Springproofs: Efficient Inner Product Arguments for Vectors of Arbitrary Length

Jianning Zhang*†, Ming Su*†‡, Xiaoguang Liu*‡, and Gang Wang*†
*College of Computer Science & Cyber Science, Nankai University, Tianjin, China
†State Key Laboratory of Public Big Data, Guizhou University, Guizhou Guiyang, China
‡DISSec, SysNet, TMCC, GTIISC, Tianjin, China
Email: zhangjn@nbjl.nankai.edu.cn, nksuker@gmail.com*, liuxg@nbjl.nankai.edu.cn, wgzwg@163.com

***Abstract*—Inner product arguments (IPA) are arguments of knowledge that two committed vectors satisfy an inner product relation. With the recursive proof technique by Bootle et al. 2016, the size of IPA proofs only grows logarithmically in the length of the vectors, without a trusted setup. The succinct proof makes IPAs well suited for blockchain applications. However, current IPA can only handle a vector with length a power of 2, which limits the application of the argument. One direct solution is to pad the vectors with zeros, which incurs additional overhead. We propose Springproofs, a new framework deriving IPAs from many existing IPA schemes. Springproofs are natively compatible with vectors of arbitrary length. With a novel recursive compression structure, Springproofs achieve the same proof size as the original IPA but with more efficient computation. In particular, we instantiate Springproofs with Bulletproofs and find the optimal recursive structure for the IPA. First, we experimentally show that Springproofs are almost twice as fast as Bulletproofs for range proof, when the vector length is slightly larger than a power of 2. Afterwards, we incorporate the Springproofs into Monero, a popular cryptocurrency supporting privacy in transactions, revealing that the Springproofs based Monero outperforms Bulletproofs based Monero both in generating and verifying transactions. Moreover, we apply the Springproofs to the general arithmetic circuit, including SHA256, Merkle tree, and typical statistics, the performances on which are better than the performances by using Bulletproofs. Interestingly, Springproofs increase the range of parameters on which the performance of Bulletproofs exceeds that of Groth16, meanwhile naturally inherit the advantages of Bulletproofs, e.g., without initial trusted setup, aggregation, and batch verification. As a result, Springproofs have many promising applications, including confidential transactions in cryptocurrency and privacy computing for specific arithmetic circuits in smart contracts.**

***Index Terms*—zero-knowledge proof, inner product argument, range proof, privacy computing, blockchain**

1. Introduction

Inner product argument (IPA) is an argument of knowledge that the inner product of two committed vectors equals a given value. More specifically, if the openings of two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n$ satisfy $P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}u^{\langle \mathbf{a},\mathbf{b} \rangle}$, where $P \in \mathbb{G}$ is a given commitment, then (1) is the inner product relation.

$$\{(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}; \mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n) : P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}u^{\langle \mathbf{a},\mathbf{b} \rangle}\}. \quad (1)$$

An argument of knowledge of the relation (1) is called IPA.

IPAs have numerous applications. By reducing statements to inner product relations, IPAs can prove a wide range of statements from range proof to arithmetic circuit relation. One of the popular privacy preserving cryptocurrencies, Monero [1], has implemented confidential transactions with IPA. Another payment system Zether [2] also adopts an IPA for confidential and anonymous transactions on smart contracts.

Bulletproof [3], a notable IPA proposed by Bünz et al., is an efficient non-interactive zero-knowledge proof for range proof without a trusted setup. The proof sizes of Bulletproofs only grow logarithmically with respect to the length of vectors, thanks to the recursive technique by Bootle et al. [4]. Furthermore, Bulletproofs can aggregate several range proofs to reduce the average size of proofs. Because of the short proof size and aggregation, Bulletproofs based range proof is particularly appropriate for blockchain applications. Indeed, the IPAs used in Monero and Zether are Bulletproofs.

There are many continued works on Bulletproofs, each with improvements in proof size or computational cost. Following [4], many works focused on improving the IPA. Zhang et al. [5] considered another variant of inner product relation, and constructed an IPA with less communication cost compared with Bootle et al. Li et al. [6] proposed Shellproof that reduced computational cost by eliminating parts of the group exponentiations in the IPA of Bulletproofs. Chung et al. [7] proposed Bulletproofs+ that accelerated the range proof with a weighted inner product argument. Bulletproofs+ also generated smaller proofs compared to Bulletproofs. Daza et al. [8] proposed an improved IPA that was updatable and had logarithmic verification time.

However, current IPAs have a fundamental limitation, where the length of the witness vectors must be a power of 2 in the inner product relation. Since the bit lengths used in most systems are powers of 2, this is not a big problem in a single IPA based range proof. But due to the restriction of vector length, only a group of proofs

whose number is exactly a power of 2 can be aggregated. Moreover, considering an arithmetic circuit whose number of multiplication gates is not a power of 2, current IPAs cannot be directly used. Naturally, there is a straightforward countermeasure: padding the vector with zeros such that the length of the padded vector is a power of 2. However, the padding method brings additional computational costs. For vector lengths only marginally larger than powers of 2, the padding method nearly doubles the vector length in IPAs, which incurs redundant calculations.

In order to solve the problem, we propose Springproof, a framework to derive IPAs with native support for arbitrary vector length. Springproofs can achieve the same proof size as the original IPA and reduce the computational cost when the length of vectors is not a power of 2. Specifically, compared with the original Monero, Springproofs based Monero only takes 59.7% of time to generate a transaction with 9 outputs and 65.5% of time to verify it. Because of the efficiency of the arguments, Springproofs are suitable for many applications in blockchain. Some notable examples include range proof in confidential transactions of cryptocurrency, and privacy computing of arithmetic circuit in smart contract.

We summarize our contribution as follows.

IPA for arbitrary vector length. We revisit the basic IPA of Bulletproofs, and find that recursive IPAs can be constructed even only parts of the witnesses are involved in each recursive step. For constructing a recursive IPA, we introduce a *scheme function* controlling the recursive structure of the IPA. By carefully choosing proper scheme functions, we construct Springproofs to derive IPAs that support witness vectors of arbitrary length. We focus on Bulletproofs based Springproofs throughout the paper, and discuss Springproofs based on other IPAs.

Security analysis. With special-soundness, we analyze the security of Bulletproofs based Springproofs with different scheme functions, and give a sufficient condition for scheme function to derive a proper argument of knowledge. Furthermore, we also construct Springproofs under a zero-knowledge setting and prove that Springproof achieves zero-knowledge when instantiated with a zero-knowledge IPA.

Optimal scheme function. We analyze the performance characteristics of Bulletproofs based Springproofs with different scheme functions. Accordingly, we give lower bounds of the computational and communication cost, and find optimal scheme functions that minimize the computational cost and proof size.

Applications on Monero & Arithmetic circuit. We implement the range proof protocol with Bulletproofs based Springproofs, and integrate the protocol into Monero. Experimental results show that Springproofs only require around half of the computational cost in range proof compared with Bulletproofs when vector length is marginally larger than a power of 2. In digital currency scenario like Monero, Springproofs can achieve notable performance improvements as well. Moreover, we apply Springproofs to general arithmetic circuit, and find that the verification time by using Springproofs is always not more than that by using Bulletproofs, meanwhile the proof size of them is the same. Additionally, Springproofs increase the range of parameters on which the performance of Bulletproofs exceeds that of zk-SNARK by Groth [9] (*Groth16*), meanwhile naturally inherit the merits of Bulletproofs.

The paper is organized as follows. First we review and discuss concurrent works related to IPA in the introduction. Then, we introduce the basic notions and definitions in Section 2. Afterwards, we give a detailed construction and analysis of Springproofs in Section 3. Finally, we evaluate the performance of Springproofs in Section 4.

2. Preliminaries

2.1. Notations

Let $\mathbb{Z}$ denote integers, $\mathbb{G}$ denote a cyclic group whose order is a prime p, $\mathbb{F}_p$ denote a finite field with the same prime order p, and $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$, $\mathbb{Z}^+ = \{x \in \mathbb{Z} \mid x > 0\}$. Let blackboard font with superscript k denote k-dimensional vector space, and bold font denote vectors. Let $(i : j)$ denote a set $\{i, i+1, \ldots, j\}$ where $i, j \in \mathbb{Z}^+, i \le j$. The power set of a set A is denoted by $\mathscr{P}(A)$. Without ambiguity, $x \in \mathbb{F}_p^*$ are uniformly distributed challenges, and $x \xleftarrow{\$} \mathbb{F}_p^*$ denotes uniformly sampling x over $\mathbb{F}_p^*$. $\dim(\mathbf{a})$ denotes the dimension of a vector $\mathbf{a}$. $|A|$ denotes the cardinality of a set A. $A \sqcup B$ denotes non-overlapping union of two sets A and B.

Furthermore, let $\mathbf{a} \parallel \mathbf{b}$ be the concatenation of two vectors $\mathbf{a}$ and $\mathbf{b}$. Let $\langle \mathbf{a}, \mathbf{b} \rangle$ be the inner product of two vectors $\mathbf{a}$ and $\mathbf{b}$. Let $\mathbf{a} \circ \mathbf{b}$ be the element-wise product of two vectors $\mathbf{a}$ and $\mathbf{b}$. For $x \in \mathbb{F}_p$, $k \in \mathbb{Z}$, let $\mathbf{x}^k$ be a k-dimensional vector $(x^0, x^1, x^2, \ldots, x^{k-1}) \in \mathbb{Z}_p^k$. For example, $\mathbf{2}^k = (1, 2, 4, \ldots, 2^k)$. We define $\mathbf{g}^{\mathbf{a}} \coloneqq \prod_{i=1}^{n} g_i^{a_i}$ and $\mathbf{g}^{-1} \coloneqq (g_1^{-1}, g_2^{-1}, \ldots, g_n^{-1})$, where $\mathbf{g} \in \mathbb{G}^n$ and $\mathbf{a} \in \mathbb{F}_p^n$. We use square brackets in the subscript of a vector to denote a new vector consisting of selected components. For instance, for $I \subseteq (1 : \dim(\mathbf{a}))$, $\mathbf{a}_{[I]}$ denotes selecting components of vector $\mathbf{a}$ with index in I, then combining those components into a new vector sequentially. For clarity, we write $\mathbf{a}_{[i:j]} \coloneqq \mathbf{a}_{[(i:j)]}$ for any $1 \le i \le j \le \dim(\mathbf{a})$. Similarly, we use square brackets in the subscript of a matrix to denote a new matrix consisting of selected rows.

Throughout the paper, we write $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n$, $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$, $u, P, Q, L, R \in \mathbb{G}$, and $S, T \in \mathscr{P}(\mathbb{Z}^+)$.

Finally, let $\{(\mathbb{x}; \mathbb{w}) : Con\}$ be a relation with constraints Con using the public input $\mathbb{x}$ and the witness $\mathbb{w}$.

2.2. Security Assumption

Definition 1 (Discrete Logarithm Relation Assumption). For all $n \ge 2$ and all probabilistic polynomial time (PPT) adversaries $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$ such that

$$\Pr\left[\begin{matrix}\mathbb{G} = \mathrm{Setup}(1^\lambda), \mathbf{g} \xleftarrow{\$} \mathbb{G}^n; \\ \mathbf{a} \in \mathbb{F}_p^n \leftarrow \mathcal{A}(\mathbb{G}, \mathbf{g})\end{matrix} \middle| \begin{matrix}\exists a_i \neq 0 \wedge \\ \mathbf{g}^{\mathbf{a}} = 1\end{matrix}\right] \le \mu(\lambda).$$

We call $\mathbf{g}^{\mathbf{a}} = 1$ a non-trivial discrete logarithm relation of $\mathbf{g}$. If the discrete logarithm relation assumption holds, then any PPT adversary can not find a non-trivial discrete logarithm relation for $\mathbf{g} \xleftarrow{\$} \mathbb{G}^n$.

2.3. Arguments of knowledge

Consider three interactive algorithms which run in probabilistic polynomial time: the common reference string generator Setup, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$. The triple $\mathit{\Pi} \coloneqq (\mathrm{Setup}, \mathcal{P}, \mathcal{V})$ forms an argument. Algorithm Setup maps 1^λ to the common reference string σ, where λ is the security parameter such that $\mu(\lambda)$ is negligible. We use $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t) \rangle$ to denote the transcripts of $\mathcal{P}$ and $\mathcal{V}$ with input s and t respectively. We write $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle = 1$ if the verifier accepts, and $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle = 0$ if the verifier rejects.

Denote by $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ a polynomial-time-decidable binary relation. $\mathbb{w}$ is a witness for the statement $\mathbb{x}$ if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$. We define the NP language as $\mathcal{L} \coloneqq \{\mathbb{x} \mid \exists \mathbb{w} : (\mathbb{x}, \mathbb{w}) \in \mathcal{R}\}$.

Intuitively, if for any $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the verifier accepts an honest execution of the argument with high probability, then the argument has *completeness*. For any statement $\mathbb{x}$, suppose a prover convinces the verifier with probability higher than a negligible *knowledge error* $\kappa(|\mathbb{x}|)$, we say the argument has *knowledge soundness*, if there exists an extractor efficiently extracting a valid witness from the prover. If for any $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, there exists an efficient simulator that given the public input $\mathbb{x}$ and the randomness used by the verifier, simulates a transcript indistinguishable from the real transcript between the prover and the verifier, we say the argument is *honest-verifier zero-knowledge (HVZK)*. The formal definitions of completeness, knowledge soundness and HVZK can be found in Appendix A.

Definition 2 (Argument of Knowledge). An argument $(\mathrm{Setup}, \mathcal{P}, \mathcal{V})$ is an *argument of knowledge* of the relation $\mathcal{R}$ if it is complete and knowledge sound. If the argument is HVZK, we say it is a *zero-knowledge argument* as well.

For the knowledge soundness, if we limit the prover to be a PPT adversary, then the argument (interactive or non-interactive) is *computational knowledge sound*. A knowledge sound argument without such a requirement is a *statistical knowledge sound* argument. Proof systems with computational knowledge soundness and statistical knowledge soundness are generally called arguments of knowledge and proofs of knowledge, respectively. Since all the proof systems we discuss only satisfy computational knowledge soundness, we use "proof" and "argument" interchangeably throughout the paper.

Definition 3 (Public Coin). For $(\mathrm{Setup}, \mathcal{P}, \mathcal{V})$ that is an argument of knowledge, if all messages sent from $\mathcal{V}$ are sampled independently of those from $\mathcal{P}$, and are sampled uniformly at random, the argument is called a public coin argument.

By applying the Fiat-Shamir transformation [10] to a public-coin interactive argument of knowledge, i.e. making the prover and verifier fetch the challenges from a random oracle instead of the verifier, we have a new *non-interactive random oracle argument* for the same relation.

To estimate the knowledge error of a public-coin argument of knowledge, we consider a property called *special soundness*, implying that a witness for the statement can be extracted from a transcript tree of the argument efficiently.

Definition 4 ($(k_1, k_2, \ldots, k_m)$-Transcript-Tree). Given an interactive public-coin argument of knowledge, a $(k_1, k_2, \ldots, k_m)$-transcript-tree is a depth-m tree that consists of accepting transcripts between the prover and the verifier. In the tree, every vertex corresponds to a message from the prover, and every edge corresponds to a challenge from the verifier. At depth i of the tree, every vertex has exactly k_i distinct children. In total, there are $\prod_{i=1}^{m} k_i$ leaves. Every path from the root to a leaf corresponds to a valid transcript.

Definition 5 ($(k_1, k_2, \ldots, k_m)$-Special-Soundness). A $(2m + 1)$-move public coin argument of knowledge for relation $\mathcal{R}$ is $(k_1, k_2, \ldots, k_m)$-*special-sound*, if for any common reference string $\sigma \leftarrow \mathrm{Setup}(1^\lambda)$ and statement $\mathbb{x} \in \mathcal{L}$, there is an efficient polynomial time algorithm that takes a $(k_1, k_2, \ldots, k_m)$-transcript-tree as input, and output a witness $\mathbb{w}$, such that $(\sigma, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$.

Attema et al. [11] [12] analyzed detailedly on the knowledge error of special sound arguments in the following Lemma 1 and Lemma 2.

Lemma 1. [11, Lemma 3] Let $(\mathrm{Setup}, \mathcal{P}, \mathcal{V})$ be a $(k_1, k_2, \ldots, k_m)$-special-sound interactive protocol for relation R, where all challenges from $\mathcal{V}$ are chosen uniformly from a challenge set. Given a common reference string $\sigma \leftarrow \mathrm{Setup}(1^\lambda)$ and a statement $\mathbb{x}$, and a prover $\mathcal{P}^*$ such that $\epsilon(\mathbb{x}) \coloneqq \Pr\left[\langle \mathcal{P}^*(\sigma, \mathbb{x}), \mathcal{V}(\sigma, \mathbb{x}) \rangle = 1\right] > \kappa_0$ where

$$\kappa_0 = \frac{\sum_{i=1}^{m} (k_i - 1) q^{m-i} \prod_{j=1}^{i-1} (q - k_j + 1)}{q^m} \leq \frac{\sum_{i=1}^{m} (k_i - 1)}{q}.$$

Then there exists a polynomial time extractor $\mathcal{E}$, such that $\Pr[(\mathbb{x}, \mathcal{E}_{\mathcal{P}^*}(\mathbb{x})) \in \mathcal{R}] \geq (\epsilon(\mathbb{x}) - \kappa_0)^K$ where $K = \prod_{i=1}^{m} k_i$, and q is the size of challenge set, then extractor has a rewindable black-box oracle access to $\mathcal{P}^*$ and only calls $\mathcal{P}^*$ at most K times in the extraction.

Lemma 2. [12, Theorem 3] The Fiat-Shamir transformation [10] of a $(k_1, k_2, \ldots, k_m)$-special-sound interactive argument is knowledge sound with knowledge error $(Q+1) \cdot \kappa$, where Q is the number of queries to random oracle and κ is the knowledge error of the interactive argument.

Lemma 1 and Lemma 2 give an estimate of knowledge error for special sound argument of knowledge. Therefore, we mainly focus on the special soundness in the discussion of knowledge soundness of Springproofs.

3. Inner Product Argument for Vectors of Arbitrary Length

3.1. Recursive Inner Product Argument

For the inner product relation (1), a naive IPA sends $\mathbf{a}, \mathbf{b}$ to the verifier directly. The verifier accepts if $\mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}u^{\langle \mathbf{a},\mathbf{b}\rangle} = P$, and rejects otherwise. But the proof size of the naive argument is $2n$ elements over $\mathbb{F}_p$, which is linear dependent with the vector length.

For building a communication efficient IPA, the essential technique is the recursive proof by Bootle et al. [4], whose idea is to start from a basic IPA that decreases the proof size by reducing the relation to a new inner product relation. As long as the new relation can be proved with the basic IPA, we can reduce the proof size further by running the basic IPA recursively.

The basic IPA in Bulletproofs is shown in Algorithm 1, which reduces the proof size from $2n$ elements over $\mathbb{F}_p$ to n elements over $\mathbb{F}_p$ and 2 elements over $\mathbb{G}$. Notice that $\mathbf{g}', \mathbf{h}', P', \mathbf{a}', \mathbf{b}'$ in step 5, 6 and 8 of Algorithm 1 form a fresh input of Algorithm 1 if $n/2$ is even. Therefore, by running the Algorithm 1 recursively instead of sending $\mathbf{a}', \mathbf{b}'$ to the verifier directly, the proof size is further decreased. Eventually, within $O(\log n)$ steps of Algorithm 1, the recursive IPA achieves the effect of compressing two vectors over $\mathbb{F}_p^n$ into two elements over $\mathbb{F}_p$, at the cost of sending additional $2\log n$ elements over $\mathbb{G}$. It is worth noting that Algorithm 1 is not zero-knowledge, because L, R in step 2 contains information of $\mathbf{a}, \mathbf{b}$.

Algorithm 1 Basic IPA of Bulletproofs

Input: $(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}; \mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n)$, where n is even.
Prover's input: $(\mathbf{g}, \mathbf{h}, u, P; \mathbf{a}, \mathbf{b})$
Verifier's input: $(\mathbf{g}, \mathbf{h}, u, P)$
Output: Verifier accepts or rejects

1: Prover computes:
$n' \leftarrow n/2$
$c_L \leftarrow \langle \mathbf{a}_{[1:n']}, \mathbf{b}_{[n'+1:n]} \rangle \in \mathbb{F}_p$
$c_R \leftarrow \langle \mathbf{a}_{[n'+1:n]}, \mathbf{b}_{[1:n']} \rangle \in \mathbb{F}_p$
$L \leftarrow \mathbf{g}_{[n'+1:n]}^{\mathbf{a}_{[1:n']}} \mathbf{h}_{[1:n']}^{\mathbf{b}_{[n'+1:n]}} u^{c_L} \in \mathbb{G}$
$R \leftarrow \mathbf{g}_{[1:n']}^{\mathbf{a}_{[n'+1:n]}} \mathbf{h}_{[n'+1:n]}^{\mathbf{b}_{[1:n']}} u^{c_R} \in \mathbb{G}$
2: Prover $\rightarrow$ Verifier: L, R
3: Verifier: $x \xleftarrow{\$} \mathbb{F}_p^*$
4: Verifier $\rightarrow$ Prover: x
5: Prover and Verifier compute:
$\mathbf{g}' \leftarrow \mathbf{g}_{[1:n']}^{x^{-1}} \circ \mathbf{g}_{[n'+1:n]}^{x} \in \mathbb{G}^{n'}$
$\mathbf{h}' \leftarrow \mathbf{h}_{[1:n']}^{x} \circ \mathbf{h}_{[n'+1:n]}^{x^{-1}} \in \mathbb{G}^{n'}$
6: Prover computes:
$\mathbf{a}' \leftarrow x\mathbf{a}_{[1:n']} + x^{-1}\mathbf{a}_{[n'+1:n]} \in \mathbb{F}_p^{n'}$
$\mathbf{b}' \leftarrow x^{-1}\mathbf{b}_{[1:n']} + x\mathbf{b}_{[n'+1:n]} \in \mathbb{F}_p^{n'}$
7: Prover $\rightarrow$ Verifier: $\mathbf{a}', \mathbf{b}'$
8: Verifier computes:
$P' \leftarrow L^{x^2} P R^{x^{-2}} \in \mathbb{G}$

In fact, Algorithm 1 is one round of Bulletproofs, and the verifier accepts if $\mathbf{g}'^{\mathbf{a}'}\mathbf{h}'^{\mathbf{b}'}u^{\langle \mathbf{a}',\mathbf{b}'\rangle} = P'$, and rejects otherwise. In the complete Bulletproofs, Algorithm 1 runs recursively for many times until the length of input becomes 1.

3.2. Springproofs Based on Bulletproofs

As stated above, Algorithm 1 is an IPA. However, it does not achieve zero-knowledge, which indicates that the Algorithm 1 can be viewed as a simple compression step in the whole recursive argument. Instead of compressing all components at once, the prover may intentionally reveal some components of the vectors to the verifier and only run the IPA on the rest of components.

This idea leads to a hybrid IPA that combines the naive IPA and Algorithm 1. Consider a partition of the index set of n-dimensional vectors $S \sqcup T = (1:n)$ where $|T|$ is even. Given such a partition, we can construct a hybrid IPA. In the argument, components $\mathbf{a}_{[S]}, \mathbf{b}_{[S]}$ whose indices are in S are directly sent to the verifier, while other components $\mathbf{a}_{[T]}, \mathbf{b}_{[T]}$ with indices in T are compressed with Algorithm 1.

For compressing $\mathbf{a}_{[T]}, \mathbf{b}_{[T]}$ with Algorithm 1, notice that

$$Q = P\mathbf{g}_{[S]}^{-\mathbf{a}_{[S]}}\mathbf{h}_{[S]}^{-\mathbf{b}_{[S]}}u^{-\langle \mathbf{a}_{[S]},\mathbf{b}_{[S]}\rangle} \tag{2}$$

is a commitment of $\mathbf{a}_{[T]}, \mathbf{b}_{[T]}$ that is consistent with (1). By using $(\mathbf{g}_{[T]}, \mathbf{h}_{[T]}, u, Q, \mathbf{a}_{[T]}, \mathbf{b}_{[T]})$ as input of Algorithm 1, we achieve the proof of knowledge for (1), while only parts of vectors, $\mathbf{a}_{[T]}$ and $\mathbf{b}_{[T]}$, are compressed. The detailed process is shown in Algorithm 2. Such an argument reduces the proof size from $2n$ elements over $\mathbb{F}_p$ to $2n-|T|$ elements over $\mathbb{F}_p$ and 2 elements over $\mathbb{G}$.

Algorithm 2 Basic IPA of Springproofs instantiated with $S \sqcup T = (1:n)$

Input: $(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}; \mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n)$
Prover's input: $(\mathbf{g}, \mathbf{h}, u, P; \mathbf{a}, \mathbf{b})$
Verifier's input: $(\mathbf{g}, \mathbf{h}, u, P)$
Output: Verifier accepts or rejects

1: Prover $\rightarrow$ Verifier: $\mathbf{a}_{[S]}, \mathbf{b}_{[S]}$
2: Prover and Verifier compute Q as (2)
3: Run algorithm 1 with input $(\mathbf{g}_{[T]}, \mathbf{h}_{[T]}, u, Q; \mathbf{a}_{[T]}, \mathbf{b}_{[T]})$

Notice that in Algorithm 2, with $\mathbf{g}', \mathbf{h}', P', \mathbf{a}', \mathbf{b}'$ in step 5, 6 and 8 of Algorithm 1, we have

$$\begin{aligned} &\left(\mathbf{g}_{[S]} \parallel \mathbf{g}', \mathbf{h}_{[S]} \parallel \mathbf{h}', u, Q'; \mathbf{a}_{[S]} \parallel \mathbf{a}', \mathbf{b}_{[S]} \parallel \mathbf{b}'\right), \\ &\text{where } Q' = P'\mathbf{g}_{[S]}^{\mathbf{a}_{[S]}}\mathbf{h}_{[S]}^{\mathbf{b}_{[S]}}u^{\langle \mathbf{a}_{[S]},\mathbf{b}_{[S]}\rangle}, \end{aligned} \tag{3}$$

which is a fresh input of Algorithm 2 as well. Therefore, the recursive technique can also be applied to further reduce the proof size.

Combined with padding, we construct Springproofs, a framework to derive IPAs. In Springproofs, the IPA proceeds recursively and takes the output of the previous step as the input of the next step, until the length of intermediate

witnesses becomes 1. In each recursive step, the protocol performs one of the following,

- *padding*. Pad $\mathbf{a}$ and $\mathbf{b}$ with certain number of $0 \in \mathbb{F}_p$ at the end of the vectors to extend the length of vectors;
- *compression*. Run Algorithm 2 with input $\mathbf{a}, \mathbf{b}$ to fold the vectors by half.

To give a formal description of Springproofs, we introduce the *scheme function* $f : \mathbb{Z}^+ \to \mathbb{Z}^+ \sqcup \mathscr{P}(\mathbb{Z}^+)$, that maps a vector length n to either a number, implying that we will pad the number of zeros after the input vector; or an index set with even cardinality belonging to $(1 : n)$, implying that we will compress the input vector into a half by Algorithm 2. With f, one can instantiate an IPA from the Springproofs framework. Denote by $\mathsf{SIPA}(f)$ a Springproof with scheme function f. Algorithm 3 shows an instantiation $\mathsf{SIPA}(f)$ with a proper scheme function f. Notice that the initial value of the total number of padding zeros is uniquely determined by the length n of witness vectors and the scheme function f. For example, if $\mathsf{SIPA}(f)$ is instantiated as the widely used padding method (see Section 3.4.1 for the detailed scheme function), the initial value of n_p is $2^{\lceil \log n \rceil} - n$. Similarly, for other f and n, the initial value of n_p is determined by the flow of Algorithm 3 and counting all the padding zeros.

Algorithm 3 A Springproof $\mathsf{SIPA}(f)$ that instantiated with a scheme function $f : \mathbb{Z}^+ \to \mathbb{Z}^+ \sqcup \mathscr{P}(\mathbb{Z}^+)$

Input: $(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, \mathbf{g}_\mathrm{p}, \mathbf{h}_\mathrm{p} \in \mathbb{G}^{n_\mathrm{p}}, u, P \in \mathbb{G}; \mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n)$
where n_p is the total number of padding zeros of $\mathsf{SIPA}(f)$ when running with n-dimensional witnesses.
Prover's input: $(\mathbf{g}, \mathbf{h}, \mathbf{g}_\mathrm{p}, \mathbf{h}_\mathrm{p}, u, P; \mathbf{a}, \mathbf{b})$
Verifier's input: $(\mathbf{g}, \mathbf{h}, \mathbf{g}_\mathrm{p}, \mathbf{h}_\mathrm{p}, u, P)$
Output: Verifier accepts or rejects

while $n \neq 1$ **do**
  if $f(n) \in \mathbb{Z}^+$ **then**
    Prover compute: // Padding step
    $\mathbf{a}' \leftarrow \mathbf{a} \parallel \mathbf{0}^{f(n)},\ \mathbf{b}' \leftarrow \mathbf{b} \parallel \mathbf{0}^{f(n)}$
    Prover and Verifier compute:
    $\mathbf{g}' \leftarrow \mathbf{g} \parallel \mathbf{g}_{\mathrm{p}[1:f(n)]},\ \mathbf{h}' \leftarrow \mathbf{h} \parallel \mathbf{h}_{\mathrm{p}[1:f(n)]}$,
    $\mathbf{g}_\mathrm{p} \leftarrow \mathbf{g}_{\mathrm{p}[f(n)+1:n_\mathrm{p}]},\ \mathbf{h}_\mathrm{p} \leftarrow \mathbf{h}_{\mathrm{p}[f(n)+1:n_\mathrm{p}]}$,
    $n_\mathrm{p} \leftarrow \dim(\mathbf{g}_\mathrm{p}),\ P' \leftarrow P$
  else if $f(n) \in \mathscr{P}(\mathbb{Z}^+)$ **then**
    Prover and Verifier compute: // Compression step
    $T \leftarrow f(n),\ S \leftarrow (1 : n) \setminus f(n)$
    $n' \leftarrow |T|/2,\ l \leftarrow (1 : n'), r \leftarrow (n' + 1 : |T|)$
    $\mathbf{g}_t \leftarrow \mathbf{g}_{[T]},\ \mathbf{h}_t \leftarrow \mathbf{h}_{[T]}$
    Prover compute:
    $\mathbf{c} \leftarrow \mathbf{a}_{[T]},\ \mathbf{d} \leftarrow \mathbf{b}_{[T]}$
    $c_L \leftarrow \langle \mathbf{c}_{[l]}, \mathbf{d}_{[r]} \rangle \in \mathbb{F}_p$
    $c_R \leftarrow \langle \mathbf{c}_{[r]}, \mathbf{d}_{[l]} \rangle \in \mathbb{F}_p$
    $L \leftarrow \mathbf{g}_{t_{[r]}}^{\mathbf{c}_{[l]}} \mathbf{h}_{t_{[l]}}^{\mathbf{d}_{[r]}} u^{c_L} \in \mathbb{G}$
    $R \leftarrow \mathbf{g}_{t_{[l]}}^{\mathbf{c}_{[r]}} \mathbf{h}_{t_{[r]}}^{\mathbf{d}_{[l]}} u^{c_R} \in \mathbb{G}$
    Prover $\to$ Verifier: L, R
    Verifier: $x \xleftarrow{\$} \mathbb{F}_p^*$
    Verifier $\to$ Prover: x
    Prover and Verifier compute:
    $\mathbf{g}' \leftarrow \mathbf{g}_{[S]} \parallel (\mathbf{g}_{t_{[l]}}^{x^{-1}} \circ \mathbf{g}_{t_{[r]}}^{x}) \in \mathbb{G}^{n-n'}$
    $\mathbf{h}' \leftarrow \mathbf{h}_{[S]} \parallel (\mathbf{h}_{t_{[l]}}^{x} \circ \mathbf{h}_{t_{[r]}}^{x^{-1}}) \in \mathbb{G}^{n-n'}$
    Prover computes:
    $\mathbf{a}' \leftarrow \mathbf{a}_{[S]} \parallel (x\mathbf{c}_{[l]} + x^{-1}\mathbf{c}_{[r]}) \in \mathbb{F}_p^{n-n'}$
    $\mathbf{b}' \leftarrow \mathbf{b}_{[S]} \parallel (x^{-1}\mathbf{d}_{[l]} + x\mathbf{d}_{[r]}) \in \mathbb{F}_p^{n-n'}$
    Verifier compute:
    $P' \leftarrow L^{x^2} P R^{x^{-2}} \in \mathbb{G}$
  end if
  Prover and Verifier compute:
  $n \leftarrow \dim(\mathbf{g}'),\ \mathbf{g} \leftarrow \mathbf{g}',\ \mathbf{h} \leftarrow \mathbf{h}'$
  Prover computes: $\mathbf{a} \leftarrow \mathbf{a}', \mathbf{b} \leftarrow \mathbf{b}'$
  Verifier computes: $P \leftarrow P'$
end while
Prover $\to$ Verifier: $\mathbf{a} = (a) \in \mathbb{F}_p^1, \mathbf{b} = (b) \in \mathbb{F}_p^1$
// Now $\dim(\mathbf{a}) = \dim(\mathbf{b}) = \dim(\mathbf{g}) = \dim(\mathbf{h}) = 1$
Verifier accepts if $\mathbf{g}^\mathbf{a}\mathbf{h}^\mathbf{b}u^{ab} = P$, rejects otherwise

Given a proper scheme function f, it is clear that $\mathsf{SIPA}(f)$ is compatible with vectors of any length. In fact, the padding method is a special case of Springproofs. We will discuss this in detail in Section 3.4.1.

Note that with an arbitrary f, $\mathsf{SIPA}(f)$ may not halt. For example, let $f(n) = 1$, then $\mathsf{SIPA}(f)$ will keep padding the vectors and never halt. We only consider f that instantiates a terminative argument for any $n \in \mathbb{Z}^+$.

Theorem 3 reveals that with some constraints, Algorithm 3 is an argument of knowledge for inner product relation (1).

Theorem 3 (Springproofs). Given a terminative $\mathsf{SIPA}(f)$, if the number of compression steps in $\mathsf{SIPA}(f)$ is $O(\log n)$, then $\mathsf{SIPA}(f)$ is a complete and computational knowledge sound argument of relation (1). Moreover, the Fiat-Shamir transformation of $\mathsf{SIPA}(f)$ is a non-interactive random oracle argument having completeness and computational knowledge soundness as well.

Proof: A $\mathsf{SIPA}(f)$ with $m = \log(n)$ steps of compression is a $(2m+1)$-move interactive protocol. In the following proof, we first consider the case of $m = 0$, then prove the general cases by mathematical induction.

Completeness. If $m = 0$, then $P = g^a h^b u^{ab}$. Hence, $\mathsf{SIPA}(f)$ is trivially complete. Suppose $\mathsf{SIPA}(f)$ is complete when $m = m'$. For $m = m' + 1$, consider the first compression step. There could be i padding steps before the step $i + 1$ for compression. Let n_z be the total number of padding zeros in the previous i padding steps. Let $(\mathbf{g}_0, \mathbf{h}_0, P_0; \mathbf{a}_0, \mathbf{b}_0)$ be the public input and the witnesses in the original inner product relation. Then before the first compression step, we have

$$\mathbf{a} = \mathbf{a}_0 \parallel \mathbf{0}^{n_z} \in \mathbb{F}_p^{n+n_z}, \qquad \mathbf{b} = \mathbf{b}_0 \parallel \mathbf{0}^{n_z} \in \mathbb{F}_p^{n+n_z},$$
$$\mathbf{g} = \mathbf{g}_0 \parallel \mathbf{g}_{\mathrm{p}[1:n_z]} \in \mathbb{G}^{n+n_z}, \quad \mathbf{h} = \mathbf{h}_0 \parallel \mathbf{h}_{\mathrm{p}[1:n_z]} \in \mathbb{G}^{n+n_z}.$$

Notice that $P_0 = \mathbf{g}_0^{\mathbf{a}_0}\mathbf{h}_0^{\mathbf{b}_0}u^{\langle \mathbf{a}_0, \mathbf{b}_0 \rangle} = \mathbf{g}^\mathbf{a}\mathbf{h}^\mathbf{b}u^{\langle \mathbf{a}, \mathbf{b} \rangle}$, and padding setup does not modify P. Therefore, for the first

compression step we have $P = P_0$ and

$$\begin{aligned}
&\mathbf{g}'^{\mathbf{a}'}\mathbf{h}'^{\mathbf{b}'}u^{\langle\mathbf{a}',\mathbf{b}'\rangle}\\
=&\mathbf{g}_{t[l]}^{\mathbf{c}_{[l]}+x^{-2}\mathbf{c}_{[r]}}\mathbf{g}_{t[r]}^{x^2\mathbf{c}_{[l]}+\mathbf{c}_{[r]}}\mathbf{h}_{t[l]}^{\mathbf{d}_{[l]}+x^2\mathbf{d}_{[r]}}\mathbf{h}_{t[r]}^{x^{-2}\mathbf{d}_{[l]}+\mathbf{d}_{[r]}}\\
&u^{x^2\langle\mathbf{c}_{[l]},\mathbf{d}_{[r]}\rangle+\langle\mathbf{c},\mathbf{d}\rangle+x^{-2}\langle\mathbf{c}_{[r]},\mathbf{d}_{[l]}\rangle}\mathbf{g}_{[S]}^{\mathbf{a}_{[S]}}\mathbf{h}_{[S]}^{\mathbf{b}_{[S]}}u^{\langle\mathbf{a}_{[S]},\mathbf{b}_{[S]}\rangle}\\
=&(\mathbf{g}_{t[r]}^{\mathbf{c}_{[l]}}\mathbf{h}_{t[l]}^{\mathbf{d}_{[r]}}u^{\langle\mathbf{c}_{[l]},\mathbf{d}_{[r]}\rangle})^{x^2}(\mathbf{g}_{t[l]}^{\mathbf{c}_{[r]}}\mathbf{h}_{t[r]}^{\mathbf{d}_{[l]}}u^{\langle\mathbf{c}_{[r]},\mathbf{d}_{[l]}\rangle})^{x^{-2}}\\
&(\mathbf{g}_{[S]}^{\mathbf{a}_{[S]}}\mathbf{h}_{[S]}^{\mathbf{b}_{[S]}}u^{\langle\mathbf{a}_{[S]},\mathbf{b}_{[S]}\rangle}\cdot\mathbf{g}_{[T]}^{\mathbf{a}_{[T]}}\mathbf{h}_{[T]}^{\mathbf{b}_{[T]}}u^{\langle\mathbf{a}_{[T]},\mathbf{b}_{[T]}\rangle})\\
=&L^{x^2}R^{x^{-2}}P_0 = P'.
\end{aligned}$$

The rest of the argument forms a $(2m'+1)$-move $\mathsf{SIPA}(f)$ with input $(\mathbf{g}', \mathbf{h}', P'; \mathbf{a}', \mathbf{b}')$. By mathematical induction, we conclude that $\mathsf{SIPA}(f)$ has completeness. For non-interactive $\mathsf{SIPA}(f)$ derived by Fiat-Shamir transformation, the statement holds analogously.

Computational Knowledge Soundness. We first discuss the special soundness of $\mathsf{SIPA}(f)$. Consider a $(k_1, k_2, \ldots, k_m)$-transcript-tree of $\mathsf{SIPA}(f)$, where $k_i = 4$, $(i \in (1:m))$. For a vertex in depth $d \in (1 : m-1)$, the vertex itself corresponds to the prover message L, R, and the four edge from the depth-d vertex to the children vertices correspond to different challenges from the verifier $x_{(j)}$ $(j \in (1:4))$. Notice that every vertex corresponds to a few padding steps followed by a compression step in the argument. The value of $\mathbf{g}$, $\mathbf{h}$, $\mathbf{g}_t$, $\mathbf{h}_t$, P, S, T, n, n', l and r in the corresponding compression step can be recovered by following the path from root to the vertex and calculating as the argument.

If $d = m-1$, then the intermediate witnesses $\mathbf{a}'^{(j)}$, $\mathbf{b}'^{(j)}$ $(j \in (1:4))$ of the children vertices are directly obtained from the leaves. If $d < m-1$, suppose $\mathbf{a}'^{(j)}$ and $\mathbf{b}'^{(j)}$ are already extracted, we now show how to further extract witnesses for the depth-d vertex. For each child vertex, let $\mathbf{a}'^{(j)}_s = \mathbf{a}'^{(j)}_{[1:|S|]}$, $\mathbf{b}'^{(j)}_s = \mathbf{b}'^{(j)}_{[1:|S|]}$, $\mathbf{a}'^{(j)}_t = \mathbf{a}'^{(j)}_{[|S|+1:\dim(\mathbf{a}'^{(j)})]}$ and $\mathbf{b}'^{(j)}_t = \mathbf{b}'^{(j)}_{[|S|+1:\dim(\mathbf{b}'^{(j)})]}$ we have

$$\begin{aligned}
&L^{x_{(j)}^2}PR^{x_{(j)}^{-2}}\\
=&u^{\langle\mathbf{a}'^{(j)},\mathbf{b}'^{(j)}\rangle}\mathbf{g}_{[S]}^{\mathbf{a}'^{(j)}_s}\left(\mathbf{g}_{t[l]}^{x_{(j)}^{-1}}\circ\mathbf{g}_{t[r]}^{x_{(j)}}\right)^{\mathbf{a}'^{(j)}_t} \quad (j\in(1:4)). \quad (4)\\
&\mathbf{h}_{[S]}^{\mathbf{b}'^{(j)}_s}\left(\mathbf{h}_{t[l]}^{x_{(j)}}\circ\mathbf{h}_{t[r]}^{x_{(j)}^{-1}}\right)^{\mathbf{b}'^{(j)}_t}
\end{aligned}$$

Consider a system of linear equations:

$$\begin{pmatrix} x_{(1)}^2 & x_{(2)}^2 & x_{(3)}^2 \\ 1 & 1 & 1 \\ x_{(1)}^{-2} & x_{(2)}^{-2} & x_{(3)}^{-2} \end{pmatrix}\begin{pmatrix}\theta_1\\ \theta_2\\ \theta_3\end{pmatrix} = \begin{pmatrix}1\\0\\0\end{pmatrix}. \quad (5)$$

Because x_1, x_2, x_3 are distinct challenges and the left hand side of (5) is equivalent to a Vandermonde matrix, we can obtain θ_1, θ_2, θ_3. With coefficients $\theta_1, \theta_2, \theta_3$, the linear combination of (4) gives $L = \mathbf{g}^{\mathbf{a}_L}\mathbf{h}^{\mathbf{b}_L}u^{c_L}$, where $\mathbf{a}_L$, $\mathbf{b}_L$, c_L can be calculated from $\mathbf{a}'^{(j)}, \mathbf{b}'^{(j)}, j = 1, 2, 3$. Similarly, we can calculate $R = \mathbf{g}^{\mathbf{a}_R}\mathbf{h}^{\mathbf{b}_R}u^{c_R}$, $P = \mathbf{g}^{\mathbf{a}_P}\mathbf{h}^{\mathbf{b}_P}u^{c_P}$. By substituting L, P, R in (4), for $j \in (1:4)$, we have

$$\begin{aligned}
&\mathbf{g}^{x_{(j)}^2\mathbf{a}_L+\mathbf{a}_P+x_{(j)}^{-2}\mathbf{a}_R}\mathbf{h}^{x_{(j)}^2\mathbf{b}_L+\mathbf{b}_P+x_{(j)}^{-2}\mathbf{b}_R}u^{x_{(j)}^2c_L+c_P+x_{(j)}^{-2}c_R}\\
=&\mathbf{g}_{[S]}^{\mathbf{a}'^{(j)}_s}\mathbf{g}_{t[l]}^{x_{(j)}^{-1}\mathbf{a}'^{(j)}_t}\mathbf{g}_{t[r]}^{x_{(j)}\mathbf{a}'^{(j)}_t}\mathbf{h}_{[S]}^{\mathbf{b}'^{(j)}_s}\mathbf{h}_{t[l]}^{x_{(j)}\mathbf{a}'^{(j)}_t}\mathbf{h}_{t[r]}^{x_{(j)}^{-1}\mathbf{b}'^{(j)}_t}\\
&u^{\langle\mathbf{a}'^{(j)},\mathbf{b}'^{(j)}\rangle}.
\end{aligned}$$

Because of the discrete logarithm relation assumption (Definition 1) , with overwhelming probability,

$$\begin{aligned}
x_{(j)}^{-1}\mathbf{a}'^{(j)}_t &= x_{(j)}^2\mathbf{a}_{L,[T][l]} + \mathbf{a}_{P,[T][l]} + x_{(j)}^{-2}\mathbf{a}_{R,[T][l]},\\
x_{(j)}\mathbf{a}'^{(j)}_t &= x_{(j)}^2\mathbf{a}_{L,[T][r]} + \mathbf{a}_{P,[T][r]} + x_{(j)}^{-2}\mathbf{a}_{R,[T][r]},\\
x_{(j)}\mathbf{b}'^{(j)}_t &= x_{(j)}^2\mathbf{b}_{L,[T][l]} + \mathbf{b}_{P,[T][l]} + x_{(j)}^{-2}\mathbf{b}_{R,[T][l]},\\
x_{(j)}^{-1}\mathbf{b}'^{(j)}_t &= x_{(j)}^2\mathbf{b}_{L,[T][r]} + \mathbf{b}_{P,[T][r]} + x_{(j)}^{-2}\mathbf{b}_{R,[T][r]},\\
\langle\mathbf{a}'^{(j)},\mathbf{b}'^{(j)}\rangle &= x_{(j)}^2c_L + c_P + x_{(j)}^{-2}c_R.
\end{aligned} \quad (6)$$

where $j \in (1:4)$. From (6), we deduce that

$$\begin{aligned}
&x^3\mathbf{a}_{L,[T][l]} + x(\mathbf{a}_{P,[T][l]} - \mathbf{a}_{L,[T][r]})\\
&+x^{-1}(\mathbf{a}_{R,[T][l]} - \mathbf{a}_{P,[T][r]}) - x^{-3}\mathbf{a}_{R,[T][r]} = 0,
\end{aligned} \quad (7)$$

$$\begin{aligned}
&x^3\mathbf{b}_{L,[T][r]} + x(\mathbf{b}_{P,[T][r]} - \mathbf{b}_{L,[T][l]})\\
&+x^{-1}(\mathbf{b}_{R,[T][r]} - \mathbf{b}_{P,[T][l]}) - x^{-3}\mathbf{b}_{R,[T][l]} = 0.
\end{aligned} \quad (8)$$

The challenges, $x_{(j)}, j \in (1:4)$, are distinct solutions of (7). This implies that the polynomials on the left side of (7) are zero polynomials. Similarly, for (8). Therefore,

$$\begin{aligned}
&\mathbf{a}_{P,[T][l]} = \mathbf{a}_{L,[T][r]}, \quad \mathbf{a}_{P,[T][r]} = \mathbf{a}_{R,[T][l]},\\
&\mathbf{b}_{P,[T][r]} = \mathbf{b}_{L,[T][l]}, \quad \mathbf{b}_{P,[T][l]} = \mathbf{b}_{R,[T][r]},\\
&\mathbf{a}_{L,[T][l]} = \mathbf{a}_{R,[T][r]} = \mathbf{b}_{L,[T][r]} = \mathbf{b}_{R,[T][l]} = \mathbf{0}.
\end{aligned}$$

Hence, for $j \in (1:4)$,

$$\begin{aligned}
\mathbf{a}'^{(j)}_t &= x_{(j)}\mathbf{a}_{P,[T][l]} + x_{(j)}^{-1}\mathbf{a}_{P,[T][r]},\\
\mathbf{b}'^{(j)}_t &= x_{(j)}\mathbf{b}_{P,[T][r]} + x_{(j)}^{-1}\mathbf{b}_{P,[T][l]}.
\end{aligned}$$

Now we have

$$\begin{aligned}
\langle\mathbf{a}'^{(j)},\mathbf{b}'^{(j)}\rangle =&\langle\mathbf{a}'^{(j)}_s,\mathbf{b}'^{(j)}_s\rangle + \langle\mathbf{a}'^{(j)}_t,\mathbf{b}'^{(j)}_t\rangle\\
=&\langle\mathbf{a}'^{(j)}_s,\mathbf{b}'^{(j)}_s\rangle + x_{(j)}^2\langle\mathbf{a}_{P,[T][l]},\mathbf{b}_{P,[T][r]}\rangle\\
&+\langle\mathbf{a}_{P,[T]},\mathbf{b}_{P,[T]}\rangle + x_{(j)}^{-2}\langle\mathbf{a}_{P,[T][r]},\mathbf{b}_{P,[T][l]}\rangle\\
=&x_{(j)}^2c_L + c_P + x_{(j)}^{-2}c_R,
\end{aligned}$$

where $j \in (1:4)$. Since $x_{(j)}$ is distinct, we conclude that

$$\langle\mathbf{a}'^{(j)}_s \parallel \mathbf{a}_{P,[T]}, \mathbf{b}'^{(j)}_s \parallel \mathbf{b}_{P,[T]}\rangle = c_P, \quad (j\in(1:4)).$$

Since the padding steps before the compression introduce extra padding zeros, by removing them from $\mathbf{a}'^{(1)}_s \parallel \mathbf{a}_{P,[T]}, \mathbf{b}'^{(1)}_s \parallel \mathbf{b}_{P,[T]}$, we obtain the "intermediate witnesses" of the depth-d vertex.

At the end, by repeatedly extracting witnesses for every vertex from the bottom of the tree to the top, the

extracted witnesses of the root is a valid witnesses for relation (1). Hence, we have an extractor of witnesses for $(k_1, k_2, \ldots, k_m)$-transcript-tree of $\mathsf{SIPA}(f)$. The extraction process above takes $O(n)$ to finish, and the whole extractor repeats the process for $\sum_{d=1}^{m-1} 4^{i-1} \leq 4^m$ times. Overall, the extractor runs in $O(n4^m) = O(n4^{\log n}) = O(n^3)$ time and is efficient. Therefore, $\mathsf{SIPA}(f)$ has $(4, \ldots, 4)$-special-soundness.

Similar to the discussion of Bünz et al. [13, Lemma 1], by Lemma 1, we have the knowledge error $\kappa \leq \epsilon(\lambda) - (\epsilon(\lambda) - \kappa_0)^K$. Since $K = 4^m = O(n^2)$ is a polynomial of λ and the size of challenge set is $p = O(2^\lambda)$, the knowledge error is negligible. Therefore, $\mathsf{SIPA}(f)$ is knowledge-sound. For the Fiat-Shamir transformation of $\mathsf{SIPA}(f)$, the number of queries to random oracle is up-bounded by m. By Lemma 2, the non-interactive $\mathsf{SIPA}(f)$ has knowledge-soundness as well. □

3.3. Optimal Scheme Functions

In this section, we discuss the theoretical lower bound of computational cost and proof size of Springproofs, and the existence of an optimal argument reaching the lower bound. First, we consider arguments consisting of compression steps only, then discuss IPAs with both compression and padding steps.

In order to analyze the lower bound of the cost of the whole IPA, it is necessary to first consider Algorithm 1, the core of a single compression step. The number of unit computational operations of Algorithm 1 is listed in Table 1. In terms of proof size of Algorithm 1, the prover sends n elements over $\mathbb{F}_p$ and 2 elements over $\mathbb{G}$ in total. Afterwards, the n elements over $\mathbb{F}_p$ will be further compressed in the next round of Springproofs, and reduced to 2 elements over $\mathbb{F}_p$ eventually. Therefore, the proof size contribution of a single compression step of Springproofs is 2 elements over $\mathbb{G}$.

TABLE 1: Computational Complexity of Operations of Algorithm 1

Prover	Verifier	Unit Operation
$2n-2$	0	Field Addition
$3n$	2	Field Multiplication
1	1	Field Inversion
$3n$	$n+2$	Group Multiplication
$4n+2$	$2n+2$	Group Exponentiation

From Table 1 and the analysis above, the computational cost and the contribution of proof size are both linear dependent with the compression length $2 \leq |T| \leq n$ in a single compression step, i.e. Algorithm 2.

More specifically, let $i = |T|/2$, $\alpha_c i + \beta_c, (\alpha_c \geq 0, \alpha_c + \beta_c \geq 0)$ be the computational cost of a single compression step, and $\alpha_p i + \beta_p (\alpha_p \geq 0, \alpha_p + \beta_p > 0)$ be the contribution of proof size of a single compression step. The α_c, β_c, α_p and β_p here are factors in the linear functions of the computational costs and the contribution of proof size. For instance, if we use the basic IPA of Bulletproofs for compression steps, the proof size contribution of a single compression step is 2 elements on $\mathbb{G}$. Hence, α_p is 0, and β_p is the total size of 2 elements on $\mathbb{G}$. Similarly, from Table 1, the values of α_c and β_c for the prover and the verifier are calculated as well.

For a $\mathsf{SIPA}(f)$ consisting only of compression steps, let $\mathrm{cost}_{\alpha_c,\beta_c}(n)$ be the optimal overall computational cost, and $\mathrm{cost}_{\alpha_p,\beta_p}(n)$ be the optimal proof size. Since $\alpha_c i + \beta_c$, and $\alpha_p i + \beta_p$ are all linear functions, we unify them into a single function $\mathrm{cost}_{\alpha,\beta}(n)$. Then, $\mathrm{cost}_{\alpha,\beta}(n)$ is consistent with the following recursive equation

$$\begin{aligned} &\mathrm{cost}_{\alpha,\beta}(1) = 0, \\ &\mathrm{cost}_{\alpha,\beta}(n) = \min_{i=1}^{\lfloor n/2 \rfloor} \{\mathrm{cost}_{\alpha,\beta}(n-i) + \alpha i + \beta\}, \end{aligned} \tag{9}$$

where the solution is as follows: (10):

$$\mathrm{cost}_{\alpha,\beta}(n) = \begin{cases} \alpha(n-1) + \beta\lceil \log n \rceil, & \beta \geq 0; \\ (\alpha+\beta)(n-1), & -\alpha < \beta < 0. \end{cases} \tag{10}$$

Now we will prove (10) by mathematical induction. First, it is straightforward that $\mathrm{cost}_{\alpha,\beta}(1) = 0$ in (10).

Assuming (10) holds for $n-1$,

(i) if $\beta \geq 0$, we have

$$\begin{aligned} &\mathrm{cost}_{\alpha,\beta}(n) \\ =& \min_{i=1}^{\lfloor n/2 \rfloor} \{\alpha(n-1) + \beta\lceil \log(n-i) + 1 \rceil\} \\ =& \alpha(n-1) + \beta\lceil \log(n - \lfloor n/2 \rfloor) \rceil + \beta \\ =& \alpha(n-1) + \beta\lceil \log n \rceil; \end{aligned} \tag{11}$$

(ii) if $-\alpha < \beta < 0$, we have

$$\begin{aligned} \mathrm{cost}_{\alpha,\beta}(n) &= \min_{i=1}^{\lfloor n/2 \rfloor} \{(\alpha+\beta)(n-i-1) + \alpha i + \beta\} \\ &= \min_{i=1}^{\lfloor n/2 \rfloor} \{\alpha(n-1) + \beta(n-i)\} \\ &= (\alpha+\beta)(n-1). \end{aligned}$$

Therefore, $\mathrm{cost}_{\alpha,\beta}(n)$ is solved by (10). From Table 1 we know that in a single compression step, $\alpha_c > 0$, $\beta_c > 0$ holds for the computational cost, and $\alpha_p = 0, \beta_p > 0$ holds for the proof size contribution. So the lower bound of computational cost and proof size are both consistent with (11).

From (11), we conclude that the optimal cost is achieved by choosing compression length $|T|$ of Algorithm 2 greedily in each step. Therefore, the existence of optimal argument is guaranteed.

Now we consider Springproofs incorporating padding and compression steps. After the last padding step of the argument, the rest steps are all compression steps. Instead of padding before compression steps, we compress directly with an optimal argument within all arguments only consisting of compression steps. Because $\mathrm{cost}_{\alpha,\beta}(n)$ increases monotonically, the computational and communication cost of the new argument do not increase compared with the

original argument, but the last padding step is removed in the new argument. Eventually, by applying this trick repeatedly, all padding steps are removed without increasing cost of proof size or computational cost. Therefore, Springproofs without padding steps are more efficient than that with padding, as the following theorem demonstrates.

Theorem 4. For any scheme function $f : \mathbb{Z}^+ \to \mathbb{Z}^+ \sqcup \mathscr{P}(\mathbb{Z}^+)$ satisfying Theorem 3, if $\mathrm{cost}_{\alpha,\beta}(n)$ increases monotonically, there exists a scheme function $g : \mathbb{Z}^+ \to \mathscr{P}(\mathbb{Z}^+)$ satisfying Theorem 3, such that the computational cost and proof size of $\mathsf{SIPA}(g)$ is not greater than $\mathsf{SIPA}(f)$ for any n.

Proof: Given a scheme function f satisfying Theorem 3, let m be the number of recursive steps involved in $\mathsf{SIPA}(f)$. We define the length function $\ell_f : \mathbb{Z}^+ \to \mathbb{Z}^+$ that measures the length of $\mathbf{a}$ and $\mathbf{b}$ after the first step of $\mathsf{SIPA}(f)$,

$$\ell_f(n) = \begin{cases} n + f(n), & f(n) \in \mathbb{Z}^+; \\ n - |f(n)|/2, & f(n) \in \mathscr{P}(\mathbb{Z}^+). \end{cases}$$

$$\ell_f^k(n) = \begin{cases} n, & k = 0; \\ \ell_f(\ell_f^{k-1}(n)), & 0 < k \le m, \end{cases} \quad (k \in [0, m]). \tag{12}$$

Similarly, from (12) we derive the length of $\mathbf{a}$ and $\mathbf{b}$ after the first k recursive steps in $\mathsf{SIPA}(f)$.

Suppose $\mathsf{SIPA}(g_0)$ is an optimal IPA within all $\mathsf{SIPA}(g)$ where g and g_0 are $\mathbb{Z}^+ \to \mathscr{P}(\mathbb{Z}^+)$. For any given $f : \mathbb{Z}^+ \to \mathbb{Z}^+ \cup \mathscr{P}(\mathbb{Z}^+)$ and $n \in \mathbb{Z}^+$, we have a vector length sequence $\{\ell_f^k(n)\}_{k=0}^m$. Suppose k_0 is the maximum k that $f(\ell_f^k(n)) \in \mathbb{Z}^+$. Then step k_0+1 is a padding step, and step k_0+2 to step m are all compression steps. Since $\mathrm{cost}_{\alpha,\beta}(n)$ increases monotonically, by replacing the scheme function f by g_0 after recursive step k_0, the padding step $k_0 + 1$ is removed without an increment of computational cost or proof size.

We keep replacing the scheme function until the whole argument transforms from $\mathsf{SIPA}(f)$ to $\mathsf{SIPA}(g_0)$. Since neither computational cost nor proof size increases in each transformation of the argument, we conclude that the computational cost and proof size of $\mathsf{SIPA}(g_0)$ are not greater than that of $\mathsf{SIPA}(f)$ for any $n \in \mathbb{Z}^+$. □

3.4. Practical Instantiations of Springproofs

This section describes some practical instantiations of Springproofs. For comparison, we first show a straightforward instantiation, padding method; then give two optimal arguments under Springproofs. Figure 1 illustrates the recursive proof structures of these three methods when the input vector length $n = 11$.

3.4.1. Padding Method. The most straightforward solution to eliminate the vector length limitation is padding the witnesses $\mathbf{a}, \mathbf{b}$ to a $2^{\lceil \log n \rceil}$-dimensional vector with 0, as we discussed earlier. Padding method is instantiated with the scheme function

$$f_{\mathrm{p}}(n) = \begin{cases} 2^{\lceil \log n \rceil} - n, & n \neq 2^{\lceil \log n \rceil}; \\ (1 : n), & n = 2^{\lceil \log n \rceil}. \end{cases}$$

It is clear that after padding, only $\lceil \log n \rceil$ steps of compression is required, which is bounded by $2 \log n$. The computational cost is

$$\alpha_{\mathrm{c}}(2^{\lceil \log n \rceil} - 1) + \beta_{\mathrm{c}} \lceil \log n \rceil.$$

When n is a power of 2, the computational cost reaches the theoretical optimum (11). However, such an optimal performance does not hold when n is not a power of 2. If $n = 2^k + 1, k \in \mathbb{Z}^+$, the computational cost is nearly 2 times of theoretical optimum. Therefore, padding method is not an ideal method in terms of performance.

As for proof size, in total, $\lceil \log n \rceil$ steps of compression require sending $2\lceil \log n \rceil$ elements over $\mathbb{G}$ which is the theoretical optimum.

3.4.2. Greedy Method. The scheme function of greedy method is

$$f_{\mathrm{g}}(n) = \begin{cases} (1 : n), & n \text{ is even}; \\ (2 : n), & n \text{ is odd}. \end{cases}$$

As discussed in Section 3.3, greedy method is an optimal method. The computational cost and proof size are both characterized by (11).

With mathematical induction, it is straightforward to prove that the greedy method takes $\lceil \log n \rceil$ of compression steps in total. Therefore, the method is an argument of knowledge of relation (1) by Theorem 3.

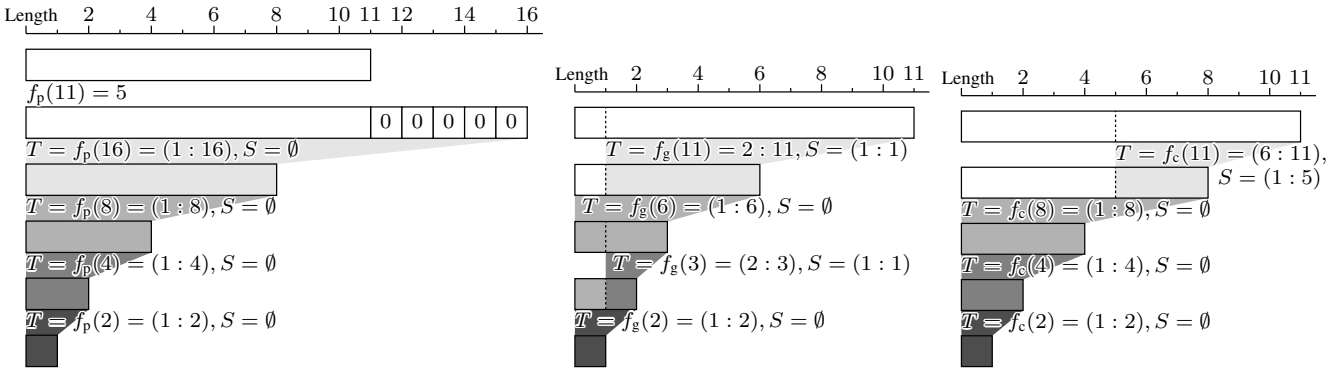
3.4.3. Pre-Compression Method. Another approach to instantiate an IPA is to compress $\mathbf{a}, \mathbf{b}$ to a smaller length that equals to a power of 2, instead of padding them to a greater length. Since the method is basically compressing the vectors once before the original Bulletproofs when n is not a power of 2, we call it the pre-compression method. The scheme function is

$$f_{\mathrm{c}}(n) = \begin{cases} (2^{\lceil \log n \rceil} - n + 1 : n), & n \neq 2^{\lceil \log n \rceil}; \\ (1 : n), & n = 2^{\lceil \log n \rceil}. \end{cases}$$

Because the IPA only adds a single compression step before the original Bulletproofs, the pre-compression method also takes $\lceil \log n \rceil$ compression steps in total. And the proof size is $2\lceil \log n \rceil$ elements over $\mathbb{G}$ and 2 elements over $\mathbb{F}_p$ as well. By Theorem 3, pre-compression method is a proof of knowledge for relation (1).

The computational cost of pre-compression method is consistent with (11):

$$\begin{aligned} &\alpha_{\mathrm{c}}(2^{\lceil \log n \rceil - 1} - 1) + \beta_{\mathrm{c}} \lceil \log(2^{\lceil \log n \rceil - 1}) \rceil \\ &\quad + \alpha_{\mathrm{c}}(n - 2^{\lceil \log n \rceil - 1}) + \beta_{\mathrm{c}} \\ =&\alpha_{\mathrm{c}}(n-1) + \beta_{\mathrm{c}} \lceil \log n \rceil. \end{aligned}$$



(a) Padding method (b) Greedy method (c) Pre-Compression method

Figure 1: Recursive proof structure of 3 different instantiations of Springproofs when the length of the witness vector is 11

Hence, the performance of pre-compression method reaches the theoretical optimum (11).

Therefore, we conclude that the pre-compression method is also an optimal argument. Although greedy method and pre-compression method are both optimal arguments, the recursive structure of pre-compression method is more concise than that of greedy method. Therefore, it is easier to build an optimized verification algorithm for pre-compression method, as we will see in Section 3.5.

3.5. Non-interactive Springproofs and Optimized Verification

Since every message from the verifier to the prover is sampled uniformly from $\mathbb{F}_p$, Bulletproofs based Springproofs are public-coin IPAs. Therefore, by Fiat-Shamir heuristic [10], we convert Springproofs into non-interactive IPAs in the random oracle model. In non-interactive Springproofs, the proof generated by the prover consists of $a \in \mathbb{F}_p, b \in \mathbb{F}_p, L_1, L_2, \ldots, L_m \in \mathbb{G}$ and $R_1, R_2, \ldots, R_m \in \mathbb{G}$, where m is the total number of compression steps.

Moreover, because Springproofs is public coin IPAs, the verifier can defer most of the computation to the end, and verify the proof with a single *multi-exponentiation*. Because a single multi-exponentiation is much faster than separate group multiplication, such an optimized verification is more efficient than the original verification.

Specifically in the multi-exponentiation based verification, the verifier skips the computation of $\mathbf{g}', \mathbf{h}', \mathbf{a}', \mathbf{b}'$ in steps 5, 6 and 7 of Algorithm 1, and calculates

$$(\mathbf{g} \parallel \mathbf{g}_{\mathrm{p}})^{a\mathbf{v}} (\mathbf{h} \parallel \mathbf{h}_{\mathrm{p}})^{b\mathbf{v}^{-1}} u^{ab} = P \prod_{j=1}^{m} L_j^{x_j^2} R_j^{x_j^{-2}} \tag{13}$$

in step 20 of Algorithm 3. If (13) holds in step 20 of Algorithm 3, the verifier accepts the proof. Otherwise, the verifier rejects the proof. In (13), $\mathbf{v}$ is a vector over $\mathbb{F}_p$ that only depends on the challenges $x_1, x_2, \ldots, x_m$, where x_j is the challenge sent from the verifier in the jth compression step of Springproofs. The dimension of $\mathbf{v}$ is the sum of the length of the witness vector and the total number of padding zeros in $\mathsf{SIPA}(f)$. For instance, we have $\dim(\mathbf{v}) = 2^{\lceil \log n \rceil}$ in padding method, and $\dim(\mathbf{v}) = n$ in greedy and pre-compression method.

The determination of $\mathbf{v}$ could be a time-consuming part in (13), but could be more efficient once the related scheme function derives a nice intrinsic structure for $\mathbf{v}$. The structure of $\mathbf{v}$ heavily depends on the recursive structure of $\mathsf{SIPA}(f)$, which is controlled by the scheme function f. In padding method, $\mathbf{v}$ is basically consistent with Bulletproofs. For $i = 1, \ldots, 2^{\lceil \log n \rceil}$ we have $v_i = \prod_{j=1}^{m} x_j^{b(i,j)}$, where $b(i,j) = 1$ if the $(m+1-j)$th bit of $i-1$ is 1, and $b(i,j) = -1$ otherwise [14]. For efficiently calculating $\mathbf{v}$, Bünz et al. [14] pointed out that $v_i = v_{i-k} x_{k+1}^2$, where $k = 2^{\lfloor \log(i-1) \rfloor}$. Furthermore, since $\dim(\mathbf{v})$ is a power of 2, the jth component of $\mathbf{v}^{-1}$ is exactly $v_{2^{\lceil \log n \rceil}+1-j}$. Therefore, the structures of $\mathbf{v}$ and $\mathbf{v}^{-1}$ are reciprocally symmetric. By one batch field inversion and $2^{\lceil \log n \rceil} - 1$ field multiplication, we calculate $\mathbf{v}$ and $\mathbf{v}^{-1}$ from the challenges.

Normally such a symmetric structure does not exist for most $\mathsf{SIPA}(f)$, including greedy method. However, with a little modification to f_{c}, we create an optimal argument of Springproofs with an efficient calculation procedure of $\mathbf{v}$. Let

$$f'_{\mathrm{c}}(n) := \begin{cases} (i_h : i_t) \cup (N+1 : n), & n \neq N; \\ (1:n), & n = N, \end{cases}$$

where $N = 2^{\lceil \log n \rceil - 1}, i_h = \lfloor (2N-n)/2 \rfloor + 1, i_t = \lfloor n/2 \rfloor$. Then $\mathsf{SIPA}(f'_{\mathrm{c}})$ is a variant of pre-compression method whose left parts of $\mathbf{a}_{[T]}$ and $\mathbf{b}_{[T]}$ are chosen from the middle of $\mathbf{a}_{[1:N]}$ and $\mathbf{b}_{[1:N]}$ in the first round of recursive proof. Figure 2 illustrates the recursive proof structures of $\mathsf{SIPA}(f'_{\mathrm{c}})$ when the input vector length $n = 11$.

Since the left parts of vectors are chosen from the middle, the $\mathbf{v}$ of $\mathsf{SIPA}(f'_{\mathrm{c}})$ has a symmetric structure similar to Bulletproofs. In $\mathsf{SIPA}(f'_{\mathrm{c}})$, we have

$$v_i = \begin{cases} v_{i-k} x_{k+1}^2 x_1^{-1}, & i_h \leq i \leq i_t \wedge i - k < i_h; \\ v_{i-k} x_{k+1}^2 x_1, & i_t < i \leq N \wedge i_h \leq i - k \leq i_t; \\ v_{i-N-1+i_h} x_1^2, & i > N; \\ v_{i-k} x_{k+1}^2, & \text{otherwise}, \end{cases}$$

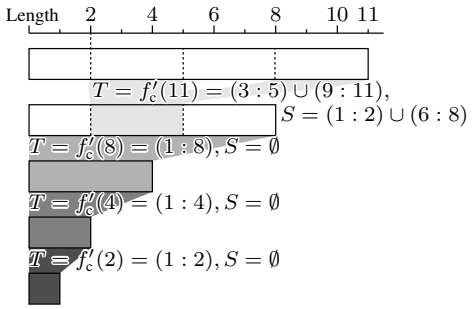


Figure 2: Recursive proof structure of $\mathsf{SIPA}(f'_c)$ when the length of the witness vector n is 11

where $k = 2^{\lfloor \log(i-1) \rfloor}$, and $v_1 = (x_1 x_2 \dots x_m)^{-1}$ when $i_h = 1$, $v_1 = (x_2 x_3 \dots x_m)^{-1}$ otherwise. Moreover, because of the symmetric structure of $\mathbf{v}$, $\mathbf{v}^{-1}$ can be calculated from $\mathbf{v}$ efficiently. If n is even, we rearrange $\mathbf{v}$ directly to get $\mathbf{v}^{-1}$. If n is odd, we have $v_{i_h}^{-1} = v_{i_t+1} x_1, v_{i_t+1}^{-1} = v_{i_h} \cdot x_1, v_{N+1}^{-1} = v_{i_t+1} x_1^{-1}$. For each of rest components of $\mathbf{v}$, we find its inverse as another component in $\mathbf{v}$. Overall, we calculate $\mathbf{v}$ and $\mathbf{v}^{-1}$ from challenges in $\mathsf{SIPA}(f'_c)$ with one batch field inversion and less than $2n$ field multiplications.

3.6. Springproofs Based on Other IPAs

For clarity, we only consider Springproofs based on Bulletproofs previously. However, there are currently IPAs that achieve lower computational cost or communication cost, or have broader applications than Bulletproofs, such as Bulletproofs+ [7], Shellproof [6], the IPA of Gentry et al. [15], the generalized IPAs of Lai et al. [16], and the generalized IPAs of Bünz et al. [13].

For the IPAs listed above, it is possible to instantiate Springproofs on top of them. Due to space constraints, we do not look into the details of Springproofs based on each IPA. In general, because Springproofs directly invoke the basic IPA in the arguments, extractors of Springproofs are constructed without much modification to the extractors of the original IPAs. Therefore, the completeness and knowledge soundness could be analyzed similarly as in Theorem 3.

Section 3.4 describes a number of optimal scheme functions of Springproofs based on Bulletproofs. They may not be optimal for an IPA different from Bulletproofs. However, as long as the computational and communication cost in a single compression step is linear dependent with the compression size $|T|$, we can use $\mathrm{cost}_{\alpha,\beta}(n)$ in (10) to describe the overall cost of $\mathsf{SIPA}(f)$, and check whether f_g and f_c yield optimal arguments or not.

It is worth noting that if Springproofs are instantiated with a non-zero-knowledge IPA, such as the IPA of Bulletproofs, Springproofs are not zero-knowledge. However, if the Springproofs are based on a zero-knowledge IPA, then Springproofs preserve the zero-knowledge property of the original IPA. We discuss the zero-knowledge property of Springproofs in detail in Section 3.7.

3.7. Zero-knowledge Springproofs

In this section, we analyze Springproof and prove that it preserves the zero-knowledge property of the basic IPA.

First of all, the original inner product relation of Bulletproof (1) is not zero-knowledge. Since the commitment P in (1) contains the information of the witnesses $\mathbf{a}$ and $\mathbf{b}$. For analyzing the zero-knowledge property of Springproofs, we instead consider a variant of (1):

$$\mathcal{R}_{\mathrm{zk},n} = \{(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, P \in \mathbb{G}; \mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n, \gamma \in \mathbb{F}_p) : P = \mathbf{g}^{\mathbf{a}} \mathbf{h}^{\mathbf{b}} u^{\langle \mathbf{a}, \mathbf{b} \rangle} u'^{\gamma}\}.$$

In $\mathcal{R}_{\mathrm{zk},n}$, by introducing a blinding γ, the commitment becomes a *Pedersen commitment* [17]. Because a Pedersen commitment is indistinguishable from a random element on $\mathbb{G}$, i.e., perfectly hiding, the commitment does not leak information of the witnesses. Some IPAs, including the IPA of Gentry et al. [15], are zero-knowledge IPAs for some relations similar to $\mathcal{R}_{\mathrm{zk},n}$.

We omit the detail of specific IPAs, only assume that IPA_k, the IPA instantiating the Springproof, satisfies the following properties:

(i) IPA_k reduces the relation $\mathcal{R}_{\mathrm{zk},k}$ into relation $\mathcal{R}_{\mathrm{zk},k/2}$, in which the blinding factors of the two relations distribute independently.

(ii) IPA_k is HVZK, i.e., there exists an efficient simulator that simulates transcripts of IPA_k, whose statistical difference $\epsilon(\lambda)$ from the real transcripts is negligible.

With IPA_k, we now construct the basic IPA of zero-knowledge Springproof.

Algorithm 4 Basic zero-knowledge IPA of Springproofs instantiated with $S \sqcup T = (1:n)$ and $\mathsf{IPA}_{|T|}$

Input: $(\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, u, u', P \in \mathbb{G}; \mathbf{a}, \mathbf{b} \in \mathbb{F}_p^n)$
Prover's input: $(\mathbf{g}, \mathbf{h}, u, u', P; \mathbf{a}, \mathbf{b})$
Verifier's input: $(\mathbf{g}, \mathbf{h}, u, u', P)$
Output: Verifier accepts or rejects

1: Prover $\to$ Verifier: $\mathbf{a}_{[S]}, \mathbf{b}_{[S]}$
2: Prover and Verifier compute:
$P_S \leftarrow \mathbf{g}_{[S]}^{\mathbf{a}_{[S]}} \mathbf{h}_{[S]}^{\mathbf{b}_{[S]}} u^{\langle \mathbf{a}_{[S]}, \mathbf{b}_{[S]} \rangle}, \quad Q \leftarrow P P_S^{-1}$.
3: Run $\mathsf{IPA}_{|T|}\left(\mathbf{g}_{[T]}, \mathbf{h}_{[T]}, u, u', Q; \mathbf{a}_{[T]}, \mathbf{b}_{[T]}, \gamma\right)$

In Algorithm 4, notice that by running $\mathsf{IPA}_{|T|}$, a reduced relation,

$$\mathcal{R}_{\mathrm{zk},|T|/2} = \{(\mathbf{g}', \mathbf{h}' \in \mathbb{G}^{|T|/2}, u, P' \in \mathbb{G}; \mathbf{a}', \mathbf{b}' \in \mathbb{F}_p^{|T|/2}, \gamma' \in \mathbb{F}_p) : P' = \mathbf{g}'^{\mathbf{a}'} \mathbf{h}'^{\mathbf{b}'} u^{\langle \mathbf{a}', \mathbf{b}' \rangle} u'^{\gamma'}\},$$

is obtained. We further construct

$$\left(\mathbf{g}_{[S]} \parallel \mathbf{g}', \mathbf{h}_{[S]} \parallel \mathbf{h}', u, u', P' P_S; \mathbf{a}_{[S]} \parallel \mathbf{a}', \mathbf{b}_{[S]} \parallel \mathbf{b}', \gamma'\right), \quad (14)$$

which is a fresh input of Algorithm 4. Therefore, by the recursive technique, a Springproof similar to Algorithm 3 is constructed. Denote by $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ the Springproof instantiated with a scheme function f and an inner product argument IPA. Similar to Theorem 3, the completeness and the knowledge soundness of $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ are analyzed with

TABLE 2: Comparison of Computational Complexity of Operations between Bulletproofs and $\mathsf{SIPA}(f'_{\mathrm{c}})$

Bulletproofs		$\mathsf{SIPA}(f'_{\mathrm{c}})$		**Unit Operation**
Prover	**Verifier**	**Prover**	**Verifier**	
$2 \cdot 2^{\lceil \log n \rceil} - 2\lceil \log n \rceil - 2$	0	$2n - 2\lceil \log n \rceil - 2$	0	Field Addition
$3 \cdot 2^{\lceil \log n \rceil} - 3$	$2\lceil \log n \rceil + 1$	$3n - 3$	$2\lceil \log n \rceil + 1$	Field Multiplication
$\lceil \log n \rceil$	$\lceil \log n \rceil$	$\lceil \log n \rceil$	$\lceil \log n \rceil$	Field Inversion
$3 \cdot 2^{\lceil \log n \rceil} - 3$	$2^{\lceil \log n \rceil} + 2\lceil \log n \rceil + 1$	$3n - 3$	$n + 2\lceil \log n \rceil + 1$	**Group Multiplication**
$4 \cdot 2^{\lceil \log n \rceil} + 2\lceil \log n \rceil - 4$	$2 \cdot 2^{\lceil \log n \rceil} + 2\lceil \log n \rceil + 1$	$4n + 2\lceil \log n \rceil - 4$	$2n + 2\lceil \log n \rceil + 1$	**Group Exponentiation**

the relations of blinding factors within IPA. We omit the detailed proof here due to space limitations.

Notice that Algorithm 4 is zero-knowledge for components $\mathbf{a}_{[T]}$ and $\mathbf{b}_{[T]}$ but not for components $\mathbf{a}_{[S]}$ and $\mathbf{b}_{[S]}$. However, because $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ has multiple rounds and is terminative, the relation $\mathcal{R}_{\mathrm{zk},n}$ is reduced into $\mathcal{R}_{\mathrm{zk},1}$ as in Algorithm 3 ultimately. At the end, intuitively all components of the witnesses participate the argument IPA for at least once. Therefore, no information of the witnesses is leaked in $\mathsf{SIPA}_{\mathsf{IPA}}(f)$.

For the proof of zero-knowledge of $\mathsf{SIPA}_{\mathsf{IPA}}(f)$, we construct an efficient extractor of the argument. Suppose $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ has a total of m compression steps when proving the relation $\mathcal{R}_{\mathrm{zk},n}$. Then $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ runs Algorithm 4 (IPA) for m times, the only interaction between the prover and the verifier excluding IPA is $\mathbf{a}_{[S]}$ and $\mathbf{b}_{[S]}$ in Algorithm 4. Because $\mathbf{a}_{[S]}$ and $\mathbf{b}_{[S]}$ in a compression step will always participate in the IPA of the next compression step, all the interactions between the prover and the verifier are limited within the IPA of each compression step. Therefore, the whole transcript is simulated by running the simulator of IPA for m times, and confirming that the inputs to each simulator distributes indistinguishably from the inputs of IPA in a real $\mathsf{SIPA}_{\mathsf{IPA}}(f)$.

By randomly choosing an element Q' from $\mathbb{G}$ uniformly, we obtain PQ'^{-1} and $P'PQ'^{-1}$ which correspond to P_{S} and $P'P_{\mathrm{S}}$ respectively. Using $P'P_{\mathrm{S}}$ as the commitment of a new relation $\mathcal{R}_{\mathrm{zk},|T|/2+|S|}$, the simulator continues the simulation for the rest steps and simulates the whole transcript of $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ ultimately. Because Pedersen commitment is perfectly hiding, Q' has the same distribution as Q in Algorithm 4, and $P'PQ'^{-1}$ has the same distribution as $P'P_{\mathrm{S}}$ in (14). Therefore, the distribution of the whole transcript from the simulator of $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is indistinguishable from the real transcript. The following theorem shows that $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is HVZK. See the formal proof in Appendix B.

Theorem 5. Suppose IPA_k is a HVZK IPA which reduces a relation $\mathcal{R}_{\mathrm{zk},k}$ into a relation $\mathcal{R}_{\mathrm{zk},k/2}$, and the blinding factors in the two relations distribute independently. Given a scheme function f, if the $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is terminative for any lengths n of the witness vector, and there exists a polynomial $\mathrm{poly}(\lambda)$ such that the number of rounds $m < \mathrm{poly}(\lambda)$, then $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is HVZK when $n \geq 2$.

Since $n = O(\lambda)$, it is clear that $\mathsf{SIPA}_{\mathsf{IPA}}(f_{\mathrm{p}})$. $\mathsf{SIPA}_{\mathsf{IPA}}(f_{\mathrm{g}})$, $\mathsf{SIPA}_{\mathsf{IPA}}(f_{\mathrm{c}})$, and $\mathsf{SIPA}_{\mathsf{IPA}}(f'_{\mathrm{c}})$ run in only $\lceil \log n \rceil$ rounds of compression steps, and thus satisfy the requirement of Theorem 5.

In Theorem 5, n must be greater than 1. If $n = 1$, the prover sends the witnesses to the verifier directly in $\mathsf{SIPA}_{\mathsf{IPA}}(f)$. Correspondingly, $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is replaced by a zero-knowledge argument of a product relation on $\mathbb{F}$, such as the argument by Gentry et al. [15].

4. Performance Evaluations

In this section, we evaluate the performance of Springproofs based on $\mathsf{SIPA}(f'_{\mathrm{c}})$, with theoretical analysis and practical experiments in different aspects, including range proof, generation and verification of Monero transaction, and privacy computing of arithmetic circuits. We also compare the performance among $\mathsf{SIPA}(f'_{\mathrm{c}})$, Bulletproofs, and Groth16 by experimental results. All experiments operate on an Intel Xeon Silver 4210 CPU with a single thread.

In the experiments, the argument of arithmetic circuit satisfiability and the argument of range proof, including the aggregation and batch verification algorithm, are identical to the arguments by Bünz et al. [3] except their IPAs. The IPAs are chosen between the original IPA of Bulletproofs and $\mathsf{SIPA}(f'_{\mathrm{c}})$ for evaluation. The argument of range proof and arithmetic circuits satisfiability achieve zero-knowledge even without zero-knowledge IPAs. Therefore, all the arguments in the experiments run with zero-knowledge setups for a fair comparison.

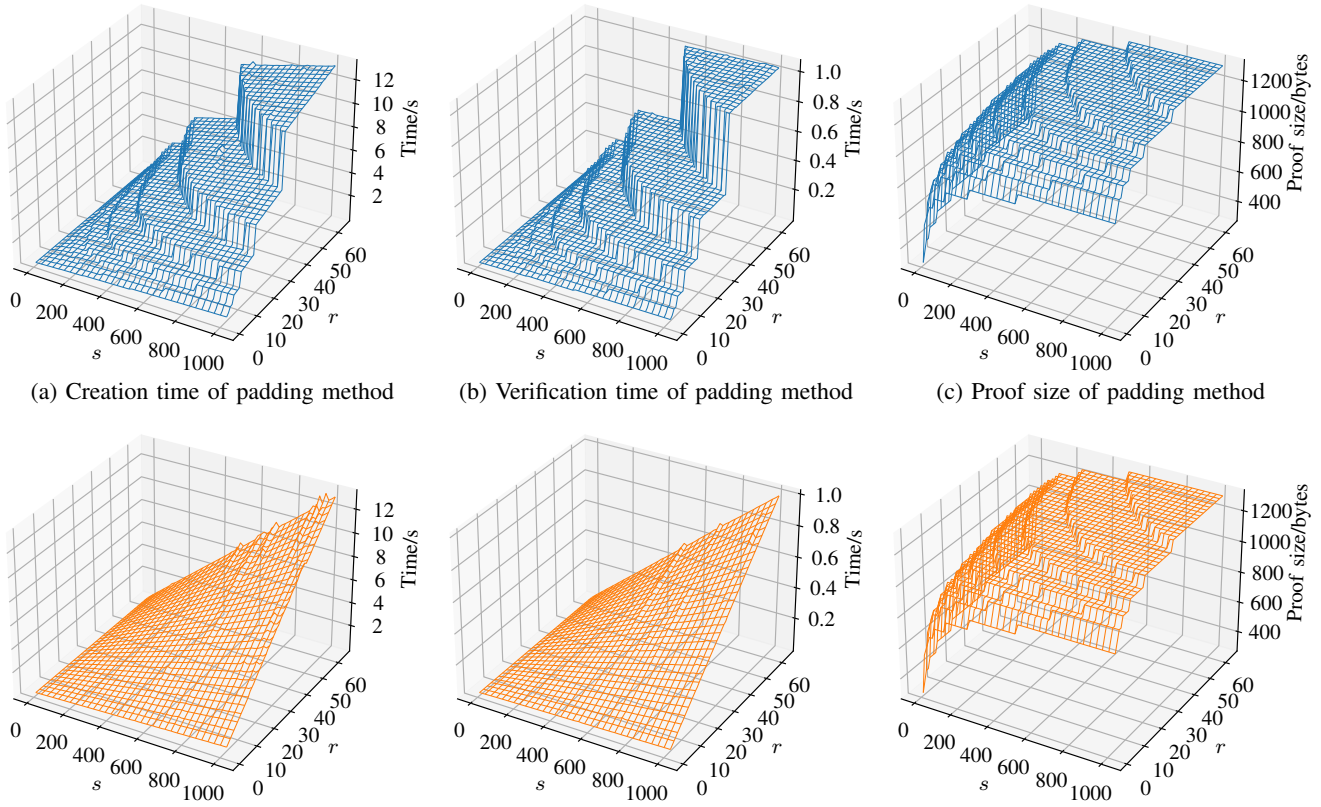
4.1. Theoretical Evaluation

We summarize the theoretical performance of Bulletproofs and $\mathsf{SIPA}(f'_{\mathrm{c}})$ in Table 2 listing the overall unit operations required by the prover and the verifier during the argument. As shown in Table 2, Springproofs require less group multiplication and group exponentiation which dominate the computational cost when n is not a power of 2.

4.2. Range Proof

For evaluating the performance of Springproofs, we use an implementation [18] of Bulletproofs in Rust as the reference, and implement the padding method, $\mathsf{SIPA}(f_{\mathrm{p}})$, and a variant of pre-compression method, $\mathsf{SIPA}(f'_{\mathrm{c}})$, based on the reference implementation to incorporate Bulletproofs into Springproofs.

The Rust implementation of Bulletproofs are built with Ristretto group [19], which is based on the quotient group of the elliptic curve Curve25519 and provides 126 bits of



(a) Creation time of padding method (b) Verification time of padding method (c) Proof size of padding method

(d) Creation time of pre-compression method (e) Verification time of pre-compression method (f) Proof size of pre-compression method

Figure 3: Aggregated range proof performance and proof size comparison between padding method and pre-compression method, with different number of proofs s and range $[0, 2^r)$

security. With point compression, every Ristretto point can be stored in 32 bytes.

We run the aggregated range proof in different range $[0, 2^r)$ where $r \in [1, 64]$ and the number of proofs $s \in [1, 1024]$, then measure the size of the generated proofs and time consumption of proof creation and proof verification. For comparison, we perform the experiments on Springproofs and Bulletproofs with padding illustrated in Figure 3.

Comparing Figures 3a, 3b and 3c with Figures 3d, 3e and 3f, we see the proof sizes of pre-compression method and the padding method are identical with different numbers of proofs and ranges, since they both reach the theoretical lower bound of Springproofs. But the pre-compression method performs better than the padding method. When sr is only marginally larger than a power of 2, the pre-compression method achieves an almost 2 times speedup compared with the padding method. In particular, when $s = 576$, $r = 57$, compared with padding method, pre-compression method takes only 59.2% and 49.9% of time to generate and verify an aggregated range proof respectively.

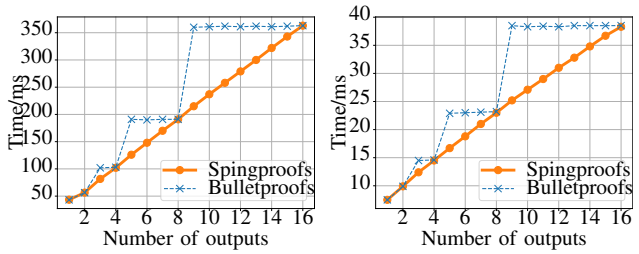
4.3. Generation and Verification of Monero Transaction

Based on Bulletproofs, we implement $\mathsf{SIPA}(f'_{\mathrm{c}})$ in C++ and integrate it into Monero. In Monero, group operations are based on Ed25519 which is birationally equivalent to Curve25519.

We run experiments on Monero with $\mathsf{SIPA}(f'_{\mathrm{c}})$ and Monero with plain Bulletproofs respectively. We record the transaction generation and verification time with different number of outputs. The batch verification time of 100 transactions is also measured with different number of outputs.

The results of transaction generation and verification time are illustrated in Figure 4. From Figures 4a and 4b, we conclude that the generation and verification time is nearly identical when the number of outputs is a power of 2 in the transaction. However, Springproofs has lower computational complexity when the number of outputs is not a power of 2. In particular, compared to Bulletproofs based Monero, Springproofs based Monero only takes 59.7% of time to generate a transaction with 9 outputs, and only 65.5% of time to verify it.

The results of transaction batch verification time are illustrated in Figure 5. Since multi-exponentiation are merged



(a) Generation time (b) Verification time

Figure 4: Generation and verification time for a Monero transaction

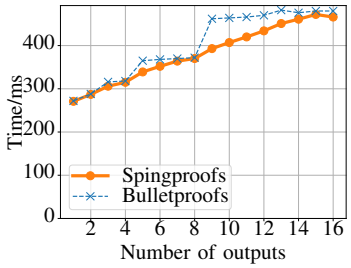


Figure 5: A batch verification time of 100 Monero transactions

into a single multi-exponentiation in the batch verification, the difference from Springproofs and Bulletproof on the batch verification time is less than that on a single transaction verification time. However, when the number of outputs of transaction is not a power of 2, Springproofs based Monero is still notably more efficient than Bulletproofs based Monero. In particular, when number of outputs is 9, the batch verification with Springproofs is 17.5% faster than the batch verification with Bulletproof.

4.4. Privacy Computing of Arithmetic Circuit

We implement the argument of arithmetic circuits from [3] and accelerate the IPA of the argument with an instantiation of Springproofs, $\mathsf{SIPA}(f'_c)$. The implementation is built on top of `arkworks` [20], which is a Rust library and offers implementations for finite fields, elliptic curves, and arithmetic circuits in *Rank-1 Constraint System* (R1CS) format.

In the experiments, we Springproofs the performance of Bulletproofs and Springproofs in different arithmetic circuits, including SHA256 circuits, membership proof for Merkle tree, circuits for statistics such as expected value and variance. For each circuit, we compare the proof generation time, proof verification time, and proof size between Bulletproofs and Springproofs. The performance of Groth16 is also measured in the experiments of circuit for statistics.

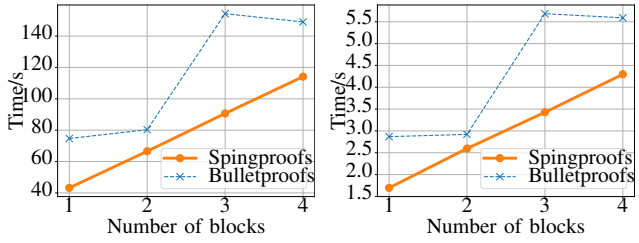
It is worth noting that `arkworks` only provides arithmetic circuits in R1CS format, different from the format supported by Springproofs and Bulletproofs natively. Therefore, a format reduction of arithmetic circuits is required as a preprocessing step in the experiments involving arithmetic circuits. See the reduction method in detail in Appendix C.

4.4.1. SHA256 Circuits. In the SHA256 circuit experiments, we test four circuits with different sizes of input messages. Specifically, the size of the message is one of 512 bits (1 Block), 1024 bits (2 Blocks), 1536 bits (3 Blocks), and 2048 bits (4 Blocks). We choose the secp256k1 provided by `arkworks` as the cyclic group in Bulletproofs and Springproofs. The curve has 128 bits of security. With point compression, every element on the secp256k1 fits in 33 bytes.

We compare the proof sizes between Bulletproofs and Springproofs in Table 3. From Table 3, we observe that Springproofs and Bulletproofs have the same proof size. Thus, Springproofs preserve the succinct proof sizes of Bulletproofs. The generation time and the verification time of proofs for SHA256 are illustrated in Figure 6. Since the numbers of multiplication gates of all the SHA256 circuits are not powers of 2, the performance of Springproofs exceeds that of Bulletproofs in all cases. Specifically, when the size of the input message is 1 block, Springproofs only takes 57.8% and 59.1% of the time to generate and verify a single proof of SHA256 circuit compared to Bulletproofs.

TABLE 3: Proof Size of Bulletproofs and $\mathsf{SIPA}(f'_c)$ for SHA256 Circuits

Number of Blocks	Number of Multiplication Gates	Proof Size of Bulletproofs	Proof Size of Springproofs
1	73950	1562 bytes	1562 bytes
2	115030	1562 bytes	1562 bytes
3	156110	1628 bytes	1628 bytes
4	197190	1628 bytes	1628 bytes



(a) Generation time (b) Verification time

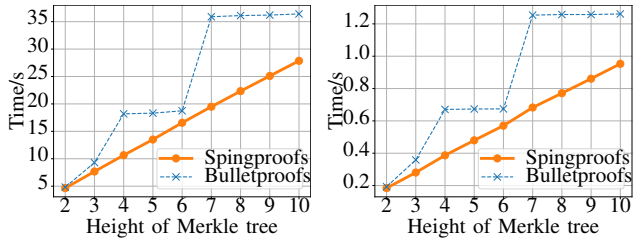
Figure 6: Comparison of generation and verification time for a single proof of SHA256 circuit between Bulletproofs and $\mathsf{SIPA}(f'_c)$

4.4.2. Membership Proof for Merkle Tree. In the experiment of membership proof for Merkle tree. The Merkle trees are built on the Pedersen hash. The Pedersen hash used in the experiments has a 4-bits window size and is based on a twisted Edwards curve whose base field is the scalar field of the curve BLS12-377 [21]. Therefore, we use the curve $\mathbb{G}_1$ of BLS12-377 as the curve to build Bulletproofs and Springproofs in the membership proof.

We list the proof size of membership proof of the Bulletproofs and Springproofs in Table 4. From Table 4, we conclude the proof size of Springproofs is consistent with the proof size of Bulletproofs. Furthermore, the size of proof grows logarithmically in the number of multiplication gates. The generation time and verification time of membership proofs are illustrated in Figure 7 for Merkle trees with different heights. We observe that Springproofs is notably faster than Bulletproofs in membership proof. Compared with Bulletproofs, when the height of Merkle tree is 7, Springproofs only takes 54.3% of time to generate a membership proof and 54.4% of time to verify.

TABLE 4: Proof Size of Bulletproofs and SIPA(f'_c) for Membership Proof of Merkle Tree

Height of Merkle Tree	Number of Multiplication Gates	Proof Size of Bulletproofs	Proof Size of Springproofs
2	3809	1712 bytes	1712 bytes
3	6346	1808 bytes	1808 bytes
4	8883	1904 bytes	1904 bytes
5	11420	1904 bytes	1904 bytes
6	13957	1904 bytes	1904 bytes
7	16494	2000 bytes	2000 bytes
8	19031	2000 bytes	2000 bytes
9	21568	2000 bytes	2000 bytes
10	24105	2000 bytes	2000 bytes

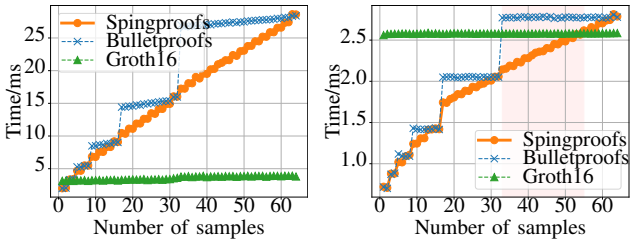


(a) Generation time (b) Verification time

Figure 7: Comparison of generation and verification time for a single membership proof for Merkle tree between Bulletproofs and SIPA(f'_c)

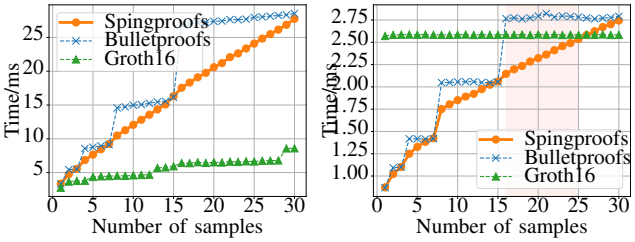
4.4.3. Circuits for Statistics. For evaluating the performance of Springproofs on circuits for typical statistics, we build arithmetic circuits for expected value and variance. Given n samples, the circuit for expected value has $\lceil n/2 \rceil$ multiplication gates, and the circuit for variance has n multiplication gates. We compare the generation time and verification time of proof for Bulletproofs, Springproofs and Groth16. We choose secp256k1 to build the Bulletproofs and the Springproofs, and use a Groth16 implementation based on BLS12-377 from `arkworks` [20] in the experiments.

The experimental results of generation time and verification time of proof of the expected value are illustrated in Figure 8 for different numbers of samples. From the figure, we conclude that Springproofs exceed Bulletproofs in the speed of proof generation and proof verification. Interestingly, compared with Groth16 and Bulletproofs, when the number of samples is within $[33, 55]$, the performance of Springproofs exceeds both the Groth16 and Bulletproofs.



(a) Generation time (b) Verification time

Figure 8: Comparison of generation and verification time for a single proof of expected value between Bulletproofs, SIPA(f'_c) and Groth16



(a) Generation time (b) Verification time

Figure 9: Comparison of generation and verification time for a single proof of variance between Bulletproofs, SIPA(f'_c), and Groth16

The experimental results of generation time and verification time of proof of variance are illustrated in Figure 9 for different numbers of samples. From the figure, we conclude that Springproofs outperform Bulletproofs in the speed of proof generation and proof verification. Notice the performance of Springproofs exceeds both the Groth16 and Bulletproofs when the number of samples is within $[16, 25]$.

5. Conclusion

In order to overcome the limitation of the length of witness vectors for IPAs and improve the efficiency of the arguments, we have presented Springproof, a framework to derive new IPAs from existing arguments. We have analyzed the characteristics of Bulletproofs based Springproofs, and have found optimal scheme functions of the arguments. With an optimal scheme function, Springproofs support vectors of arbitrary lengths, and achieve the theoretical lower bound of computational and communication cost, hence leading to an optimal IPA, by which we do not pay the cost of zero-padding.

Springproofs are well-suited for blockchain applications. By employing Springproofs in confidential transactions of cryptocurrencies, we improve the efficiency of a transaction generation and verification, and increase the throughput of

confidential transactions accordingly. For privacy related smart contracts, Springproofs make the privacy computing of arithmetic circuits always efficient compared with Bulletproofs, meanwhile the proof sizes of them are the same.

Experimental results show that Springproofs preserve the succinct proof size of Bulletproofs, and Springproofs can achieve an almost 2 times speedup compared with Bulletproofs, when the length of the witness vectors is only marginally larger than a power of 2. Digital currencies such as Monero may also benefit from Springproofs. Compared with Bulletproofs based Monero, Springproofs based Monero only takes 59.7% and 65.5% of time cost to generate and verify a transaction with 9 outputs respectively. Moreover, Springproofs increase the range of parameters on which the performance of privacy computing supported by Bulletproofs exceeds that of Groth16, for instance privacy computing on the statistics like expected value and variance. Note that the available circuit size can be further increased because the most time-consuming multi-exponentiation part of Springproofs can be parallelly accelerated, while the paring of Groth16 could be hardly parallelly accelerated. Meanwhile, Springproofs naturally inherit the advantages of Bulletproofs, such as without initial trusted setup, aggregation, and batch verification.

Acknowledgement

The authors and the corresponding author Dr. Ming Su would express their sincere thanks to anonymous reviewers and Shepherd for the comments that greatly improve this paper. This research is supported in part by Foundation of State Key Laboratory of Public Big Data(No. PBD2022-12), the Science and Technology Development Plan of Tianjin(19YFZCSF00900, 20JCZDJC00610), the NSF of China(No. 62272253, No. 62272252, No. 62141412), and the Fundamental Research Funds for the Central Universities.

References

[1] S. Noether, A. Mackenzie, and t. M. R. Lab, "Ring Confidential Transactions," *Ledger*, vol. 1, pp. 1–18, Dec. 2016. [Online]. Available: https://ledger.pitt.edu/ojs/ledger/article/view/34

[2] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards Privacy in a Smart Contract World," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science, J. Bonneau and N. Heninger, Eds. Cham: Springer International Publishing, 2020, pp. 423–443.

[3] 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)*. San Francisco, CA: IEEE, May 2018, pp. 315–334. [Online]. Available: https://ieeexplore.ieee.org/document/8418611/

[4] 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*, ser. Lecture Notes in Computer Science, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer, 2016, pp. 327–357.

[5] Z. Zhang, Z. Zhou, W. Li, and H. Tao, "An Optimized Inner Product Argument with More Application Scenarios," in *Information and Communications Security*, ser. Lecture Notes in Computer Science, D. Gao, Q. Li, X. Guan, and X. Liao, Eds. Cham: Springer International Publishing, 2021, pp. 341–357.

[6] X. Li, C. Xu, and Q. Zhao, "Shellproof: More Efficient Zero-Knowledge Proofs for Confidential Transactions in Blockchain," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, May 2020, pp. 1–5.

[7] H. Chung, K. Han, C. Ju, M. Kim, and J. H. Seo, "Bulletproofs+: Shorter Proofs for a Privacy-Enhanced Distributed Ledger," *IEEE Access*, vol. 10, pp. 42 067–42 082, 2022, conference Name: IEEE Access.

[8] V. Daza, C. Ràfols, and A. Zacharakis, "Updateable Inner Product Argument with Logarithmic Verifier and Applications," in *Public-Key Cryptography – PKC 2020*, ser. Lecture Notes in Computer Science, A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, Eds. Cham: Springer International Publishing, 2020, pp. 527–557.

[9] J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," in *Advances in Cryptology – EUROCRYPT 2016*, ser. Lecture Notes in Computer Science, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer, 2016, pp. 305–326.

[10] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and communications security*, ser. CCS '93. New York, NY, USA: Association for Computing Machinery, Dec. 1993, pp. 62–73. [Online]. Available: https://doi.org/10.1145/168588.168596

[11] T. Attema and R. Cramer, "Compressed Sigma-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics," in *Advances in Cryptology – CRYPTO 2020*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 513–543.

[12] T. Attema, S. Fehr, and M. Klooß, "Fiat-Shamir Transformation of Multi-round Interactive Proofs," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, E. Kiltz and V. Vaikuntanathan, Eds. Cham: Springer Nature Switzerland, 2022, pp. 113–142.

[13] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely, "Proofs for Inner Pairing Products and Applications," in *Advances in Cryptology – ASIACRYPT 2021*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds. Cham: Springer International Publishing, 2021, pp. 65–97.

[14] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More," 2017, report Number: 1066. [Online]. Available: https://eprint.iacr.org/2017/1066

[15] C. Gentry, S. Halevi, and V. Lyubashevsky, "Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties," in *Advances in Cryptology – EUROCRYPT 2022*, ser. Lecture Notes in Computer Science, O. Dunkelman and S. Dziembowski, Eds. Cham: Springer International Publishing, 2022, pp. 458–487.

[16] R. W. F. Lai, G. Malavolta, and V. Ronge, "Succinct Arguments for Bilinear Group Arithmetic: Practical Structure-Preserving Cryptography," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 2057–2074. [Online]. Available: https://doi.org/10.1145/3319535.3354262

[17] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *Advances in Cryptology — CRYPTO '91*, ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer, 1992, pp. 129–140.

[18] zkcrypto, "Bulletproofs," Jan. 2023, original-date: 2021-01-14T21:17:17Z. [Online]. Available: https://github.com/zkcrypto/bulletproofs

[19] Henry de Valence, Isis Lovecruft, and Tony Arcieri, "The Ristretto Group." [Online]. Available: https://ristretto.group/

[20] arkworks contributors, "arkworks zkSNARK ecosystem," 2022. [Online]. Available: https://arkworks.rs

[21] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "ZEXE: Enabling Decentralized Private Computation," in *2020 IEEE Symposium on Security and Privacy (SP)*, May 2020, pp. 947–964, iSSN: 2375-1207.

[22] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge," in *Advances in Cryptology – CRYPTO 2013*, ser. Lecture Notes in Computer Science, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer, 2013, pp. 90–108.

[23] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, vol. 4, p. 220, 2016.

Appendix A. Formal Definitions of Arguments of Knowledge

Definition 2 states that an argument of knowledge is an argument with completeness and knowledge soundness. And a zero-knowledge argument is a HVZK argument of knowledge. In this section, we give formal definitions of completeness, knowledge soundness and zero-knowledge (HVZK).

Definition 6 (Completeness). An argument $(\mathrm{Setup}, \mathcal{P}, \mathcal{V})$ is *complete*, if for any statement $\mathbb{x} \in \mathcal{L}$ and witness $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, there exists a negligible function $\mu(\lambda)$, such that

$$\Pr\left[\langle \mathcal{P}(\sigma,\mathbb{x},\mathbb{w}), \mathcal{V}(\sigma,\mathbb{x})\rangle = 1 \middle| \sigma \leftarrow \mathrm{Setup}(1^\lambda)\right] \geq 1 - \mu(\lambda).$$

Definition 7 (Knowledge Soundness). An interactive argument $(\mathrm{Setup}, \mathcal{P}, \mathcal{V})$ for relation $\mathcal{R}$ is *knowledge sound*, if there exists an efficient *knowledge extractor* $\mathcal{E}$ and a positive polynomial z, such that for any statement $\mathbb{x} \in \{0,1\}^\lambda$ and prover $\mathcal{P}^*$,

$$\Pr\left[(\mathbb{x},\mathbb{w}') \in \mathcal{R} \middle| \begin{matrix}\sigma \leftarrow \mathrm{Setup}(1^\lambda),\\ \mathbb{w}' \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathbb{x})\end{matrix}\right] \geq \frac{\epsilon(\mathcal{P}^*, \mathbb{x}) - \kappa(|\mathbb{x}|)}{z(|\mathbb{x}|)},$$

where $\epsilon(\mathcal{P}^*, \mathbb{x}) := \Pr\left[\langle \mathcal{P}^*(\sigma, \mathbb{x}), \mathcal{V}(\sigma, \mathbb{x})\rangle = 1\right]$, κ is the *knowledge error* and is negligible. $\mathcal{E}$ has a black-box oracle access to the prover $\mathcal{P}^*$.

Definition 8 (HVZK). An interactive argument $(\mathrm{Setup}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ is *honest-verifier zero-knowledge (HVZK)*, if there exists a PPT simulator SIM, such that there exists a negligible function $\epsilon(\lambda)$, for all adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$,

$$\left| \begin{matrix} \Pr\left[\begin{matrix}(\mathbb{x},\mathbb{w})\in\mathcal{R}\wedge\\ \mathcal{A}_1(tr)=1\end{matrix}\middle|\begin{matrix}\sigma\leftarrow\mathrm{Setup}(1^\lambda)\\ (\mathbb{x},\mathbb{w},\rho)\leftarrow\mathcal{A}_2(\sigma)\\ tr\leftarrow\mathsf{SIM}(\mathbb{x},\rho)\end{matrix}\right] - \\ \Pr\left[\begin{matrix}(\mathbb{x},\mathbb{w})\in\mathcal{R}\wedge\\ \mathcal{A}_1(tr)=1\end{matrix}\middle|\begin{matrix}\sigma\leftarrow\mathrm{Setup}(1^\lambda)\\ (\mathbb{x},\mathbb{w},\rho)\leftarrow\mathcal{A}_2(\sigma)\\ tr\leftarrow\langle\mathcal{P}(\sigma,\mathbb{x},\mathbb{w}),\mathcal{V}_\rho(\sigma,\mathbb{x})\rangle\end{matrix}\right] \end{matrix} \right| \leq \epsilon(\lambda),$$

where ρ is the randomness used by the verifier $\mathcal{V}_\rho$, and $\epsilon(\lambda)$ is the *statistical difference* between $\mathsf{SIM}(\mathbb{x}, \rho)$ and $\langle \mathcal{P}(\sigma, \mathbb{x}, \mathbb{w}), \mathcal{V}_\rho(\sigma, \mathbb{x})\rangle$.

By Fiat-Shamir transformation, we can obtain a non-interactive random oracle argument from a public-coin interactive argument. For such a non-interactive argument, the random oracle must be considered in knowledge soundness.

Definition 9 (Knowledge Soundness - Non-interactive). A non-interactive random oracle argument $(\mathrm{Setup}, \mathcal{P}, \mathcal{V})$ for relation $\mathcal{R}$ is *knowledge sound*, if there exists a efficient *knowledge extractor* $\mathcal{E}$ and a positive polynomial z, such that for any statement $\mathbb{x} \in \{0,1\}^\lambda$ and prover $\mathcal{P}^*$ with at most Q query to the random oracle RO,

$$\Pr\left[(\mathbb{x},\mathbb{w}')\in\mathcal{R}\middle|\begin{matrix}\sigma\leftarrow\mathrm{Setup}(1^\lambda),\\ \mathbb{w}'\leftarrow\mathcal{E}_{\mathcal{P}^*}(\mathbb{x})\end{matrix}\right] \geq \frac{\epsilon(\mathcal{P}^*, \mathbb{x}) - \kappa(|\mathbb{x}|, Q)}{z(|\mathbb{x}|)},$$

where $\epsilon(\mathcal{P}^*, \mathbb{x}) := \Pr\left[\langle \mathcal{P}^*_{\mathsf{RO}}(\sigma, \mathbb{x}), \mathcal{V}_{\mathsf{RO}}(\sigma, \mathbb{x})\rangle = 1\right]$, $\kappa(\lambda, Q) \in [0,1]$ is the *knowledge error* and is negligible. $\mathcal{E}$ has a black-box oracle access to the prover $\mathcal{P}^*$ and can manipulate the random oracle RO for $\mathcal{A}$ arbitrarily.

Appendix B. Proof of Theorem 5

Proof: Consider a $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ with padding steps and compression steps. Suppose the length of the witness vectors is n, the total number of padding zeros in $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is n_p, and the public input of $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is $\mathbb{x} = (\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, \mathbf{g}_\mathrm{p}, \mathbf{h}_\mathrm{p} \in \mathbb{G}^{n_\mathrm{p}}, u, u', P \in \mathbb{G})$. We now construct a simulator of $\mathsf{SIPA}_{\mathsf{IPA}}(f)$. Let τ be the total number of steps of $\mathsf{SIPA}_{\mathsf{IPA}}(f)$. For $i \in (1 : \tau)$, let n_i be the length of the witness vector at the beginning of the ith step, $n_{\mathrm{p},i}$ be the number of rest padding zeros at the beginning of the ith step, and $\mathbb{x}_i = (\mathbf{g}_i, \mathbf{h}_i, \mathbf{g}_\mathrm{p}, \mathbf{h}_\mathrm{p}, u, u', P_i)$ be the public input of the ith step. Thus, we obtain $n_1 = n$, $n_{\mathrm{p},1} = n_\mathrm{p}$, and $\mathbb{x}_1 = \mathbb{x}$. Furthermore, denoted by $m = \left|\{i \mid f(n_i) \in \mathscr{P}(\mathbb{Z}^+), i \in (1,\tau)\}\right|$ the total number of compression steps in the argument. Denoted by SIM_k the simulator of IPA_k. Finally, let $\epsilon(\lambda)$ be the statistical difference between the transcript simulated by SIM_k and the real transcript.

Suppose the transcripts before the ith step is simulated, we now simulate the transcript in the ith step. Depending on the specific value of $f(n_i)$, the simulator works differently:

(i) $f(n_i) \in \mathbb{Z}^+$. Hence, step i is a padding step, and there is no interaction between the prover and the verifier. The simulator calculates

$$\begin{gathered} n_{i+1} = n_i + f(n_i),\ n_{\mathrm{p},i+1} = n_{\mathrm{p},i} - f(n_i),\ P_{i+1} = P_i, \\ \mathbf{g}_{i+1} = \mathbf{g}_i \parallel \mathbf{g}_{\mathrm{p},i+1_{[1:f(n_i)]}},\ \mathbf{h}_{i+1} = \mathbf{h}_i \parallel \mathbf{h}_{\mathrm{p},i+1_{[1:f(n_i)]}}, \\ \mathbf{g}_{\mathrm{p},i+1} = \mathbf{g}_{\mathrm{p},i_{[f(n_i)+1:n_{\mathrm{p},i}]}},\ \mathbf{h}_{\mathrm{p},i+1} = \mathbf{h}_{\mathrm{p},i_{[f(n_i)+1:n_{\mathrm{p},i}]}}, \end{gathered}$$

and continues simulating for the rest of the argument;

(ii) $f(n_i) \in \mathscr{P}(\mathbb{Z})^+$. Hence, step i is a compression step. Let $T_i = f(n_i)$, $S_i = (1 : n_i) \setminus T_i$. The simulator samples $Q'_i \xleftarrow{\$} \mathbb{G}$ at random uniformly if $S_i \neq \emptyset$, otherwise the simulator directly let $Q'_i = P_i$. With Q'_i, the simulator runs

$$\mathcal{T}_i = \mathsf{SIM}_{|T_i/2|}(\mathbf{g}_{[T_i]}, \mathbf{h}_{[T_i]}, u, u', Q'_i),$$

to simulate a transcript $\mathcal{T}_i$ for the ith step. From $\mathcal{T}_i$, we calculate the input of the reduced public relation, $\mathbf{g}'_i, \mathbf{h}'_i \in \mathbb{G}^{|T_i|/2}, u, u', P'_i \in \mathbb{G}$. In order to simulate the rest of the argument, the simulator computes

$$n_{i+1} = n_i - |T_i|/2,\ n_{\mathrm{p},i+1} = n_{\mathrm{p},i},\ P_{i+1} = P'_i P_i Q'^{-1}_i,$$
$$\mathbf{g}_{i+1} = \mathbf{g}'_i \parallel \mathbf{g}_{i[S]},\ \mathbf{h}_{i+1} = \mathbf{h}'_i \parallel \mathbf{h}_{i[S]},$$
$$\mathbf{g}_{\mathrm{p},i+1} = \mathbf{g}_{\mathrm{p},i},\ \mathbf{h}_{\mathrm{p},i+1} = \mathbf{h}_{\mathrm{p},i}.$$

and continues simulating for the rest of the argument.

By simulating with the aforementioned approach for τ times, finally the whole transcript of the argument is simulated. Therefore a simulator SIM of $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is constructed. Since $m < \mathrm{poly}(\lambda)$ and the simulator calls SIM_k exactly m times, the simulator is efficient. Additionally, all the inputs to SIM_k include parts either calculated by the verifier directly, or derived from the public input and the simulated transcript at the previous step. Because IPA_k always introduces new independently distributed blinding factors in reduction, the transcripts at different steps are independent from each other. Therefore, for any adversary $\mathcal{A}_1$ and each step i, let $\Pr_{\mathrm{sim},i}$ be the probability that $\mathcal{A}_1$ outputs 1 for $\mathcal{T}_i$, $\Pr_{\mathrm{real},i}$ be the probability that $\mathcal{A}_1$ outputs 1 for the real transcript in step i. We have $|\Pr_{\mathrm{sim},i} - \Pr_{\mathrm{real},i}| \leq \epsilon(\lambda)$ for $i \in (1:\tau)$. Therefore, the statistical difference between the final simulated transcript and the real transcript is

$$\left|\prod_{i=1}^{m} \Pr_{\mathrm{sim},i} - \prod_{i=1}^{m} \Pr_{\mathrm{real},i}\right| \leq \sum_{i=1}^{m} |\Pr_{\mathrm{sim},i} - \Pr_{\mathrm{real},i}| \leq m\epsilon(\lambda).$$

Because $m < \mathrm{poly}(\lambda)$, $m\epsilon(\lambda)$ is negligible. We now conclude that $\mathsf{SIPA}_{\mathsf{IPA}}(f)$ is honest-verifier zero-knowledge. □

Appendix C. Reduction of Arithmetic Circuit Formats

In this section, we discuss the reduction of arithmetic circuit formats between *rank-1 constraint systems* (*R1CS*) and Bulletproofs (*BP*).

First, R1CS is a format of arithmetic circuits proposed by Ben-Sasson et al. [22] and is widely used because of the simplicity and efficiency of the format in arithmetization, see the definition as follows:

Definition 10 (R1CS). An R1CS is a relation of the form

$$\begin{aligned}&\{(A, B, C \in \mathbb{F}_p^{N_\mathrm{c} \times (1+N_\mathrm{i}+N_\mathrm{w})}, \mathbf{x} \in \mathbb{F}_p^{N_\mathrm{i}};\ \mathbf{w} \in \mathbb{F}_p^{N_\mathrm{w}}) :\\ &A \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w}) \circ B \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w}) = C \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\},\end{aligned} \tag{15}$$

where $\mathbf{x}$ are the public inputs of the circuit, $\mathbf{w}$ are the private inputs of the circuit, and A, B, C specify the constraints of the circuit.

Next, BP is the arithmetic format used in Bulletproofs. A significant feature of BP is that the Pedersen commitment can be used as the input of the circuit directly. In contrast, using commitments as inputs in R1CS requires embedding the commitment circuit into the R1CS constraints, which brings overhead, and the definition of BP is in Definition 11.

Definition 11 (BP). A constraint system in BP format is a relation of the form

$$\begin{aligned}&\{(g, h \in \mathbb{G}, \mathbf{V} \in \mathbb{G}^{N_\mathrm{v}}, W_L, W_R, W_O \in \mathbb{F}_p^{N_\mathrm{q} \times N_\mathrm{a}},\\ &W_V \in \mathbb{F}_p^{N_\mathrm{q} \times N_\mathrm{v}}, \mathbf{c} \in \mathbb{F}_p^{N_\mathrm{q}};\ \mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O \in \mathbb{F}_p^{N_\mathrm{a}}, \mathbf{v}, \boldsymbol{\gamma} \in \mathbb{F}_p^{N_\mathrm{v}}) :\\ &V_j = g^{v_j} h^{\gamma_j}, \forall j \in (1:N_\mathrm{v}) \wedge \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \wedge\\ &W_L \cdot \mathbf{a}_L + W_R \cdot \mathbf{a}_R + W_O \cdot \mathbf{a}_O = W_V \cdot \mathbf{v} + \mathbf{c}\},\end{aligned}$$

where g, h are the generators of the Pedersen commitment, $\mathbf{V}$ are the commitment inputs, $\mathbf{v}$ are the committed values, $\boldsymbol{\gamma}$ are the blinding factors, $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O$ are the left inputs, the right inputs and the outputs of all 2-fan-in multiplication gates of the circuit respectively, and W_L, W_R, W_O, W_V and $\mathbf{c}$ specify the constraints of the circuit.

C.1. BP to R1CS

For the commitment constraints $V_j = g^{v_j} h^{\gamma_j}, j \in (1 : N_\mathrm{v})$ in BP, we have to incorporate the constraints into the R1CS by verifying the commitments are consistent with committed private inputs in arithmetic circuit. A possible solution is to introduce a new curve, such as the Jubjub curve in ZCash [23], for the Pedersen hash.

For the rest of the constraints, it is straightforward to transform the BP into R1CS format by letting

$$\mathbf{w} = (\mathbf{a}_L \parallel \mathbf{a}_R \parallel \mathbf{a}_O \parallel \mathbf{v}) \in \mathbb{F}_p^{3N_\mathrm{a}+N_\mathrm{v}},$$
$$A = \begin{pmatrix} & I & & & \\ \mathbf{0} & W_L & W_R & W_O & \mathbf{0} \end{pmatrix} \in \mathbb{F}_p^{(N_\mathrm{a}+N_\mathrm{q}) \times (1+3N_\mathrm{a}+N_\mathrm{v})},$$
$$B = \begin{pmatrix} & & I & & \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix} \in \mathbb{F}_p^{(N_\mathrm{a}+N_\mathrm{q}) \times (1+3N_\mathrm{a}+N_\mathrm{v})},$$
$$C = \begin{pmatrix} & & & I & \\ \mathbf{c} & \mathbf{0} & \mathbf{0} & \mathbf{0} & W_V \end{pmatrix} \in \mathbb{F}_p^{(N_\mathrm{a}+N_\mathrm{q}) \times (1+3N_\mathrm{a}+N_\mathrm{v})},$$

and $\mathbf{x}$ be a zero-dimensional vector in R1CS. It is clear that $\big(A \cdot (\mathbf{1}^1 \parallel \mathbf{w})\big) \circ \big(B \cdot (\mathbf{1}^1 \parallel \mathbf{w})\big) = C \cdot (\mathbf{1}^1 \parallel \mathbf{w})$ holds in the new R1CS if and only if $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \wedge W_L \cdot \mathbf{a}_L + W_R \cdot \mathbf{a}_R + W_O \cdot \mathbf{a}_O = W_V \cdot \mathbf{v} + \mathbf{c}$ in the original BP. Combined with the constraints of commitments, a BP relation is transformed into a R1CS.

C.2. R1CS to BP

Since R1CS does not natively support commitments as inputs, we do not consider the commitment constraints of BP here.

To begin with the reduction, if we split (15) row by row, for $j \in (1 : N_\mathrm{c})$ we have

$$\langle A_j, (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\rangle \cdot \langle B_j, (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\rangle = \langle C_j, (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\rangle,$$

where A_j, B_j and C_j are the jth row of A, B and C respectively. Now we put all the indices of the equations whose $\langle A_j, (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\rangle$ and $\langle B_j, (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\rangle$ are not constant into a set J. Let

$$J \coloneqq \{j \in (1 : N_\mathrm{c}) \mid A_{j,[2:N_\mathrm{i}+N_\mathrm{w}]} \neq \mathbf{0} \wedge B_{j,[2:N_\mathrm{i}+N_\mathrm{w}]} \neq \mathbf{0}\},$$
$$K \coloneqq (1 : N_\mathrm{c}) \setminus J.$$

It is clear that

$$\begin{aligned}&\big(A_{[J]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\big) \circ \big(B_{[J]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\big)\\ &= C_{[J]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\end{aligned} \tag{16}$$

represents the constraints of multiplication gates, and

$$\begin{aligned}&\big(A_{[K]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\big) \circ \big(B_{[K]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\big)\\ &= C_{[K]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w})\end{aligned} \tag{17}$$

represents the linear constraints. Based on (16), we assign values to the witnesses of the new BP as follows:

$$\begin{cases}\mathbf{a}_L = A_{[J]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w}),\\ \mathbf{a}_R = B_{[J]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w}),\\ \mathbf{a}_O = C_{[J]} \cdot (\mathbf{1}^1 \parallel \mathbf{x} \parallel \mathbf{w}).\end{cases} \tag{18}$$

If we consider (18) as a system of linear equations with unknowns $\mathbf{x}$ and $\mathbf{w}$, there are three cases:

1) There exist $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O$, such that (18) has an exact solution. In this case, we obtain a solution
$$(\mathbf{x} \parallel \mathbf{w}) = M_L\mathbf{a}_L + M_R\mathbf{a}_R + M_O\mathbf{a}_O. \tag{19}$$
2) For any $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O$, (18) is either overdetermined or underdetermined. For the case where (18) is underdetermined, we expand the dimension of $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O$ and add new equations $\mathbf{a}_{L,i} = \mathbf{w}_i$ to (18) for some components of $\mathbf{w}$, such that (18) combined with the new equations does have an exact solution. The final solution is similar to (19).
3) For any $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O$, (18) is overdetermined, implying that the original R1CS is impossible to be satisfied. In this case, the R1CS is equivalent to any BP without any solutions. Therefore, any unsolvable BP is a valid reduction. We ignore this case in rest of the discussion, since an unsolvable R1CS is unlikely in real scenarios.

Notice that the solution (19) maps $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O$ into $\mathbf{x}$ and $\mathbf{w}$. By plugging (19) into (17), we have

$$\begin{aligned}&\big(A \cdot (\mathbf{1}^1 \parallel (M_L\mathbf{a}_L + M_R\mathbf{a}_R + M_O\mathbf{a}_O))\big) \circ\\ &\big(B \cdot (\mathbf{1}^1 \parallel (M_L\mathbf{a}_L + M_R\mathbf{a}_R + M_O\mathbf{a}_O))\big)\\ =&C \cdot (\mathbf{1}^1 \parallel (M_L\mathbf{a}_L + M_R\mathbf{a}_R + M_O\mathbf{a}_O)).\end{aligned}$$

With rearrangements to the linear relations, we obtain an equivalent equation

$$W_L \cdot \mathbf{a}_L + W_R \cdot \mathbf{a}_R + W_O \cdot \mathbf{a}_O = \mathbf{c},$$

where W_L, W_R, $W_O \in \mathbb{F}_p^{|K|\times|J|}$ and $\mathbf{c} \in \mathbb{F}_p^{|K|}$. Combined with the relation $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O$, we have a new BP whose $N_{\mathrm{q}} = |K|$, $N_{\mathrm{a}} = |J|$ and $N_{\mathrm{v}} = 0$. By applying (19) to $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O$, we derive the witnesses of the original R1CS from the witnesses of the new BP. Therefore, the new BP is indeed the reduction of the original R1CS.

Appendix D. Meta-Review

D.1. Summary

The paper introduces Springproofs, a new framework for inner product arguments (IPAs) that makes it more efficient to use inner-product arguments like Bulletproofs with vectors of size n that are not powers-of-two. Existing methods involve zero-padding the vector up to the nearest-power-of-two, which can be expensive for n just greater than a power of two.

The Springproofs framework modifies the recursion in IPAs to avoid the need for this zero-padding, and shows that one can achieve up to 50% performance improvements over standard padding-based IPAs. This is demonstrated by a fairly comprehensive evaluation that demonstrates the efficacy of Springproofs in various applications of IPAs, including polynomial commitments, NIZKs, and range proofs.

D.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field.

D.3. Reasons for Acceptance

The paper's techniques provide a valuable step forward in improving the efficiency of all existing IPAs. This directly impacts existing deployments of IPAs, such as range proofs, which have thus far had to unnecessarily pay the cost of zero-padding.