{IVC.P}(\mathsf{pk}, i, z_0, z_i, z_{i-1}, \omega_{i-1}, \pi_{i-1})$ such that $\mathsf{IVC.V}(\mathsf{vk}, i, z_0, z_i, \pi_i) = 1$ by induction on i.

Base Case ($i = 1$): By the base case of $\mathsf{IVC.P}$ ($i = 1$), we have $\pi_1 = \big((u^*_{F,\perp}, w^*_{F,\perp}), (u_{\mathsf{Fold},1}, w_{\mathsf{Fold},1}), (u^*_{\mathsf{Fold},\perp}, w^*_{\mathsf{Fold},\perp})\big)$. By definition, $(u^*_{F,\perp}, w^*_{F,\perp})$ is a trivial satisfying instance-witness pair for F and $(u^*_{\mathsf{Fold},\perp}, w^*_{\mathsf{Fold},\perp})$ is a trivial satisfying instance-witness pair for CircF. Moreover, by construction, $(u_{\mathsf{Fold},1}, w_{\mathsf{Fold},1})$ must also be satisfying and $u_{\mathsf{Fold},1}.h = \mathrm{Hash}(\mathsf{vk}, 1, z_0, z_1, u^*_{F,\perp}, u^*_{\mathsf{Fold},\perp})$. Additionally, $u_{\mathsf{Fold},1}$ is a non-relaxed instance as it is generated from the CircF directly. Therefore, we have $\mathsf{IVC.V}(\mathsf{vk}, 1, z_0, z_1, \pi_1) = 1$.

Inductive Step ($i = 1$ to n): Assume the proof in the $(i-1)$-th step $\pi_{i-1} = \big((u^*_{F,i-1}, w^*_{F,i-1}), (u_{\mathsf{Fold},i-1}, w_{\mathsf{Fold},i-1}), (u^*_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1})\big)$ is a satisfying proof such that $\mathsf{IVC.V}(\mathsf{vk}, i-1, z_0, z_{i-1}, \pi_{i-1}) = 1$. Given $\pi_i = \big((u^*_{F,i}, w^*_{F,i}),$
$(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i}), (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})\big)$ generated by $\mathsf{IVC.P}(\mathsf{pk}, i, z_0, z_{i-1}, \omega_{i-1}, \pi_{i-1})$, by construction, we have

$$(u^*_{F,i}, w^*_{F,i}) \leftarrow \mathsf{Fold}_F.\mathsf{P}\big(\mathsf{pk}_F, (u_{F,i}, w_{F,i}), (u^*_{F,i-1}, w^*_{F,i-1})\big),$$
$$(u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i}) \leftarrow \mathsf{Fold.P}\big(\mathsf{pk}_{\mathsf{Fold}}, (u_{\mathsf{Fold},i-1}, w_{\mathsf{Fold},i-1}), (u^*_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1})\big).$$

Since $(u^*_{F,i-1}, w^*_{F,i-1})$, $(u_{\mathsf{Fold},i-1}, w_{\mathsf{Fold},i-1})$, and $(u^*_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1})$ are satisfying instance-witness pairs (implied by $\mathsf{IVC.V}$ in the previous round), and $(u_{F,i}, w_{F,i})$ is also a satisfying instance-witness pair (implied by the construction), we have that $(u^*_{F,i}, w^*_{F,i})$ and $(u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})$ are satisfying instance-witness pairs based on the perfect completeness of Fold_F and Fold.

Moreover, since $u_{\mathsf{Fold},i-1}.h = \mathrm{Hash}(\mathsf{vk}, i-1, z_0, z_{i-1}, u^*_{F,i-1}, u^*_{\mathsf{Fold},i-1})$ and $u_{\mathsf{Fold},i-1}$ is a non-relaxed instance (implied by the correctness of $\mathsf{IVC.V}$ in the previous round), the $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$ constructed by $\mathsf{tr}\big(\mathsf{CirFold}, (\mathsf{vk}, i, z_0, z_{i-1}, z_i, u_{F,i}, u^*_{F,i-1}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1})\big)$ is a satisfying pair where $u_{\mathsf{Fold},i}.h = \mathrm{Hash}(\mathsf{vk}, i, z_0, z_i, u^*_{F,i}, u^*_{\mathsf{Fold},i})$ and $u_{\mathsf{Fold},i}$ non-relaxed, by the perfect correctness of the underlying folding schemes. Thus, we have $\mathsf{IVC.V}(\mathsf{vk}, i, z_0, z_i, \pi_i) = 1$.

Knowledge Soundness. Let $\mathsf{pp} \leftarrow \mathsf{IVC.G}(1^\lambda)$. Given adversarially chosen (F) by an expected PPT adversary $\mathcal{A}$, generate the keys with $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{IVC.K}(\mathsf{pp}, F)$. $\mathcal{A}$ additionally outputs (z_0, z, π) such that $\mathsf{IVC.V}(\mathsf{vk}, i, z_0, z, \pi) = 1$ for a constant n with probability ϵ. We construct an expected polynomial-time extractor $\mathcal{E}$ that takes (pp, z_0, z) and outputs $(\omega_0, ..., \omega_{n-1})$ such that $z_i = F(z_{i-1}, \omega_{i-1})$ and $z_n = z$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

Specifically, we show $\mathcal{E}$ can be constructed inductively by an expected polynomial-time extractor $\mathcal{E}_i(\mathsf{pp}; \rho)$ that with probability $\epsilon - \mathsf{negl}(\lambda)$ outputs $\big((z_i, ..., z_{n-1}), (\omega_i, ..., \omega_{n-1}), \pi_i\big)$ such that $z_j = F(z_{j-1}, \omega_{j-1})$ and $\mathsf{IVC.V}(\mathsf{vk}, i, z_0, z_i, \pi_i) = 1$ for all $j \in \{i+1, ..., n\}$. This implies $z_n = z$ since $z_0 = z_i$ when $i = 0$ (implied by $\mathsf{IVC.V} = 1$ when $i = 0$) and $(\omega_0, ..., \omega_{n-1})$ extracted by $\mathcal{E}_0$ satisfies $z_i = F(z_{i-1}, \omega_{i-1})$ for all $i > 0$ (implied by $\mathsf{IVC.V} = 1$ when $i > 0$).

To construct $\mathcal{E}_{i-1}$, we assume there exists an $\mathcal{E}_i$ that satisfies the inductive hypothesis. Then we use $\mathcal{E}_i$ to construct an adversary $\mathcal{A}_{i-1}$ for the folding schemes. By further invoking two extractors for folding schemes, we can construct $\mathcal{E}_{i-1}$ that satisfies the inductive hypothesis. The detailed construction is as follows.

Base Case ($i = n$): $\mathcal{E}_n$ outputs $(\perp, \perp, \pi_n)$ where $\pi_n = \pi$ is the output of $\mathcal{A}$. By the premise, $\mathcal{E}_n$ succeeds with probability ϵ in expected polynomial-time.

Inductive Step ($i = n-1$ to 1): Suppose we have constructed $\mathcal{E}_i$ that outputs $\big((z_i, ..., z_{n-1}), (\omega_i, ..., \omega_{n-1}), \pi_i\big)$ and a π_i satisfies the inductive hypothesis. We first construct an adversary $\mathcal{A}_{i-1}$ for the folding schemes as follows:

$\mathcal{A}_{i-1}(\mathsf{pp}; \rho)$

1 : $\big((z_i, ..., z_{n-1}), (\omega_i, ..., \omega_{n-1}), \pi_i\big) \leftarrow \mathcal{E}_i(\mathsf{pp}; \rho)$,
2 : **parse** π_i **as** $\big((u^*_{F,i}, w^*_{F,i}), (u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i}), (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})\big)$,
3 : **parse** $w_{\mathsf{Fold},i}$ to retrieve $(u_{F,i}, u^*_{F,i-1}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1})$,
4 : **output** $\big((u_{F,i}, u^*_{F,i-1}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1}), (u^*_{F,i}, w^*_{F,i}), (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})\big)$.

The third step is correct since $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$ is derived from $\mathsf{tr}(\mathsf{CirFold}, *)$. By the inductive hypothesis, we have $\mathsf{IVC.V}(\mathsf{vk}, i, z_0, z_i, \pi_i) = 1$. This implies $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$ is a non-relaxed satisfying pair for $\mathsf{CirFold}$ relation in Equation (2). Therefore, $w_{\mathsf{Fold},i}$ must include $u^*_{F,i-1}$, $u_{\mathsf{Fold},i-1}$, and $u^*_{\mathsf{Fold},i-1}$.

Based on the construction of $\mathsf{CirFold}$ and the binding property of the hash function, we have

$$u^*_{F,i} \leftarrow \mathsf{Fold}_F.\mathsf{V}(\mathsf{vk}_F, u_{F,i}, u^*_{F,i-1}),$$
$$u^*_{\mathsf{Fold},i} \leftarrow \mathsf{Fold}.\mathsf{V}(\mathsf{vk}_{\mathsf{Fold}}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1}).$$

Therefore, $\mathcal{A}_{i-1}$ succeeds in producing satisfying folded instance-witness pairs $(u^*_{F,i}, w^*_{F,i})$ and $(u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})$ for instances $(u_{F,i}, u^*_{F,i-1})$ and $(u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1})$, respectively.

Based on the knowledge soundness of the folding schemes Fold_F and Fold, we can further invoke the extractors of them, $\mathcal{E}_F$ and $\mathcal{E}_{\mathsf{Fold}}$, which will output satisfying witnesses $(w_{F,i}, w^*_{F,i-1})$ for CirF and $(w_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1})$ for $\mathsf{CirFold}$ respectively in $(i-1)$-th round. z_{i-1} and ω_{i-1} will be included in $w_{F,i}$. The detailed construction of $\mathcal{E}_{i-1}$ is as follows:

$\mathcal{E}_{i-1}(\mathsf{pp}; \rho)$

1 : $\big((u_{F,i}, u^*_{F,i-1}, u_{\mathsf{Fold},i-1}, u^*_{\mathsf{Fold},i-1}),$
$\quad (u^*_{F,i}, w^*_{F,i}), (u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})\big) \leftarrow \mathcal{A}_{i-1}(\mathsf{pp}; \rho),$
2 : retrive $\big((z_i, ..., z_{n-1}), (\omega_i, ..., \omega_{n-1}), \pi_i\big)$ from $\mathcal{A}_{i-1}$'s internal states,
3 : $(w_{F,i}, w^*_{F,i-1}) \leftarrow \mathcal{E}_F(\mathsf{pp}; \rho)$, $\quad (w_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1}) \leftarrow \mathcal{E}_{\mathsf{Fold}}(\mathsf{pp}; \rho)$,
4 : **parse** $w_{F,i}$ to retrieve (z_{i-1}, ω_{i-1}),
5 : $\pi_{i-1} \leftarrow \big((u^*_{F,i-1}, w^*_{F,i-1}), (u_{\mathsf{Fold},i-1}, w_{\mathsf{Fold},i-1}), (u^*_{\mathsf{Fold},i-1}, w^*_{\mathsf{Fold},i-1})\big)$,
6 : **output** $\big((z_{i-1}, ..., z_{n-1}), (\omega_{i-1}, ..., \omega_{n-1}), \pi_{i-1}\big)$.

Since $\mathcal{E}_F$ and $\mathcal{E}_{\mathsf{Fold}}$ only incur a negligible soundness error, $\mathcal{E}_{i-1}$ succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$.

Now we argue the validity of the outputs. First, $(z_{i-1}, ..., z_{n-1})$ and $(\omega_{i-1}, ..., \omega_{n-1})$ are valid since $z_j = F(z_{j-1}, \omega_{j-1})$ for all $j \in \{i+1, ..., n\}$ by hypothesis, and $(u_{F,i}, w_{F,i})$ is a non-relaxed (by the validity of $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$) satisfying (by $\mathcal{E}_F$) pair, which implies $z_i = F(z_{i-1}, \omega_{i-1})$.

Next, we argue π_{i-1} is valid. Since $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$ satisfies $\mathsf{CirFold}$ in i-th round and $u_{\mathsf{Fold},i-1}$ is retrieved from $w_{\mathsf{Fold},i}$, we have $u_{\mathsf{Fold},i-1}.h = \mathrm{Hash}(\mathsf{vk}, i-1, z_0, z_{i-1}, u^*_{F,i-1}, u^*_{\mathsf{Fold},i-1})$ and $u_{\mathsf{Fold},i-1}$ are non-relaxed instances. Additionally, by the base case check of $\mathsf{CirFold}$, we ensure $z_{i-1} = z_0$ when $i = 1$. Since $\mathcal{E}_{i-1}$ succeeds with probability $\epsilon - \mathsf{negl}(\lambda)$, we have $\mathsf{IVC.V}(\mathsf{vk}, i-1, z_0, z_{i-1}, \pi_{i-1}) = 1$ with probability $\epsilon - \mathsf{negl}(\lambda)$.

Succinctness. The proof π_i contains three instance-witness pairs: $(u^*_{F,i}, w^*_{F,i})$, $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$, and $(u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})$. The folding schemes ensure $(u^*_{F,i}, w^*_{F,i})$ and $(u^*_{\mathsf{Fold},i}, w^*_{\mathsf{Fold},i})$ are of the size of one instance-witness pair, which are independent from i. $(u_{\mathsf{Fold},i}, w_{\mathsf{Fold},i})$ is also independent from i since it is a non-relaxed instance-witness pair. Therefore, the size of π_i does not increase with i.

C.3 Proof of Theorem 3

Perfect completeness. Given two augmented relaxed Groth16 instance-proof pairs, $\big((\mu_1, H_1, E_1, R_1, S_2), (A_1, B_1, C_1)\big)$ and

$\big((\mu_2, H_2, E_2, R_2, S_2), (A_2, B_2, C_2)\big)$, the folded pair $\big((\mu^*, H^*, E^*, R^*, S^*), (A^*, B^*, C^*)\big)$ is a satisfying pair since

$$
\begin{aligned}
& e(A^*, B^*) \cdot e(C^*, [\delta]_2)^{-\mu^*} \cdot e(H^*, [\gamma]_2)^{-\mu^*} \cdot D^{-\mu^{*2}} \\
=& e(A_1, B_1) \cdot e(C_1, [\delta]_2)^{-\mu_1} \cdot e(H_1, [\gamma]_2)^{-\mu_1} \cdot D^{-\mu_1^2} \\
& \cdot \big(e(A_1, B_2) \cdot e(A_2, B_1) \cdot e(C_1^{-\mu_2} C_2^{-\mu_1}, [\delta]_2) \\
& \quad \cdot e(H_1^{-\mu_2} H_2^{-\mu_1}, [\gamma]_2) \cdot D^{-2\mu_1\mu_2}\big)^r \\
& \cdot \Big(e(A_2, B_2) \cdot e(C_2, [\delta]_2)^{-\mu_2} \cdot e(H_2, [\gamma]_2)^{-\mu_2} \cdot D^{-\mu_2^2}\Big)^{r^2} \\
=& E_1 \cdot e(R_1, [\delta]_2) \cdot e(S_1, [\gamma]_2) \cdot D^{\kappa_1} \cdot \Big(T' \cdot e(R, [\delta]_2) \cdot e(S, [\gamma]_2) \cdot D^{\kappa}\Big)^r \\
& \cdot \Big(E_2 \cdot e(R_2, [\delta]_2) \cdot e(S_2, [\gamma]_2) \cdot D^{\kappa_2}\Big)^{r^2} \\
=& E^* \cdot e(R^*, [\delta]_2) \cdot e(S^*, [\gamma]_2) \cdot D^{\kappa^*},
\end{aligned}
$$

where $T' = e(A_1, B_2) \cdot e(A_2, B_1)$, $R = C_1^{-\mu_2} C_2^{-\mu_1}$, $S = H_1^{-\mu_2} H_2^{-\mu_1}$, and $\kappa = -2\mu_1\mu_2$.

Knowledge Soundness. Consider public parameters pp for $\mathsf{Fold}_{\mathsf{Gro16}}$, an adversarially chosen augmented relaxed Groth16 proof structure $([\delta]_2, [\gamma]_2, D) \in (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, and two adversarially chosen instances $(\mu_1, H_1, E_1, R_1, S_1, \kappa_1)$ and $(\mu_2, H_2, E_2, R_2, S_2, \kappa_2)$. We prove knowledge soundness of the *interactive* version of the protocol via the forking lemma in Lemma 1 (the corresponding non-interactive version can be proved under the Fiat-Shamir heuristic in the random oracle model): there exists a PPT extractor $\mathcal{E}$ such that when given pp, $([\delta]_2, [\gamma]_2, D)$, and a tree of accepting transcripts and the corresponding folded instance-proof pair, outputs two satisfying proofs with probability $1 - \mathsf{negl}(\lambda)$.

Specifically, when given 2 accepting transcripts with the same T, $\big(T, r_i, (A_i^*, B_i^*, C_i^*)\big)_{i=1}^2$, interpolate points $(\hat{A}_1, \hat{B}_1, \hat{C}_1)$ and $(\hat{A}_2, \hat{B}_2, \hat{C}_2)$ such that for all $i \in \{1, 2\}$,

$$
\hat{A}_1 \cdot \hat{A}_2^{r_i} = A_i^*, \quad \hat{B}_1 \cdot \hat{B}_2^{r_i} = B_i^*, \quad \hat{C}_1 \cdot \hat{C}_2^{r_i} = C_i^*.
$$

Now we show that the extracted proofs $(\hat{A}_1, \hat{B}_1, \hat{C}_1)$ and $(\hat{A}_2, \hat{B}_2, \hat{C}_2)$ are valid proofs to the corresponding instances. Since (A_i^*, B_i^*, C_i^*) are satisfying proofs for $i \in \{1, 2\}$, we have

$$
\begin{aligned}
& e(A_i^*, B_i^*) \cdot e(C_i^*, [\delta]_2)^{-\mu_i^*} \cdot e(H_i^*, [\gamma]_2)^{-\mu_i^*} \cdot D^{-(\mu_i^*)^2} \\
=& E_i^* \cdot e(R_i^*, [\delta]_2) \cdot e(S_i^*, [\gamma]_2) \cdot D^{\kappa_i^*}, \qquad \forall i \in \{1, 2\},
\end{aligned}
\tag{5}
$$

where $(\mu_i^*, H_i^*, E_i^*, R_i^*, S_i^*, \kappa_i^*)_{i=1}^2$ are computed based on $\mathsf{Fold}_{\mathsf{Gro16}}.\mathsf{V}$ by $\mathcal{E}$. The left-hand side of Equation (5) equals

$$\begin{aligned}
& e(A_i^*, B_i^*) \cdot e(C_i^*, [\delta]_2)^{-\mu_i^*} \cdot e(H_i^*, [\gamma]_2)^{-\mu_i^*} \cdot D^{-(\mu_i^*)^2} \\
=& e(\hat{A}_1 \hat{A}_2^{r_i}, \hat{B}_1 \hat{B}_2^{r_i}) \cdot e(\hat{C}_1 \hat{C}_2^{r_i}, [\delta]_2)^{-(\mu_1 + r_i \mu_2)} \\
& \cdot e(H_1 H_2^{r_i}, [\gamma]_2)^{-(\mu_1 + r_i \mu_2)} \cdot D^{-(\mu_1 + r_i \mu_2)^2} \\
=& e(\hat{A}_1, \hat{B}_1) \cdot e(\hat{C}_1, [\delta]_2)^{-\mu_1} \cdot e(H_1, [\gamma]_2)^{-\mu_1} \cdot D^{-\mu_1^2} \\
& \cdot \left(T' \cdot e(R, [\delta]_2) \cdot e(S, [\gamma]_2) \cdot D^{\kappa} \right)^{r_i} \\
& \cdot \left(e(\hat{A}_2, \hat{B}_2) \cdot e(\hat{C}_2, [\delta]_2)^{-\mu_2} \cdot e(H_2, [\gamma]_2)^{-\mu_2} \cdot D^{-\mu_2^2} \right)^{r_i^2}.
\end{aligned}$$

The right-hand side of Equation (5) equals

$$\begin{aligned}
& E_i^* \cdot e(R_i^*, [\delta]_2) \cdot e(S_i^*, [\gamma]_2) \cdot D^{\kappa_i^*} \\
=& \left(E_1 (T')^{r_i} E_2^{r_i^2} \right) \cdot e\left(R_1 R^{r_i} R_2^{r_i^2}, [\delta]_2 \right) \cdot e\left(S_1 S^{r_i} S_2^{r_i^2}, [\gamma]_2 \right) \cdot D^{\kappa_1 + r_i \kappa + r_i^2 \kappa_2} \\
=& E_1 \cdot e(R_1, [\delta]_2) \cdot e(S_1, [\gamma]_2) \cdot D^{\kappa_1} \cdot \left(T' \cdot e(R, [\delta]_2) \cdot e(S, [\gamma]_2) \cdot D^{\kappa} \right)^{r_i} \\
& \cdot \left(E_2 \cdot e(R_2, [\delta]_2) \cdot e(S_2, [\gamma]_2) \cdot D^{\kappa_2} \right)^{r_i^2}.
\end{aligned}$$

Since Equation (5) holds for $i \in \{1, 2\}$, by expanding and interpolating, we have the following relations hold with probability $1 - \mathsf{negl}(\lambda)$

$$\begin{aligned}
& e(\hat{A}_i, \hat{B}_i) \cdot e(\hat{C}_i, [\delta]_2)^{-\mu_i} \cdot e(H_i, [\gamma]_2)^{-\mu_i} \cdot D^{-\mu_i^2} \\
=& E_i \cdot e(R_i, [\delta]_2) \cdot e(S_i, [\gamma]_2), \qquad \forall i \in \{1, 2\},
\end{aligned}$$

which implies the extracted proofs $(\hat{A}_1, \hat{B}_1, \hat{C}_1)$ and $(\hat{A}_2, \hat{B}_2, \hat{C}_2)$ are valid proofs to the corresponding instances.