





Cougar: Cubic Root Verifier Inner Product Argument under Discrete Logarithm Assumption

Hyeonbum Lee , Seunghun Paik , Hyunjung Son , and Jae Hong Seo *

Department of Mathematics & Research Institute for Natural Sciences,
Hanyang University, Seoul 04763, Republic of Korea
{leehb3706, whitesoonguh, dk9050rx, jaehongseo}@hanyang.ac.kr

Abstract. An inner product argument (IPA) is a cryptographic primitive used to construct a zero-knowledge proof system, which is a notable privacy-enhancing technology. We propose a novel efficient IPA called *Cougar*. *Cougar* features cubic root verifier and logarithmic communication under the discrete logarithm (DL) assumption. At Asiacrypt2022, Kim et al. proposed two square root verifier IPAs under the DL assumption. Our main objective is to overcome the limitation of square root complexity in the DL setting. To achieve this, we combine two distinct square root IPAs from Kim et al.: one with pairing (*Protocol3*; one was later named *Leopard*) and one without pairing (*Protocol4*). To construct *Cougar*, we first revisit *Protocol4* and reconstruct it to make it compatible with the proof system for the homomorphic commitment scheme. Next, we utilize *Protocol3* as the proof system for the reconstructed *Protocol4*. Finally, to facilitate proving the relation between elliptic curve points appearing in *Protocol4*, we introduce a novel Plonkish-based proof system equipped with custom gates for mixed elliptic curve addition. We show that *Cougar* indeed satisfies all the claimed features, along with providing a soundness proof under the DL assumption. In addition, we implemented *Cougar* in Rust, demonstrating that the verification time of *Cougar* increases much slowly as the length of the witness N grows, compared to other IPAs under the DL assumption and transparent setup: *BulletProofs* and *Leopard*. Concretely, *Cougar* takes 0.346s for verification in our setting when $N = 2^{20}$, which is a $50\times$ speed-up from *BulletProofs*.

Keywords: Inner product argument, zero knowledge proof, polynomial commitment, discrete logarithm assumption

1 Introduction

Zero-Knowledge Proof (ZKP) is one of the privacy-enhancing technologies spotlighted by many international organizations and others [42]. ZKP is a protocol allowing a prover to convince a verifier that a statement is true without

* Corresponding Author.

Schemes	Comm.	Prover	Verifier	Assumption	Setup	Pairing
Updatable IPA[23]	$O(\log_2 N)$	$O(N)$	$O(\log_2 N)$	DL, DPair	Trusted	Yes
Dory[41]	$O(\log_2 N)$	$O(N)$	$O(\log_2 N)$	SXDH	Trustless	Yes
Bulletproofs[11,16]	$O(\log_2 N)$	$O(N)$	$O(N)$	DL	Trustless	No
Leopard[39,38]	$O(\log_2 N)$	$O(N)$	$O(\sqrt{N})$	DL	Trustless	Yes
Protocol4[39]	$O(\log_2 N)$	$O(N)$	$O(\sqrt{N} \log_2 N)$	DL	Trustless	No
TENET[40]	$O(\sqrt{\log_2 N})$	$O(N \cdot 2^{\sqrt{\log_2 N}})$	$O(N/2^{\sqrt{\log_2 N}})$	DL, DPair	Trustless	Yes
This Work	$O(\log_2 N)$	$O(N)$	$O(\sqrt[3]{N} \sqrt{\log_2 N})$	DL	Trustless	Yes

Comm., Prover, and Verifier mean cost of communication, prover computation, and verifier computation, respectively. Pairing means requirement of pairing-friendly groups.

Table 1. Comparison Table of IPAs for length- N vectors

leaking any additional information [32]. ZKP schemes are employed as foundational components in various cryptographic applications, including identification [25,20], verifiable computation [7,9,50,12], and confidential transactions [48,16,36,22,31]. To this end, ZKP for proving the satisfiability of the arithmetic (AC) is essential, and several constructions [35,11,16,6,29,18,33,21] have been proposed.

Among them, one notable approach is to utilize an inner product argument (IPA) as a building block of the ZKP scheme [11,16,18,41,22]. Bootle et al. [11] first proposed an IPA with logarithmic proof size under the discrete logarithm (DL) assumption, and later, Bünz et al. [16] improved the IPA, which is called Bulletproofs. In [18], Bünz et al. proposed a paradigm for constructing ZKPs by applying a polynomial commitment scheme (PCS), which can be seen as a specialized form of IPA, to a polynomial interactive oracle proof (PIOP) system. Following this paradigm, the complexity of ZKPs heavily relies on that of the IPA. Hence, the efficient construction of an IPA is crucial for designing efficient ZKPs.

Bulletproofs is a widely known IPA because of its efficient proof size and lack of reliance on trusted parties. However, one of the main drawbacks of Bulletproofs is its linear verification cost, which makes it challenging to apply in certain applications, such as verifiable computation and incrementally verifiable computation. To avoid linear verification, Daza et al. [23] proposed a sublinear verifier IPA using bilinear pairing. However, the sublinear IPA [23] requires a trusted setup, which means that a trusted third party is necessary to generate a common reference string (CRS), whereas Bulletproofs does not. After, Lee [41] proposed a sublinear pairing-based IPA, called Dory, without a trusted setup. However, Dory depends on stronger cryptographic assumptions, such as the symmetric external Diffie-Hellman (SXDH) assumption.

Without relying on more cryptographic assumptions than Bulletproofs, Kim et al. [39] proposed two square root verifier IPAs, pairing-based IPA (Protocol3) and pairing-free IPA (Protocol4). Both IPAs provide linear prover and logarithm communication, equivalent to Bulletproofs. In [38], Kim et al. presented optimizations and a concrete implementation of Protocol3, which is called Leopard.

Contribution. In this paper, we introduce the first cubic root verifier and logarithmic communication IPA, called *Cougar*, under the DL assumption with transparent setup. Our IPA maintains the same assumptions and setup as previous IPAs, such as [11,16,39]. In Table 1, we provide a comparison on computation and communication complexity between previous IPA proposals and ours.

To achieve cubic root verifier, we deepen our understanding of previous pairing-free IPA *Protocol4* from the lens of a two-tier commitment scheme. From this, we present a generalization of *Protocol4* and improve its verifier’s complexity by combining with *Leopard* in the second layer. With these framework-level improvements, we also propose novel techniques to efficiently deal with relations between elliptic curve points in *Protocol4*. First, we present a Plonkish-based proof system tailored for these relations, which utilizes new custom gates for *mixed* elliptic curve group operations. Moreover, we present a Plonk-friendly encoding method for a homomorphic PCS, enabling an efficient consistency check between committed vectors and intermediate wire values in AC.

We prove that *Cougar* satisfy the claimed features under the DL assumption. Furthermore, we also conducted experimental analyses of *Cougar* by implementing it in our experimental environment. We compared *Cougar* with other previous IPAs under the same setting: *BulletProofs* and *Leopard*. We showed that the verification cost of *Cougar* increases much slower than these two prior works as the length of the witness vectors increases.

1.1 Technical Overview

Two-tier Commitment with Proof. We first revisit *Protocol4*, pairing-free square root verifier IPA [39]. The main idea of *Protocol4* is a two-tier commitment scheme with a proof for the second layer. The two-tier commitment scheme comprises two layers. In the first layer, mn -length vectors are compressed into n elliptic curve points using a parallel Pedersen commitment scheme with a m -dimensional commitment key. Subsequently, these n elliptic curve points are interpreted as $3n$ -length vectors in the embedding field. In the second layer, these vectors are further compressed into a single elliptic curve point through a Pedersen commitment scheme with a $3n$ -dimensional commitment key.

The proof of the second layer is intricately connected to the commitment scheme used in the second layer. Concretely, the proof should ensure knowledge of the first layer results and the elliptic curve relation between them. To address this issue, homomorphic commitment and a related proof system are required. From this viewpoint, we generalize the second layer commitment from the Pedersen commitment to any homomorphic commitments.

Proof for Elliptic Curve Relation. The second-layer proof concerns the elliptic curve relation. It is constructed by reducing the elliptic curve relation to an arithmetic relation over the embedding field and then adapting the Plonkish proof system [55], which is specialized for elliptic curve operations.

In the second-layer, the prover must convince knowledge of vectors and their corresponding elliptic curve relation. Designing a proof system that satisfies both

conditions is challenging, as the elliptic curve relation involves all committed vectors. To address this, we propose a Plonk-friendly extended PCS derived from a homomorphic PCS, compatible with the Plonkish proof system for elliptic curve operations. Specifically, this new PCS ensures consistency between committed vectors and wire polynomials from Plonkish arithmetization.

Cubic Root Verifier Inner Product Argument. From the above results, we conclude that the protocol features $O(\log_2 mn)$ communication and $O(m + \|\mathcal{V}_{\text{Eval}}(n \log_2 m)\|)$, where $\|\mathcal{V}_{\text{Eval}}(n \log_2 m)\|$ is the verifier complexity of `Eval`, the evaluation protocol of the PCS for degree $O(n \log_2 m)$ polynomials. Then, we apply `Leopard` evaluation protocol, which features square root verifier complexity. Finally, we set the parameters $m = \sqrt[3]{N}$ and $n = \sqrt[3]{N^2}$, where N is the length of the witness vectors. Then, the total verifier complexity is $O(m + \|\mathcal{V}_{\text{Eval}}(n \log_2 m)\|) = O(\sqrt[3]{N} + \sqrt[3]{N} \sqrt{\log_2 N})$, which is the cubic root of N .

1.2 Related Works

ZK Argument based on Discrete Logarithm Setting. Groth [34] first proposed a sublinear ZK argument for AC under the DL assumption, and Seo [49] improved it. These works also feature constant round complexity. Later works [11,16,18,41,22,39] focus on reducing communication complexity (to logarithmic scale) rather than round complexity (allowing logarithmic complexity). Starting from Bulletproofs [11,16], various works have been proposed to improve the verifier complexity of Bulletproofs [41,22,39,24]. In a different view point, Kim et al. [39] proposed a sublogarithmic communication ZK argument for the first time, and then Lee et al. [40] enhanced it from a linear verifier cost to a sublinear one with sublogarithmic communication. Furthermore, Zhang et al. [56] proposed an efficient framework for IPA to handle arbitrary length of vectors directly, and some works [30,53] proposed improved IPAs for specific applications.

ZK Argument based on Unknown order group. DL-based ZK arguments feature a small proof size but require a large verification cost. To handle this, Búnz et al. [18] and Arun et al. [2] proposed short size ZK arguments with logarithmic verification. However, both schemes demands large prover cost.

ZK Argument based on Lattice Setting. To overcome vulnerability against quantum computer-aided attacks, several ZK arguments from cryptographic assumptions on lattice, e.g. short integer solution, ring learning with errors, have been proposed [3,14,17,4]. However, lattice based ZK arguments has more expensive communication costs than DL based ZK arguments due to limited challenge sampling issue.

ZK Argument based on Collision Resistant Hash function. In another direction for post-quantum security, zero-knowledge (ZK) schemes based on cryptographic hash functions have also been proposed [8,6,13,52]. Since hash-based ZK schemes do not rely on cryptographic groups, the computations for

both the prover and the verifier are relatively simple. However, without an underlying algebraic structure, it is challenging to apply algebraic techniques such as proof aggregation and algebraic reduction.

2 Preliminaries

We first define the notations used in the paper. $[\ell]$ denotes a set of integers from 1 to ℓ . We denote a negligible function as negl . For a prime p , we denote asymmetric bilinear groups of order p , $\mathbb{G}_1, \mathbb{G}_2$, and \mathbb{G}_t with a non-degenerated bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$. We use additive notation to describe group operations on $\mathbb{G}_1, \mathbb{G}_2$, and \mathbb{G}_t . To denote a scalar multiplication, we denote $[k]G$ for a scalar $k \in \mathbb{Z}_p$ and $G \in \mathbb{G}$. We prefer to use upper and lowercase letters to denote group elements and field elements, respectively. We use bold font to represent vectors in \mathbb{Z}_p^m or \mathbb{G}^m . For a vector $\mathbf{a} \in \mathbb{Z}_p^m$ and $i \in [m]$, we use a_i (non-bold style letter with a subscript i) to denote the i -th element of \mathbf{a} . We use \parallel notation to represent concatenation of two vectors, *i.e.*, for $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^m$, $\mathbf{a} \parallel \mathbf{b} = (a_1, \dots, a_m, b_1, \dots, b_m)$.

For $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^m$, $\mathbf{G} \in \mathbb{G}_1^m$, and $\mathbf{H} \in \mathbb{G}_2^m$, we use the following vector notations:

- **Component-wise addition** : $\mathbf{a} + \mathbf{b} = (a_1 + b_1, \dots, a_m + b_m) \in \mathbb{Z}_p^m$ and $\mathbf{G} + \mathbf{H} = (G_1 + H_1, \dots, G_m + H_m) \in \mathbb{G}^m$.
- **Component-wise product** : $\mathbf{a} \circ \mathbf{b} = (a_1 b_1, \dots, a_m b_m) \in \mathbb{Z}_p^m$.
- **Multi-Scalar Multiplication** : $[\mathbf{x}]\mathbf{G} = \sum_{i \in [m]} [x_i]G_i \in \mathbb{G}_1$.
- **Inner Pairing Product** : $\mathbf{E}(\mathbf{G}, \mathbf{H}) = \sum_{i \in [m]} e(G_i, H_i) \in \mathbb{G}_t$.

Parallel Multi-Scalar Multiplication. Let $\mathbf{a} \in \mathbb{Z}_p^{m \times n}$ be a matrix and $\mathbf{G} \in \mathbb{G}^m$ be group elements. We denote $[\mathbf{a}]\mathbf{G} := ([\mathbf{a}_1]\mathbf{G}, \dots, [\mathbf{a}_n]\mathbf{G})$, where $\mathbf{a}_i \in \mathbb{Z}_p^m$ is the i -th column vector of matrix \mathbf{a} .

Argument of Knowledge. Let \mathcal{R} be a polynomial-time verifiable relation consisting of common reference string (CRS), statement, and witness, denoted by σ, x , and w respectively. An interactive argument system for relation \mathcal{R} consists of three probabilistic polynomial-time algorithms (PPTs) $(\mathcal{K}, \mathcal{P}, \mathcal{V})$. The \mathcal{K} algorithm takes the security parameter λ and outputs CRS σ , which is the input of \mathcal{P} and \mathcal{V} . \mathcal{P} and \mathcal{V} generate a transcript interactively, denoted by $tr \leftarrow \langle \mathcal{P}(\sigma, x, w), \mathcal{V}(\sigma, x) \rangle$. At the end of the transcript, \mathcal{V} outputs a bit, 0(reject) or 1(accept). An argument of knowledge (AoK) is a special case of an argument system that satisfies the properties of completeness and witness extractability.

Definition 1 (Argument of Knowledge). *Let \mathcal{R} be a polynomial-time verifiable relation. We call that the argument system $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ for the relation \mathcal{R} an Argument of knowledge if the following properties hold:*

[Completeness]: For every PPT adversary \mathcal{A} , the following inequality holds:

$$\Pr \left[\begin{array}{l} tr \leftarrow \langle \mathcal{P}(\sigma, x, w), \mathcal{V}(\sigma, x) \rangle \\ tr \text{ is accepting} \end{array} \middle| \begin{array}{l} \sigma \leftarrow \mathcal{K}(1^\lambda); \\ (x, w) \leftarrow \mathcal{A}(\sigma) \\ \wedge (\sigma, x; w) \in \mathcal{R} \end{array} \right] > 1 - \text{negl}(\lambda)$$

[Witness Extended Emulation]: For every deterministic polynomial prover \mathcal{P}^* , which may not follow \mathcal{P} , and all pairs of PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, there exists a PPT emulator \mathcal{E} , the following inequality holds:

$$\left| \Pr \left[\mathcal{A}_1(tr) = 1 \mid \begin{array}{l} \sigma \leftarrow \mathcal{K}(1^\lambda); (x, s) \leftarrow \mathcal{A}_2(\sigma); \\ tr \leftarrow \langle \mathcal{P}^*(\sigma, x, s), \mathcal{V}(\sigma, x) \rangle \end{array} \right] - \Pr \left[\mathcal{A}_1(tr) = 1 \wedge \begin{array}{l} \sigma \leftarrow \mathcal{K}(1^\lambda); (x, s) \leftarrow \mathcal{A}_2(\sigma); \\ (\sigma, w, x) \in \mathcal{R} \mid (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(\sigma, x), tr \text{ is accepting} \end{array} \right] \right| < \text{negl}(\lambda)$$

The emulator \mathcal{E} can access the oracle $\mathcal{O} = \langle \mathcal{P}^*(\sigma, x, s), \mathcal{V}(\sigma, x) \rangle$, which outputs the transcript between \mathcal{P}^* and \mathcal{V} . \mathcal{E} permits to rewind \mathcal{P}^* at a specific round and rerun \mathcal{V} using fresh randomness. s can be considered as the state of \mathcal{P}^* .

Trusted Setup. In some arguments, the key generation algorithm takes a trapdoor that should not be revealed to anyone, including the prover and verifier. In this case, CRS generation should be run by a trusted third party. A setting requiring a trusted party is called the trusted setup.

Definition 2 (Discrete Logarithm Relation Assumption). Let \mathcal{G} be a group generator that takes security parameters λ and then outputs \mathbb{G} , describing a group of order p . We say that \mathbb{G} satisfies the discrete logarithm relation (DLR) assumption if, for all non-uniform polynomial-time adversaries \mathcal{A} , the following inequality holds:

$$\Pr \left[\mathbf{a} \neq \mathbf{0} \wedge \mathbf{g}^{\mathbf{a}} = 1_{\mathbb{G}} \mid \begin{array}{l} (p, g, \mathbb{G}) \leftarrow \mathcal{G}(1^\lambda), \mathbf{g} \xleftarrow{\$} \mathbb{G}^n; \\ \mathbf{a} \leftarrow \mathcal{A}(\mathbf{g}, p, g, \mathbb{G}) \end{array} \right] \leq \text{negl}(\lambda)$$

It is well-known that the discrete logarithm relation (DLR) assumption is equivalent to the discrete logarithm (DL) assumption [16,39].

Definition 3 (Commitment Scheme). A commitment scheme \mathcal{C} consists of three PPT algorithms: a key generation Gen , a commitment Com , and an open Open . A commitment scheme $\mathcal{C} = (\text{Gen}, \text{Com}, \text{Open})$ over a message space \mathbb{M} , a random space \mathbb{R} , and a commitment space \mathbb{C} is defined by:

- $\text{Gen}(1^\lambda, \ell) \rightarrow \text{ck}$: On input security parameter λ and dimension of message space ℓ , sample commitment key ck
- $\text{Com}(\text{ck}, m; r) \rightarrow C$: Take commitment key ck , message $m \in \mathbb{M}$, and randomness $r \in \mathbb{R}$, output commitment $C \in \mathbb{C}$
- $\text{Open}(\text{ck}, m, r, C) \rightarrow 0/1$: Take commitment key ck , message $m \in \mathbb{M}$, randomness $r \in \mathbb{R}$, and commitment $C \in \mathbb{C}$ output 1 if $\text{Com}(\text{ck}, m; r) = C$, 0 otherwise.

Since the Open algorithm can be described by using Com algorithm, we omit the Open algorithm from the commitment scheme \mathcal{C} . Now, we call $\mathcal{C} = (\text{Gen}, \text{Com})$ a commitment scheme if the following properties hold:

[Binding]: For any expected PPT adversary \mathcal{A} ,

$$\Pr \left[m_0 \neq m_1 \mid \begin{array}{l} \text{ck} \leftarrow \text{Gen}(1^\lambda, \ell); (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\text{ck}) \\ \wedge C_0 = C_1 \text{ where } C_i = \text{Com}(\text{ck}, m_i; r_i) \end{array} \right] \leq \text{negl}(\lambda)$$

[Hiding]: For any expected PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$

$$\left| \Pr \left[\mathbf{b} = \mathbf{b}' \mid \begin{array}{l} \text{ck} \leftarrow \text{Gen}(1^\lambda, \ell); (m_0, m_1, \text{state}) \leftarrow \mathcal{A}_1(\text{ck}); \\ \mathbf{b} \xleftarrow{\$} \{0, 1\}, r \xleftarrow{\$} \mathcal{R}, \\ C \leftarrow \text{Com}(\text{ck}, m_b; r); \mathbf{b}' \leftarrow \mathcal{A}_2(\text{ck}, C, \text{state}), \end{array} \right] - \frac{1}{2} \right| \leq \text{negl}(\lambda)$$

Additionally, we call a commitment scheme \mathcal{C} is (additively) homomorphic if the following property holds:

[(Additive) Homomorphic]: For any commitment key $\text{ck} \leftarrow \text{Gen}(1^\lambda, \ell)$ and pairs of message-randomness $(m_0, r_0), (m_1, r_1) \in \mathcal{M} \times \mathcal{R}$, the following equality holds: $\text{Com}(\text{ck}, m_0; r_0) + \text{Com}(\text{ck}, m_1; r_1) = \text{Com}(\text{ck}, m_0 + m_1; r_0 + r_1)$

Homomorphic Vector Commitment Schemes. A homomorphic vector commitment scheme is a homomorphic commitment for N -dimensional message, etc. \mathbb{Z}_p^N or \mathbb{G}^N . We introduce two widely used homomorphic vector commitment schemes: *Pedersen vector commitment* [43] and *AFGHO group commitment* [1].

Pedersen vector commitment. Pedersen vector commitment, denoted by $\mathcal{C}_{\text{Ped}} = (\text{Gen}_{\text{Ped}}, \text{Com}_{\text{Ped}})$, is a commitment scheme over message space \mathbb{Z}_p^N as follows:

- $\text{Gen}_{\text{Ped}}(1^\lambda, N) \rightarrow (\mathbf{G}, H)$:
 1. Sample $\mathbf{G} \xleftarrow{\$} \mathbb{G}^N$ and $H \xleftarrow{\$} \mathbb{G}$
 2. Output $\text{ck} = (\mathbf{G}, H) \in \mathbb{G}^N \times \mathbb{G}$
- $\text{Com}_{\text{Ped}}((\mathbf{G}, H), \mathbf{a}; r) \rightarrow C$:
 1. Compute $C = [\mathbf{a}]\mathbf{G} + [r]H$
 2. Output $C \in \mathbb{G}$

Specially, we sometimes use subscript Ped, p for Pedersen commitment over group \mathbb{G}_p of order p to distinguish base group.

AFGHO group commitment. AFGHO group commitment $\mathcal{C}_{\text{GC}} = (\text{Gen}_{\text{GC}}, \text{Com}_{\text{GC}})$ is a commitment scheme over message space \mathbb{G}_1^N as follows:

- $\text{Gen}_{\text{GC}}(1^\lambda, N) \rightarrow (\mathbf{F}, K)$:
 1. Sample $\mathbf{F} \xleftarrow{\$} \mathbb{G}_2^N$ and $K \xleftarrow{\$} \mathbb{G}_t$
 2. Output $\text{ck} = (\mathbf{F}, K) \in \mathbb{G}_2^N \times \mathbb{G}_t$
- $\text{Com}_{\text{GC}}((\mathbf{F}, K), \mathbf{G}; r) \rightarrow C$:
 1. Compute $C = \mathbf{E}(\mathbf{G}, \mathbf{F}) + [r]K$
 2. Output $C \in \mathbb{G}_t$

Polynomial Commitment Scheme [37,18]. A polynomial commitment scheme (PCS) is a special case of the commitment scheme that commits the given polynomial within the specific degree bound d . PCS allows convincing polynomial evaluation without opening the polynomial itself. Concretely, PCS contains an argument system $\text{Eval} = (\mathcal{K}, \mathcal{P}, \mathcal{V})$ for the following relation:

$$\mathcal{R}_{\text{Eval}} = \left\{ \left(\text{ck}_{\text{PC}}, C \in \mathcal{C}, z, y \in \mathbb{Z}_p, d \in \mathbb{N}; f \in \mathbb{Z}_p^{\leq d}[X] \right) : \begin{array}{l} C = \text{Com}(\text{ck}_{\text{PC}}, f(X)) \wedge y = f(z) \end{array} \right\} \quad (1)$$

The formal definition of PCS is given as below:

Definition 4 (Polynomial Commitment Scheme). A polynomial commitment scheme $\text{PCS} = (\text{Gen}, \text{Com}, \text{Eval})$ consists of key generation algorithm Gen , commitment algorithm Com , and argument system Eval for the relation $\mathcal{R}_{\text{Eval}}$. We call $\text{PCS} = (\text{Gen}, \text{Com}, \text{Eval})$ is a polynomial commitment scheme if the following properties hold:

- The (Gen, Com) is commitment schemesatisfies the binding property.
- The argument system Eval is an AoK for the relation $\mathcal{R}_{\text{Eval}}$ in Eq. (1)

3 Construction of Cubic Root Verifier IPA

3.1 Two-tier Commitment Scheme and Inner Product Argument

A two-tier commitment is a commitment scheme for a two-dimensional array, e.g. $\mathbb{Z}_p^{m \times n}$. Using two-tier commitment scheme has some merits. To construct an IPA with a two-tier commitment, the size of the common reference string (CRS) can be reduced sublinear of $N = mn$, concretely, $O(n + m)$. This reduced CRS leads to a reduction in the verification cost of IPA [19,41,39,38]. A two-tier commitment scheme is constructed by combining two distinct commitment schemes $\mathcal{C}_1 = (\text{Gen}_1, \text{Com}_1)$ and $\mathcal{C}_2 = (\text{Gen}_2, \text{Com}_2)$. For a matrix in $\mathbb{Z}_p^{m \times n}$, commit m row vectors using the first commitment algorithm Com_1 in parallel. After then, with regard m commitments from Com_1 as a message of Com_2 , use the second commitment algorithm Com_2 , and output it.

Definition 5 (Two-tier Commitment Scheme). Let $\mathcal{C}_1 = (\text{Gen}_1, \text{Com}_1)$ and $\mathcal{C}_2 = (\text{Gen}_2, \text{Com}_2)$ be commitment schemes over (message, commitment, randomness) space $(\mathbb{Z}_p^n, \mathcal{C}_1, \mathcal{R}_1)$ and $(\mathbb{C}_1^m, \mathcal{C}_2, \mathcal{R}_2)$ respectively. Then, the commitment scheme $\mathcal{C} = (\text{Gen}, \text{Com})$ over space $(\mathbb{Z}_p^{m \times n}, \mathcal{C}_2, \mathcal{R}_1 \times \mathcal{R}_2)$ is called as a two-tier commitment scheme based on \mathcal{C}_1 and \mathcal{C}_2 defined by:

- | | |
|---|--|
| <ul style="list-style-type: none"> – $\text{Gen}(1^\lambda, mn) \rightarrow \text{ck} = (\text{ck}_1, \text{ck}_2)$: <li style="padding-left: 20px;">1. Run $\text{Gen}_1(1^\lambda, n) \rightarrow \text{ck}_1$ <li style="padding-left: 20px;">2. Run $\text{Gen}_2(1^\lambda, m) \rightarrow \text{ck}_2$ <li style="padding-left: 20px;">3. Return $\text{ck} = (\text{ck}_1, \text{ck}_2)$ | <ul style="list-style-type: none"> – $\text{Com}(\text{ck}, M; (\mathbf{r}, r_f)) \rightarrow \mathcal{C}$: <li style="padding-left: 20px;">1. Comp. $\text{Com}_1(\text{ck}_1, M_i; r_i) \rightarrow C_i, \forall i$ <li style="padding-left: 20px;">2. Comp. $\text{Com}_2(\text{ck}_2, \mathcal{C}; r_f) \rightarrow \mathcal{C}$ <li style="padding-left: 20px;">3. Return \mathcal{C} |
|---|--|

Specially, we use roman-style to denote commitment from two-tier commitment schemes. In terms of IPA, the binding property of the commitment is sufficient for ensuring soundness. So, we omit the randomness r in the commitment algorithm, which does not affect the binding property. Hereafter, we simply write $\text{Com}(\text{ck}, M)$ to describe the commitment algorithm for a message M .

Pairing-based Two-tier Commitment Scheme. From two commitment schemes $\mathcal{C}_1 = (\text{Gen}_{\text{Ped}}, \text{Com}_{\text{Ped}})$ and $\mathcal{C}_2 = (\text{Gen}_{\text{GC}}, \text{Com}_{\text{GC}})$ over spaces $(\mathbb{Z}_p^{mn}, \mathbb{G}_1^n, \mathbb{G}_1)$ and $(\mathbb{G}_1^m, \mathbb{G}_2^m, \mathbb{G}_t)$ respectively, one can construct a homomorphic two-tier commitment scheme. The homomorphic two-tier commitment is widely used for constructing sublinear verifier IPA schemes [19,41,39]. The homomorphic property helps to apply the folding technique in Bulletproofs; however, this construction is restricted to a choice of a base group: pairing-friendly elliptic curves.

Doubly-Pedersen Two-tier Commitment Scheme. To remove reliance on the pairing operation, Kim et al. proposed Pedersen commitment for the elliptic curve points, which are already committed by the Pedersen commitment scheme. This approach can be viewed as a two-tier commitment scheme using the Pedersen commitment on both the first and second layers. For convenience, we call this commitment scheme as the *Doubly-Pedersen two-tier commitment scheme*. The doubly-Pedersen two-tier commitment process for $\mathbf{a} \in \mathbb{Z}_p^{m \times n}$ is as follows: First, commit each row vector of \mathbf{a} using Pedersen vector commitment

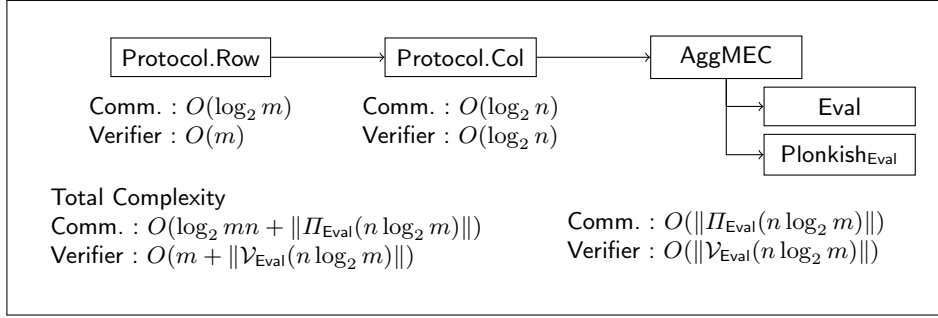


Fig. 1. Overall Process of the Protocol

on the group of elliptic curve points $\mathbb{G} = E(\mathbb{Z}_q)$ over the field \mathbb{Z}_q . After the first layer commitment, one gets n distinct elliptic curve points. For the second layer commitment, one considers n elliptic curve points in $E(\mathbb{Z}_q)$ as coordinates of the field elements in \mathbb{Z}_q and then recommits them using the Pedersen vector commitment on the elliptic curve \mathbb{G}_q of order q .

Homomorphic Vector Commitment in Second Layer. To apply the folding technique [11,16] to the doubly-Pedersen commitment-based IPA, the prover must provide additional proofs to validate the group operations introduced by the Pedersen commitment in the first layer. This is because the doubly-Pedersen two-tier commitment scheme lacks a homomorphic property [39]. Since the homomorphic property of the second commitment simplifies the construction of additional proofs, it is preferable to use a homomorphic commitment at the second layer. Furthermore, the role of the second commitment is to compress a large message into a single commitment, e.g., reducing a vector in \mathbb{Z}_q^N to a single element in \mathbb{C} . Therefore, the second commitment should satisfy the *compression* property, converting an N -dimensional message into a single element.

For a precise description, let us consider the Pedersen commitment scheme $\mathcal{C}_1 = (\text{Gen}_{\text{Ped}}, \text{Com}_{\text{Ped}})$ over $(\mathbb{Z}_p^m, \mathbb{G}_p = E(\mathbb{Z}_q))$ at the first layer and a homomorphic commitment scheme $\mathcal{C}_2 = (\text{Gen}_2, \text{Com}_2)$ over $(\mathbb{Z}_q^{2n}, \mathbb{C})$ at the second layer. At the second commitment, we consider group elements (elliptic curve points) as pair of \mathbb{Z}_q elements following affine coordinates. Now, we consider two-tier commitment scheme $\mathcal{C}_{\text{TC}} = (\text{Gen}_{\text{TC}}, \text{Com}_{\text{TC}})$ as follows:

- $\text{Gen}_{\text{TC}}(1^\lambda, mn) \rightarrow \text{ck} = (\mathbf{G}, \text{ck}_2)$: – $\text{Com}_{\text{TC}}((\mathbf{G}, \text{ck}_2), \mathbf{a} \in \mathbb{Z}_p^{m \times n}) \rightarrow \mathbb{C} \in \mathbb{G}_q$:

<ol style="list-style-type: none"> 1. Run $\text{Gen}_{\text{Ped},p}(1^\lambda, n) \rightarrow \mathbf{G} \in \mathbb{G}_p^n$ 2. Run $\text{Gen}_2(1^\lambda, 2m) \rightarrow \text{ck}_2 \in \mathbb{G}_q^{2m}$ 3. Return $\text{ck} = (\mathbf{G}, \text{ck}_2)$ 	<ol style="list-style-type: none"> 1. Comp. $\text{Com}_{\text{Ped},p}(\mathbf{G}, \mathbf{a}_i) \rightarrow (C_i)_{i=1}^m \in \mathbb{G}_p^m$ 2. Comp. $\text{Com}_2(\text{ck}_2, \mathbb{C}) \rightarrow \mathbb{C} \in \mathbb{C}$ 3. Return \mathbb{C}
--	--

Inner Product Argument with Two-tier Commitment. We focus on construction of IPA with commitment scheme \mathcal{C}_{TC} for the following relation:

$$\mathcal{R}_{\text{GenPT4}}^{m,n} = \left\{ \begin{array}{l} (\mathbf{G}, \mathbf{H} \in \mathbb{G}_p^m, \text{ck}_2, \mathbb{P} \in \mathbb{G}_q, c \in \mathbb{Z}_p; \mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^{m \times n}) : \\ \mathbb{P} = \text{Com}_{\text{TC}}((\mathbf{G} \parallel \mathbf{H}, \text{ck}_2), \mathbf{a} \parallel \mathbf{b}) \wedge c = \langle \mathbf{a}, \mathbf{b} \rangle \end{array} \right\} \quad (2)$$

Following the framework in [39], we construct an IPA in two parts: the reduction part and the proof of the multi-elliptic curve (MEC) operation part. The reduction part reduces the argument from the relation $\mathcal{R}_{\text{GenPT4}}^{m,n}$ to $\mathcal{R}_{\text{GenPT4}}^{m/2,n}$ (Row-reduction) or $\mathcal{R}_{\text{GenPT4}}^{1,n}$ to $\mathcal{R}_{\text{GenPT4}}^{1,n/2}$ (Column-reduction). The overall process of the proposed IPA is as follows: first, the prover and verifier run row-wise reduction Protocol.Row recursively until the row of the witness reaches $m = 1$. Then, they run column-wise reduction Protocol.Col recursively until the column of the witness reaches $n = 1$. Next is proof for the MEC operation part. In this part, the prover and verifier run AggMEC for ensuring elliptic curve relation between witness vectors. In this phase, Eval and Plonkish_{Eval} are used as subroutines. Notice that both have verifier complexity $\|\mathcal{V}_{\text{Eval}}(n \log_2 m)\|$. We illustrate the overall process in Figure 1.

3.2 Reduction Protocol

In the reduction protocol, the prover and verifier recursively run the reduction process: reduction from an argument for vectors to those for half-sized vectors. Contrary to Bulletproofs [11,16] or Leopard [38], the prover and verifier store the history of reduction processes because the verifier has not been convinced of the group operation relation between received commitments yet. The states st_V and st_P role recording the history of the verifier and prover, respectively. st_V and st_P are used to run the aggregated multi-elliptic curve operation proof, AggMEC, which guarantees the validity of the inner value of commitment for every round.

We construct two type of reduction protocol: row-wise reduction protocol Protocol.Row and column-wise reduction protocol Protocol.Col but core idea of them are equivalent. In the reduction process, the \mathcal{P} sends crossed inner product values c_L, c_R with commitments L, R, whose messages are pairs of half-sized witness vectors, to \mathcal{V} . Then, \mathcal{V} sends challenge x to \mathcal{P} . Contrary to other IPAs based on homomorphic commitments, \mathcal{V} cannot update the instance \hat{P} for the next round. To resolve this issue, \mathcal{P} sends an updated instance \hat{P} to \mathcal{V} . In this phase, \mathcal{V} should verify the well-construction of \hat{P} , but we postpone the verification of it and run the reduction recursively. We describe Protocol.Row and Protocol.Col in Algorithm 1 and Algorithm 3 (in Appendix A) respectively.

Theorem 1. *Assume that both Protocol.Col provide perfect completeness and computational witness-extended emulation. Then, Protocol.Row in Algorithm 1 has perfect completeness and computational witness-extended emulation under the DL assumption.*

Theorem 2. *Assume that AggMEC provides perfect completeness and computational witness-extended emulation. Then, Protocol.Col in Algorithm 3 has perfect completeness and computational witness-extended emulation under the DL assumption.*

Due to space constraints, we defer the proofs for Theorem 1 and Theorem 2 in Appendix B and Appendix C, respectively.

Algorithm 1 Protocol.Row

Protocol.Row($\mathbf{G}, \mathbf{H}, (\text{ck}_k)_{k=s}^\mu, \text{ck}_{\text{Col}}, \mathcal{P}, c, st_V; \mathbf{a}, \mathbf{b}, st_P$)
 where $\text{ck}_k = (\text{ck}_{L,k}, \text{ck}_{R,k}, \text{ck}_{P,k})$, $\text{ck}_{\text{Col}} = (\text{ck}_{P,k})_{k=\mu+1}^{\mu+\nu+1}$

- 1: **if** $m = 1$, base case $s = \mu$ **then**:
- 2: \mathcal{P} and \mathcal{V} run Protocol.Col($G, H, \text{ck}_{\text{Col}}, \mathcal{P}, c, st_V; \mathbf{a}, \mathbf{b}, st_P$)
- 3: **else**
- 4: **if** $st_P = \perp$ and $st_V = \perp$ **then**
- 5: \mathcal{P} sets $\mathbf{P} = [\mathbf{a}]\mathbf{G} \parallel [\mathbf{b}]\mathbf{H}$ and adds $(\cdot, \cdot, \mathbf{P})$ into the bottom row of st_P .
- 6: \mathcal{V} adds $(\text{ck}_{P,0}, \cdot, \cdot, \mathbf{P}, \cdot)$ into the bottom row of st_V .
- 7: **else**
- 8: \mathcal{P} refers \mathbf{P} in the bottom row of st_P
- 9: **end if**

Set $\hat{m} = \frac{m}{2}$ and $\mathbf{a} = [\mathbf{a}_L \parallel \mathbf{a}_R]$, $\mathbf{b} = [\mathbf{b}_L \parallel \mathbf{b}_R]$, $\mathbf{G} = \mathbf{G}_L \parallel \mathbf{G}_R$, $\mathbf{H} = \mathbf{H}_L \parallel \mathbf{H}_R$

- 10: \mathcal{P} computes c_L, c_R and L, R and sends them to \mathcal{V} :
 $\mathbf{L} = [\mathbf{a}_L]\mathbf{G}_R \parallel [\mathbf{b}_R]\mathbf{H}_L$, $\mathbf{R} = [\mathbf{a}_R]\mathbf{G}_L \parallel [\mathbf{b}_L]\mathbf{H}_R \in \mathbb{G}_p^{2n}$,
 $c_L = \langle \mathbf{a}_L, \mathbf{b}_R \rangle$, $c_R = \langle \mathbf{a}_R, \mathbf{b}_L \rangle \in \mathbb{Z}_p$,
 $L = \text{Com}_2(\text{ck}_{L,s}, \mathbf{L})$, $R = \text{Com}_2(\text{ck}_{R,s}, \mathbf{R}) \in \mathbb{G}_q$
- 11: \mathcal{V} chooses $x \xleftarrow{\$} \mathbb{Z}_p^*$ and returns it to \mathcal{P}
- 12: \mathcal{P} computes $\hat{\mathbf{P}}$ and sends it to \mathcal{V} :
 $\hat{\mathbf{P}} = [x^{-1}]\mathbf{L} + \mathbf{P} + [x]\mathbf{R} \in \mathbb{G}_p^{2n}$, $\hat{\mathbf{P}} = \text{Com}_2(\text{ck}_{P,s}, \hat{\mathbf{P}}) \in \mathbb{G}_q$
- 13: Both \mathcal{P} and \mathcal{V} update:
 $\hat{\mathbf{G}} = \mathbf{G}_L + [x^{-1}]\mathbf{G}_R$, $\hat{\mathbf{H}} = \mathbf{H}_L + [x]\mathbf{H}_R \in \mathbb{G}_p^{\hat{m}}$, $\hat{c} = x^{-1}c_L + c + xc_R \in \mathbb{Z}_p$
- 14: \mathcal{P} updates $\hat{\mathbf{a}} = \mathbf{a}_L + xa_R$, $\hat{\mathbf{b}} = \mathbf{b}_L + x^{-1}\mathbf{b}_R \in \mathbb{Z}_p^{\hat{m} \times n}$.
- 15: \mathcal{V} adds $(\text{ck}_s, L, R, \hat{\mathbf{P}}, x)$ into the bottom row of st_V .
- 16: \mathcal{P} adds $(\mathbf{L}, \mathbf{R}, \hat{\mathbf{P}})$ into the bottom row of st_P .
- 17: Both \mathcal{P} and \mathcal{V} run Protocol.Row($\hat{\mathbf{G}}, \hat{\mathbf{H}}, (\text{ck}_k)_{k=s+1}^\mu, \text{ck}_{\text{Col}}, \hat{\mathbf{P}}, \hat{c}, st_V; \hat{\mathbf{a}}, \hat{\mathbf{b}}, st_P$)
- 18: **end if**

3.3 Proof for Elliptic Curve Relation

By Theorem 1 and 2, it is sufficient to construct AggMEC to complete IPA for relation $\mathcal{R}_{\text{GenPT4}}^{m,n}$ in Eq. (2). AggMEC guarantees the well-constructed updated instances $\hat{\mathbf{P}}$ from every round of reductions. That is, the k -th row of state tuples $(st_V; st_P)_k = (\text{ck}_k, (L_k, R_k, P_k, x_k); (L_k, R_k, P_k))$ satisfy the following:

1. Commitment

$$L_k = \text{Com}_2(\text{ck}_{L,k}, \mathbf{L}_k), R_k = \text{Com}_2(\text{ck}_{R,k}, \mathbf{R}_k) \text{ for } k = 1, \dots, \mu$$

$$P_k = \text{Com}_2(\text{ck}_{P,k}, \mathbf{P}_k) \text{ for } k = 0, \dots, \mu + \nu - 1, P_{\mu+\nu} = \text{Com}_2(\text{ck}, [a]G \parallel [b]H) \quad (3)$$

2. Elliptic Curve Operation on $\mathbb{G}_p = E(\mathbb{Z}_q)$

$$\bigwedge_{k=0}^{\mu-1} (\mathbf{P}_{k+1} = [x_k^{-1}]\mathbf{L}_{k+1} + \mathbf{P}_k + [x_k]\mathbf{R}_{k+1} \in \mathbb{G}_p^{2n}) \quad (4)$$

$$\bigwedge_{k=\mu}^{\mu+\nu} (\mathbf{P}_{k+1} = (\mathbf{P}_k^{(q_1)} + [x_k]\mathbf{P}_k^{(q_2)}) \parallel \mathbf{P}_k^{(q_3)} + [x_k^{-1}]\mathbf{P}_k^{(q_4)}) \in \mathbb{G}_p^{n/2^{k-\mu}} \quad (5)$$

Plonkish: Proof system for elliptic curve relation. Plonk [29] is one of the well-known methods to represent the circuit satisfiability of the given arithmetic circuit (AC) as the constraints system. By Lagrange interpolation, the latter can be converted to showing the equality of polynomials which can be proved efficiently by PIOP instantiated by PCS [18]. Plonkish [55] is an extension of Plonk by constructing a constraint system about the execution trace for running specific operation, such as elliptic curve operation. The details of designing execution traces for elliptic curve operations are deferred to Appendix F.

Throughout this paper, we will denote $\text{Plonkish}_{\text{Eval}}$ as the proof system for elliptic curve operation utilizing Plonkish method with PCS Eval. It takes a commitment key ck_{PC} for the underlying PCS, selector polynomials $\{s_i(X)\}_{i=0}^{N_g-1}$, gate polynomials $\{g_i(X_1, \dots, X_M)\}_{i=0}^{N_g-1}$, and permutation polynomials $\{r_i(X)\}_{i=0}^{M-1}$ as public inputs. For witnesses, $\text{Plonkish}_{\text{Eval}}$ takes wire polynomials $\{w^{(i)}(X)\}_{i=0}^{M-1}$, which is encoding of inter-values of elliptic curve operation. The detailed procedure of $\text{Plonkish}_{\text{Eval}}$ is provided in Algorithm 4 in Appendix E.

Two Roots of Unity. To construct the protocol, we consider two roots of unity: one for the commitment part and the other for the execution trace of the elliptic operation. Using the two roots of unity, we encode vectors into interpolated polynomial on the power of unities. First, we consider total number d of elements consisting of message vectors \mathbf{L}_k , \mathbf{R}_k , and \mathbf{P}_k of L_k , R_k , and P_k . Since each \mathbf{L}_k , \mathbf{R}_k consist of $2n$ elements for all $k \in [\mu]$, and \mathbf{P}_k consists of $2n$ elements for $k = 0, \dots, \mu$ and $n/2^{k-\mu-1}$ for all $k = \mu + 1 \dots \mu + \nu$, the total number d should be $6n\mu + 4n - 2$. We denote the d -th root of unity as ζ .

Next, we consider the root of unity for the execution trace. In Eq. (4) and (5), the elliptic curve operation consists of $4n\mu + n - 1$ complete additions and $4n\mu + 4n - 2$ multi-scalar multiplications. Each multi-scalar multiplication can be represented as $2 \log_2 q$ complete additions. Then, the total number of complete additions for Eq. (4) and (5) is at most $8n(\mu + 1) \log_2 q$. We choose a sufficiently large integer D that satisfies $D \geq 8n(\mu + 1) \log_2 q$ and $d|D$ (d is a divisor of D). Next, we define the D -root of unity ξ , which will be used for interpolating the wire polynomial in Plonkish. Note that $\zeta = \xi^t$ for some t and each ζ^i and ξ^i is the root of the polynomial $X^d - 1$ and $X^D - 1$ respectively.

Plonkish-friendly Extended Polynomial Commitment Scheme. To prove the consistency of the wire polynomial and commitments L_k, R_k, P_k , we construct a commitment scheme for the message vectors $\mathbf{L}_k, \mathbf{R}_k$ and \mathbf{P}_k considering compatibility with the polynomial commitment scheme. To this end, we first encode vectors $\mathbf{L}_k, \mathbf{R}_k$ and \mathbf{P}_k into polynomials $F_{L,k}, F_{R,k}, F_{P,k}$ and then commit them. The encoding function Enc_{type} takes ξ , index k and a vector \mathbf{a} , returning a polynomial $F_{\text{type},k}$ in $\mathbb{Z}_q[X]$, where $\text{type} \in \{L, R, P\}$. The encoding process extends $2n$ vectors to D -degree polynomials. We intend that each encoded function is *activated* at different positions. That is, for two encoded functions F_{type_1, k_1} and F_{type_2, k_2} with $(\text{type}_1, k_1) \neq (\text{type}_2, k_2)$, $F_{\text{type}_1, k_1}(\xi^i) F_{\text{type}_2, k_2}(\xi^i) = 0$ holds for all $i \in [D]$. In our setting, decoding of a polynomial $F_{\text{type},k}$ can be performed uniquely when the type type and position k are determined. Furthermore, the

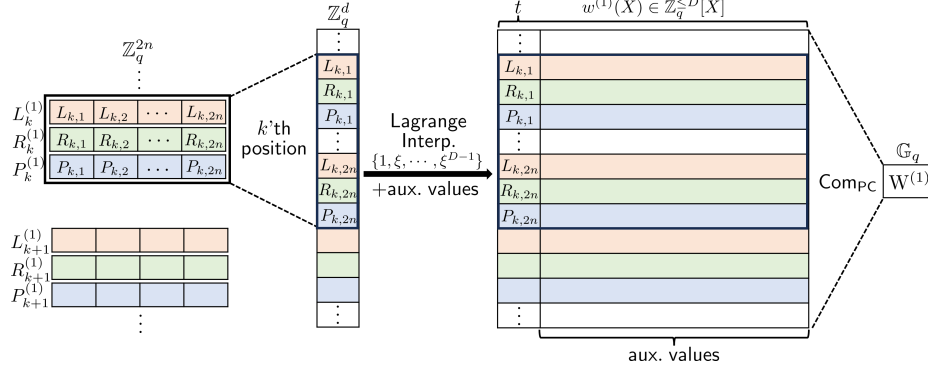


Fig. 2. Structure of Wire Polynomial. Best viewed in color.

sum of two encoded functions preserves their original non-zero evaluations at ξ^i . We define the encoding function as follows:

- $\text{Enc}_L(\xi, k \in [\mu], \mathbf{a} \in \mathbb{Z}_q^{\leq 2n}) \rightarrow F_{L,k} \in \mathbb{Z}_q[X]$
Construct degree D polynomial $F_{L,k}(X)$ such that:

$$F_{L,k}(\xi^i) = \begin{cases} \mathbf{a}[j - 2n(k - 1)], & \text{if } i = (3j - 2)t \text{ for } 2n(k - 1) < j \leq 2nk \\ 0, & \text{otherwise} \end{cases}$$

- $\text{Enc}_R(\xi, k \in [\mu], \mathbf{a} \in \mathbb{Z}_q^{\leq 2n}) \rightarrow F_{R,k} \in \mathbb{Z}_q[X]$
Construct degree D polynomial $F_{R,k}(X)$ such that:

$$F_{R,k}(\xi^i) = \begin{cases} \mathbf{a}[j - 2n(k - 1)], & \text{if } i = (3j - 1)t \text{ for } 2n(k - 1) < j \leq 2nk \\ 0, & \text{otherwise} \end{cases}$$

- $\text{Enc}_P(\xi, k \in \{0, \dots, \mu + \nu + 1\}, \mathbf{a} \in \mathbb{Z}_q^{\leq 2n}) \rightarrow F_{P,k} \in \mathbb{Z}_q[X]$
Construct degree D polynomial $F_{P,k}(X)$ such that:

$$F_{P,k}(\xi^i) = \begin{cases} \mathbf{a}[j - 2nk], & \text{if } i = 3jt \text{ for } 2nk < j \leq 2n(k + 1) \\ 0, & \text{otherwise} \end{cases}$$

Using the encoding function, we define the commitment Com_2 based on the homomorphic polynomial commitment Com_{PC} . The $\text{ck}_{\text{type},k}$ consists of four tuples: $(\text{ck}_{\text{PC}}, \xi, \text{type}, k)$. We describe the commitment Com_2 for message \mathbf{a} as follows:

- $\text{Com}_2(\text{ck}_{\text{type},k}, \mathbf{a} = (\mathbf{a}^{(1)}, \mathbf{a}^{(2)}) \in \mathbb{Z}_q^{4n}) \rightarrow \mathbf{A}$
 1. $\text{Enc}_{\text{type}}(\xi, k, \mathbf{a}^{(i)}) \rightarrow F_{\text{type},k}^{(i)}$ for $i \in \{1, 2\}$
 2. $\text{Com}_{\text{PC}}(\text{ck}_{\text{PC}}, F_{\text{type},k}^{(i)}) \rightarrow \mathbf{A}^{(i)}$ for $i \in \{1, 2\}$
 3. Output $\mathbf{A} = (\mathbf{A}^{(1)}, \mathbf{A}^{(2)})$

Consistency Proof. Recall that the goal of AggMEC is to prove the relations Eq. (3) and Eq. (4), (5). Since the latter two relations can be ensured by

Algorithm 2 AggMEC

-
- AggMEC($\mathcal{P}_{\text{Pub}}, \text{ck}_k, (\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k, x_k); (\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k)$)
 $\text{ck}_k = (\text{ck}_{L,k}, \text{ck}_{R,k}, \text{ck}_{P,k})$, each ck_k contains ck_{PC}
- 1: \mathcal{P} and \mathcal{V} set $A^{(i)} = \sum_{k=1}^{\mu} (\mathbf{L}_k^{(i)} + \mathbf{R}_k^{(i)}) + \sum_{k=0}^{\mu+\nu} \mathbf{P}_k^{(i)}$ for $i \in \{1, 2\}$
 - 2: \mathcal{P} sets $a^{(i)} = \sum_{k=1}^{\mu} (F_{L,k}^{(i)} + F_{R,k}^{(i)}) + \sum_{k=0}^{\mu+\nu} F_{P,k}^{(i)}$ for $i \in \{1, 2\}$:
 $F_{L,k}^{(i)} = \text{Enc}_L(\xi, k, \mathbf{L}_k^{(i)})$, $F_{R,k}^{(i)} = \text{Enc}_R(\xi, k, \mathbf{R}_k^{(i)})$, $F_{P,k}^{(i)} = \text{Enc}_P(\xi, k, \mathbf{P}_k^{(i)})$
 - 3: \mathcal{P} construct wire polynomials $\{w^{(i)}(X)\}_{i=1}^2$ from execution table with public in/out \mathcal{P}_{Pub} and then computes $W^{(1)}, W^{(2)}, Q^{(1)}, Q^{(2)}$ and sends them to \mathcal{V} :
 $q^{(i)}(X) = \frac{w^{(i)}(X) - a^{(i)}(X)}{X^d - 1}$, $W^{(i)} = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, w^{(i)})$, $Q^{(i)} = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, q^{(i)})$
 - 4: \mathcal{V} chooses $z, \rho \xleftarrow{\$} \mathbb{Z}_q$ and sends them to \mathcal{P} .
 - 5: \mathcal{P} and \mathcal{V} compute:
 $V = \sum_{i=1}^2 (\sum_{k=1}^{\mu} ([\rho^{4k-2-i}] \mathbf{L}_k^{(i)} + [\rho^{4k-i}] \mathbf{R}_k^{(i)}) + \rho^{4\mu} (\sum_{k=0}^{\mu+\nu} [\rho^{2k-1+i}] \mathbf{P}_k^{(i)}))$
 - 6: \mathcal{P} computes $F_V(X)$:
 $F_V = \sum_{i=1}^2 (\sum_{k=1}^{\mu} (\rho^{4k-2-i} F_{L,k}^{(i)} + \rho^{4k-i} F_{R,k}^{(i)}) + \rho^{4\mu} (\sum_{k=0}^{\mu+\nu} \rho^{2k-1+i} F_{P,k}^{(i)}))$
 - 7: \mathcal{P} sends $s, t^{(1)}, t^{(2)}, r^{(1)}, r^{(2)}$ to \mathcal{V} : $s = F_V(z)$, $t^{(i)} = q^{(i)}(z)$, $r^{(i)} = w^{(i)}(z)$
 - 8: \mathcal{V} chooses $\tau \xleftarrow{\$} \mathbb{Z}_q$ and sends them to \mathcal{P} .
 - 9: \mathcal{P} and \mathcal{V} set $P = V + \sum_{i=1}^2 ([\tau^i] A^{(i)} + [\tau^{2+i}] Q^{(i)})$ and
 $y = s + \sum_{i=1}^2 (\tau^i (r^{(i)} - t^{(i)} (z^d - 1)) + \tau^{2+i} t^{(i)})$
 - 10: \mathcal{P} set $F_P = F_V + \sum_{i=1}^2 (\tau^i a^{(i)} + \tau^{2+i} q^{(i)})$
 - 11: \mathcal{P} sets wire polynomials $\{w^{(i)}\}_{i=0}^{M-1} \in \mathbb{Z}_q[X]$ containing $w^{(1)}$ and $w^{(2)}$.
 - 12: \mathcal{P} and \mathcal{V} set run $\text{Eval}(\text{ck}_{\text{PC}}, P, z, y; F_P)$ and $\text{Eval}(\text{ck}_{\text{PC}}, W^{(i)}, z, r^{(i)}; w^{(i)})$ for $i \in \{1, 2\}$
 - 13: \mathcal{P} and \mathcal{V} run $\text{Plonkish}_{\text{Eval}}(\text{ck}_{\text{PC}}; \{s_i(X), g_i(X_0, \dots, X_{M-1})\}_{i=0}^{N_g-1}, \{r_i\}_{i=0}^{M-1}; \{w^{(i)}\}_{i=0}^{M-1})$
-

$\text{Plonkish}_{\text{Eval}}$, we now focus on checking Eq. (3) and the consistency between $\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k$ and the wire polynomial of the execution trace. First, to ensure consistency of $\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k$, we first merge every commitment $\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k$ to one commitment A , whose message polynomial is the sum of encoding polynomials, $a(X) = \sum F_{\text{type},k}(X)$. Then the difference polynomial $w(X) - a(X)$ is divided by $X^d - 1$ due to $w(\xi^i) - a(\xi^i) = 0$ for all i . The verifier can check it by using Eval after receiving a commitment of the wire polynomial $w(X)$. Furthermore, we employed V and F_V to ensure Eq. (3), *i.e.*, each $\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k$ corresponds to the commitment of $\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k$, respectively. Through the randomness ρ chosen by the verifier, these commitments and the encoding polynomials of $\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k$ can be merged into V and F_V , respectively. Finally, we aggregate these consistency checks by constructing P and F_P with a randomness τ from the verifier. Then the verifier can check them all at once by opening them at another random point z , convincing these relations with negligible soundness error by the Schwartz-Zippel Lemma. Using these techniques, we present the AggMEC protocol in Algorithm 2.

Theorem 3. *Assume that a polynomial commitment scheme $\text{PCS} = (\text{Gen}, \text{Comp}_{\text{PC}}, \text{Eval})$ satisfies all properties of Definition 4 and the homomorphic property. Then,*

AggMEC in Algorithm 2 has perfect completeness and computational witness-extended-emulation.

Due to the space limit, the proof of Theorem 3 is presented in Appendix D.

3.4 Cougar: Cubic Root Verifier IPA

Using Sublinear Verifier PCS. To reduce verifier complexity, we adopt the sublinear verifier PCS Leopard_{PC} [38] from the above construction. This achieves a cubic root complexity for the verifier. Full details of Leopard_{PC} are provided in Appendix G.

Complexity Analysis. In this paragraph, we provide a complexity analysis of Cougar divided into Protocol.Row, Protocol.Col, and AggMEC.

1. Row-reduction, Algorithm 1

- **[Prover Cost]:** For commitments L, R and \hat{P} at i -th round, \mathcal{P} computes $O(\frac{N}{2^i}) \mathbb{G}_p$ operations and $O(n \log_2 m) \mathbb{G}_q$ operations. For updating \hat{G}, \hat{H} and $\hat{a}, \hat{b}, \hat{c}$ at i -th round, \mathcal{P} computes $O(\frac{m}{2^i}) \mathbb{G}_p$ operation and $O(n \cdot \frac{m}{2^i}) \mathbb{Z}_p$ respectively. Then, the total prover cost is $O(N) \mathbb{Z}_p$ and $O(N) \mathbb{G}_p$ operations.
- **[Verifier Cost]:** For updating \hat{G}, \hat{H} and \hat{c} at i -th round, \mathcal{V} computes $O(\frac{m}{2^i}) \mathbb{G}_p$ operation and 2 multiplication in \mathbb{Z}_p . Then, the total verifier cost is $O(m) \mathbb{G}_p$ and $O(\log_2 m) \mathbb{Z}_p$ operations.
- **[Communication Cost]:** For each round, \mathcal{P} sends $L, R, \hat{P}, c_L,$ and c_R . Then, the total communication cost is $3 \log_2 m |\mathbb{G}_q| + 2 \log_2 m |\mathbb{Z}_p|$.

2. Column-reduction, Algorithm 3

- **[Prover Cost]:** For a inner product c_L and c_R at i -th round, the prover computes $O(\frac{n}{2^i}) \mathbb{Z}_p$ operations. For updating $\hat{P}, \hat{a}, \hat{b},$ and \hat{c} at i -th round, \mathcal{P} computes $O(\frac{n}{2^i}) \mathbb{G}_p$ and \mathbb{Z}_p operations, $O(\frac{n}{2^i} \log_2 m) \mathbb{G}_q$ operations. Then, the total prover cost is $O(n \log_2 m) \mathbb{G}_q$ operations.
- **[Verifier Cost]:** For updating \hat{c} at each round except the final round, \mathcal{V} computes 2 multiplication in \mathbb{Z}_p . In the final round, \mathcal{V} computes one \mathbb{Z}_p operation for verification. Then, the total verifier cost is $O(\log_2 n) \mathbb{Z}_p$ operations.
- **[Communication Cost]:** For each round, the prover sends $\hat{P}, c_L,$ and c_R . The total communication cost is $\log_2 n |\mathbb{G}_q| + 2 \log_2 n |\mathbb{Z}_p|$.

3. Aggregated MEC, Algorithm 2

- **[Prover Cost]:** From line 1 to 11, \mathcal{P} treats at most $\log_2 N$ polynomials of degree D . Then, \mathcal{P} computes $O(D \log_2 N) = O(n \log_2 N)$ operations, including $\mathbb{Z}_p, \mathbb{G}_p,$ and \mathbb{G}_q . And the cost of Eval and Plonkish_{Eval} is $O(\|\mathcal{P}_{\text{Eval}}(D)\|)$. Then, total prover cost is $O(n \log_2 N + \|\mathcal{P}_{\text{Eval}}(D)\|)$.
- **[Verifier Cost]:** From line 1 to 11, \mathcal{V} computes $O(\log_2 N) \mathbb{G}_q$ operations. And the cost of Eval and Plonkish_{Eval} is $O(\|\mathcal{V}_{\text{Eval}}(D)\|)$. Then the total verifier cost is $O(\log_2 N + \|\mathcal{V}_{\text{Eval}}(D)\|)$.

- [Communication Cost]: From line 1 to 11, \mathcal{P} sends \mathcal{V} 4 \mathbb{G}_q elements and 5 field elements. Additionally, for Eval and Plonkish the prover sends $O(\|II_{\text{Eval}}(D)\|)$. Then total communication cost is $O(\|II_{\text{Eval}}(D)\|)$

Cubic Root Verifier IPA from Parameter Setting. Let $N = mn$ be the length of the witness vectors with $n = \sqrt[3]{N^2}$ and $m = \sqrt[3]{N}$. Since **Leopard** features $(\|\mathcal{P}_{\text{Eval}}(D)\|, \|\mathcal{V}_{\text{Eval}}(D)\|, \|II_{\text{Eval}}(D)\|) = (O(D), O(\sqrt{D}), O(\log_2 D))$, we conclude that the **Cougar** features $O(N)$ prover cost, $O(\log_2 N)$ communication cost and $O(\sqrt[3]{N}\sqrt{\log_2 N})$ verifier cost, which is the cubic root of N .

Theorem 4. *Cougar is an IPA, which features $O(\log_2 N)$ communication cost, $O(N)$ prover cost and $O(\sqrt[3]{N}\sqrt{\log_2 N})$ verifier cost where N is length of witness. Cougar provides perfect completeness and computational witness extended emulation under the DL assumption.*

Proof. The prover, verifier, and communication costs can be checked in the above analysis. By Theorem 1, Theorem 2, and Theorem 3 and the soundness of **Leopard** under the DL assumption [38], **Cougar** satisfies perfect completeness and computational witness-extended-emulation under the DL assumption. \square

Efficiency Analysis. We implemented **Cougar** in Rust, employing the BN254 and Grumpkin curves as a half-pairing cycles. We run **Cougar** from $N = 2^{10}$ to $N = 2^{20}$ and compared it with previous IPAs from the DL assumption, including **BulletProofs** and **Leopard**. Under our experimental results, the measured time spent by verifier in $N = 2^{20}$ is 0.346s, which is 50 \times speed-up from **BulletProofs**. We also emphasize that as N enlarges, the growth of the verifier time of **Cougar** is much slower than those of **BulletProofs** and **Leopard**. In addition, we checked that the (estimated) verifier time of **Cougar** becomes faster than that of **Leopard** for a larger N , though the concrete verifier time of **Cougar** is slower than **Leopard** in our experiments. For more details about implementation, we recommend the reader refer to the Appendix I and our source code in github¹.

¹ <https://github.com/Cryptology-Algorithm-Lab/Cougar>

References

1. M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. *J. Cryptol.*, 29(2):363–421, 2016.
2. A. Arun, C. Ganesh, S. V. Lokam, T. Mopuri, and S. Sridhar. Dew: Transparent constant-sized zkSNARKs. In *Public-Key Cryptography – PKC 2023*, pages 542–571. Springer, 2023.
3. C. Baum, J. Bootle, A. Cerulli, R. del Pino, J. Groth, and V. Lyubashevsky. Sublinear lattice-based zero-knowledge arguments for arithmetic circuits. In *Advances in Cryptology – CRYPTO 2018*, pages 669–699. Springer, 2018.
4. C. Baum and A. Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In *Public-Key Cryptography – PKC 2020*, pages 495–526. Springer, 2020.
5. S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, 2012.
6. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable zero knowledge with no trusted setup. In *Annual International Cryptology Conference*, pages 701–732. Springer, 2019.
7. E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO 2013*, volume 8043 of *LNCS*, pages 90–108. Springer, 2013.
8. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. Aurora: Transparent succinct arguments for r1cs. In *EUROCRYPT 2019*, volume 11476 of *LNCS*, pages 103–128. Springer, 2019.
9. E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security 2014*, pages 781–796, 2014.
10. D. Boneh, J. Drake, B. Fisch, and A. Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. *Cryptology ePrint Archive*, 2020.
11. J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 327–357. Springer, 2016.
12. J. Bootle, A. Cerulli, J. Groth, S. Jakobsen, and M. Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *ASIACRYPT 2018, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *LNCS*, pages 595–626. Springer, 2018.
13. J. Bootle, A. Chiesa, and J. Groth. Linear-time arguments with sublinear verification from tensor codes. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part II 18*, pages 19–46. Springer, 2020.
14. J. Bootle, V. Lyubashevsky, N. K. Nguyen, and G. Seiler. A non-PCP approach to succinct quantum-safe zero-knowledge. In *Advances in Cryptology – CRYPTO 2020*, pages 441–469. Springer, 2020.
15. S. Bowe, J. Grigg, and D. Hopwood. Recursive proof composition without a trusted setup. *Cryptology ePrint Archive*, Report 2019/1021, 2019.
16. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy 2018*, pages 315–334. IEEE Computer Society, 2018.

17. B. Bünz and B. Fisch. Multilinear schwartz-zippel mod n and lattice-based succinct arguments. In *Theory of Cryptography Conference*, pages 394–423. Springer, 2023.
18. B. Bünz, B. Fisch, and A. Szeponiec. Transparent snarks from dark compilers. In *EUROCRYPT 2020, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, 2020.
19. B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely. Proofs for inner pairing products and applications. In *ASIACRYPT 2021, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *LNCS*, pages 65–97. Springer, 2021.
20. M. Burmester, Y. Desmedt, and T. Beth. Efficient zero-knowledge identification scheme for smart cards. *Comput. J.*, 35(1):21–29, 1992.
21. B. Chen, B. Bünz, D. Boneh, and Z. Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 499–530. Springer, 2023.
22. H. Chung, K. Han, C. Ju, M. Kim, and J. H. Seo. Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger. *IEEE Access*, 10:42067–42082, 2022.
23. V. Daza, C. Ràfols, and A. Zacharakis. Updateable inner product argument with logarithmic verifier and applications. In *PKC 2020*, volume 12110 of *LNCS*, pages 527–557. Springer, 2020.
24. M. Dutta, C. Ganesh, and N. Jawalkar. Succinct verification of compressed sigma protocols in the updatable srs setting. In *IACR International Conference on Public-Key Cryptography*, pages 305–336. Springer, 2024.
25. U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptol.*, 1(2):77–94, 1988.
26. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO 1986*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.
27. Filecoin. blstrs, 2024. <https://github.com/filecoin-project/blstrs>.
28. A. Gabizon and Z. J. Williamson. Proposal: The turbo-plonk program syntax for specifying snark programs, 2020.
29. A. Gabizon, Z. J. Williamson, and O. Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. <https://eprint.iacr.org/2019/953.pdf>.
30. S. Gao, Z. Peng, F. Tan, Y. Zheng, and B. Xiao. Symmeproof: Compact zero-knowledge argument for blockchain confidential transactions. *IEEE Transactions on Dependable and Secure Computing*, 2022.
31. S. Gao, Z. Peng, F. Tan, Y. Zheng, and B. Xiao. Symmeproof: Compact zero-knowledge argument for blockchain confidential transactions. *IEEE Transactions on Dependable and Secure Computing*, 20(3):2289–2301, 2023.
32. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.
33. A. Golovnev, J. Lee, S. Setty, J. Thaler, and R. S. Wahby. Brakedown: Linear-time and field-agnostic snarks for r1cs. In *Annual International Cryptology Conference*, pages 193–226. Springer, 2023.
34. J. Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO 2009*, volume 5677 of *LNCS*, pages 192–208. Springer, 2009.
35. J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016.

36. A. Jivanyan. Lelantus: Towards confidentiality and anonymity of blockchain transactions from standard assumptions. Cryptology ePrint Archive, Report 2019/373, 2019.
37. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, 2010.
38. S. Kim, G. Lee, H. Lee, and J. H. Seo. Leopard: Sublinear verifier inner product argument under discrete logarithm assumption. *IEEE Transactions on Information Forensics and Security*, 18:5332–5344, 2023.
39. S. Kim, H. Lee, and J. H. Seo. Efficient zero-knowledge arguments in discrete logarithm setting: Sublogarithmic proof or sublinear verifier. In *ASIACRYPT 2022, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II*, volume 13792 of *LNCS*, pages 403–433. Springer, 2022.
40. H. Lee and J. H. Seo. TENET: sublogarithmic proof and sublinear verifier inner product argument without a trusted setup. In *Advances in Information and Computer Security - 18th International Workshop on Security, IWSEC 2023, Yokohama, Japan, August 29-31, 2023, Proceedings*, volume 14128 of *Lecture Notes in Computer Science*, pages 214–234. Springer, 2023.
41. J. Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, 2021.
42. OECD. Emerging privacy-enhancing technologies. *OECD Digital Economy Papers*, (351), 2023.
43. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.
44. N. Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250, 1980.
45. PSE. halo2curves, 2024. <https://github.com/privacy-scaling-explorations/halo2curves>.
46. PSE. halo2_proofs, 2024. https://github.com/zcash/halo2/tree/main/halo2_proofs.
47. J. Renes, C. Costello, and L. Batina. Complete addition formulas for prime order elliptic curves. In M. Fischlin and J. Coron, editors, *EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 403–428. Springer, 2016.
48. E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy 2014*, pages 459–474. IEEE, 2014.
49. J. H. Seo. Round-efficient sub-linear zero-knowledge arguments for linear algebra. In *PKC 2011*, volume 6571 of *LNCS*, pages 387–402. Springer, 2011.
50. S. Setty, S. Angel, T. Gupta, and J. Lee. Proving the correct execution of concurrent services in zero-knowledge. In *OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 339–356. USENIX Association, 2018.
51. R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zkSNARKs without trusted setup. In *IEEE Symposium on Security and Privacy 2018*, pages 926–943. IEEE, 2018.
52. T. Xie, Y. Zhang, and D. Song. Orion: Zero knowledge proof with linear prover time. In *Annual International Cryptology Conference*, pages 299–328. Springer, 2022.
53. T. H. Yuen, M. F. Esgin, J. K. Liu, M. H. Au, and Z. Ding. Dualring: generic construction of ring signatures with efficient instantiations. In *Annual International Cryptology Conference*, pages 251–281. Springer, 2021.

54. A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, and M. Simkin. Caulk: Lookup arguments in sublinear time. In *CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 3121–3134. ACM, 2022.
55. zcash. The halo2 book, 2022. <https://zcash.github.io/halo2/>.
56. J. Zhang, M. Su, X. Liu, and G. Wang. Springproofs: Efficient inner product arguments for vectors of arbitrary length. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 67–67. IEEE Computer Society, 2023.

A Deferred Protocol Description

In this section, we display protocol descriptions for Protocol.Col.

Algorithm 3 Protocol.Col

- Protocol.Col($G, H, (\text{ck}_{P,k+\mu})_{k=s}^\nu, P, c, st_V; \mathbf{a}, \mathbf{b}, st_P$)
- 1: **if** $n = 1$, base case $s = \nu$ **then**:
 - 2: \mathcal{P} sends a and b to \mathcal{V}
 - 3: \mathcal{V} checks $c \stackrel{?}{=} a \cdot b$ and set $\mathbf{P}_{\text{Pub}} = [a]G \parallel [b]H \in \mathbb{Z}_q^4$
 - 4: \mathcal{P} and \mathcal{V} run $\text{AggMEC}(\mathbf{P}_{\text{Pub}}, st_V; st_P)$
 - 5: **else**
 - 6: **if** $st_P = \perp$ and $st_V = \perp$ **then**
 - 7: \mathcal{P} sets $\mathbf{P} = [a]G \parallel [b]H$ and adds (\mathbf{P}) into the bottom row of st_P
 - 7: \mathcal{V} adds $(\text{ck}_{P,\mu}, \mathbf{P}, \cdot)$ into the bottom row of st_V .
 - 8: **else**
 - 9: \mathcal{P} refers \mathbf{P} in the bottom row of st_P
 - 10: **end if**
 - Set $\hat{n} = \frac{n}{2}$ and $\mathbf{a} = \mathbf{a}_L \parallel \mathbf{a}_R$, $\mathbf{b} = \mathbf{b}_L \parallel \mathbf{b}_R$, $\mathbf{P} = \mathbf{P}^{(q_1)} \parallel \mathbf{P}^{(q_2)} \parallel \mathbf{P}^{(q_3)} \parallel \mathbf{P}^{(q_4)}$
 - 11: \mathcal{P} computes c_L and c_R and sends them to \mathcal{V} :
 $c_L = \langle \mathbf{a}_L, \mathbf{b}_R \rangle \in \mathbb{Z}_p$, $c_R = \langle \mathbf{a}_R, \mathbf{b}_L \rangle \in \mathbb{Z}_p$.
 - 12: \mathcal{V} chooses $x \xleftarrow{\$} \mathbb{Z}_p^*$ and returns it to \mathcal{P}
 - 13: \mathcal{P} computes $\hat{\mathbf{P}}$ and sends it to \mathcal{V} :
 $\hat{\mathbf{P}} = (\mathbf{P}^{(q_1)} + [x]\mathbf{P}^{(q_2)} \parallel \mathbf{P}^{(q_3)} + [x^{-1}]\mathbf{P}^{(q_4)}) \in \mathbb{G}_p^{2\hat{n}}$, $\hat{\mathbf{P}} = \text{Com}_2(\text{ck}_{P,\mu+s}, \hat{\mathbf{P}}) \in \mathbb{G}_q$
 - 14: Both \mathcal{P} and \mathcal{V} compute $\hat{c} = x^{-1}c_L + c + xc_R \in \mathbb{Z}_p$
 - 15: Additionally, \mathcal{P} computes $\hat{\mathbf{a}} = \mathbf{a}_L + x\mathbf{a}_R$, $\hat{\mathbf{b}} = \mathbf{b}_L + x^{-1}\mathbf{b}_R \in \mathbb{Z}_p^{\hat{n}}$.
 - 16: \mathcal{V} adds $(\text{ck}_{P,\mu+s}, \hat{\mathbf{P}}, x)$ into the bottom row of st_V .
 - 17: \mathcal{P} adds $(\hat{\mathbf{P}})$ into the bottom row of st_P .
 - 18: Both \mathcal{P} and \mathcal{V} run $\text{Protocol.Col}(G, H, (\text{ck}_{P,k+\mu})_{k=s+1}^\nu, \hat{\mathbf{P}}, \hat{c}, st_V; \hat{\mathbf{a}}, \hat{\mathbf{b}}, st_P)$
 - 19: **end if**
-

B Proof of Theorem 1

Proof. (Completeness) For a base case $m = 1$, the completeness is held by the completeness of Protocol.Col and AggMEC. Let us consider the case $m > 1$. In this case, we show that if the input $(\mathbf{G}, \mathbf{H}, \text{ck}_{P,s}, P, c; \mathbf{a}, \mathbf{b})$ belongs to $\mathcal{R}_{\text{GenPT4}}^{m,n}$:

then the updated input $(\widehat{\mathbf{G}}, \widehat{\mathbf{H}}, \text{ck}_{P,s+1}, \widehat{\mathbf{P}}, \widehat{c}; \widehat{\mathbf{a}}, \widehat{\mathbf{b}})$ belongs to $\mathcal{R}_{\text{GenPT4}}^{m/2,n}$. Following the \mathcal{P} algorithm, we get the following equations:

$$\begin{aligned} \widehat{c} &= x^{-1}c_L + c + xc_R = \langle \mathbf{a}_L, x^{-1}\mathbf{b}_R \rangle + \langle \mathbf{a}, \mathbf{b} \rangle + \langle x\mathbf{a}_R, \mathbf{b}_L \rangle = \langle \widehat{\mathbf{a}}, \widehat{\mathbf{b}} \rangle \\ \widehat{\mathbf{P}} &= [x^{-1}]\mathbf{L} + \mathbf{P} + [x]\mathbf{R} \\ &= x^{-1}[\mathbf{a}_L]\mathbf{G}_R \parallel [x^{-1}\mathbf{b}_R]\mathbf{H}_L + [\mathbf{a}]\mathbf{G} \parallel [\mathbf{b}]\mathbf{H} + [x\mathbf{a}_R]\mathbf{G}_L \parallel x[\mathbf{b}_L]\mathbf{H}_R \\ &= \widehat{\mathbf{a}}\widehat{\mathbf{G}} \parallel \widehat{\mathbf{b}}\widehat{\mathbf{H}} \\ \widehat{\mathbf{P}} &= \text{Com}_2(\text{ck}_{P,s+1}, \widehat{\mathbf{P}}) = \text{Com}_{\text{TC}}((\widehat{\mathbf{G}} \parallel \widehat{\mathbf{H}}, \text{ck}_{P,s+1}), \widehat{\mathbf{a}} \parallel \widehat{\mathbf{b}}) \end{aligned}$$

Therefore, we can conclude that updated input $(\widehat{\mathbf{G}}, \widehat{\mathbf{H}}, \text{ck}_{P,s+1}, \widehat{\mathbf{P}}, \widehat{c}; \widehat{\mathbf{a}}, \widehat{\mathbf{b}})$ belongs to $\mathcal{R}_{\text{GenPT4}}^{m/2,n}$.

(Witness-extended-emulation) For the computational witness-extended emulation, we construct an expected polynomial time extractor \mathcal{E}_{Row} whose goal is to extract a witness by using a polynomially bounded tree of accepting transcripts. To this end, we utilize the general forking lemma [11], which is stated as follows:

Theorem 5 (General Forking Lemma). *Let $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ be a $(2\mu + 1)$ -move, public coin interactive protocol with μ challenges x_1, \dots, x_μ in sequence. Let $n_i \geq 1$ for $i \in [\mu]$. Consider an (n_1, \dots, n_μ) -tree of accepting transcripts with challenges in the following format. The tree has $N = \prod_{i=1}^\mu n_i$ leaves with depth μ . The root of the tree is labeled with the statement. Each node of depth i has exactly n_i children, each labeled with a distinct value of the i -th challenge x_i .*

Let \mathcal{E} be a witness extractor that succeeds with probability $1 - \text{negl}(\lambda)$ for some negligible function $\text{negl}(\lambda)$ in extracting a witness from an (n_1, \dots, n_μ) -tree of accepting transcripts in probabilistic polynomial time. Assume that $\prod_{i=1}^\mu n_i$ is bounded above by a polynomial in the security parameter λ . Then, $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ has witness-extended emulation.

\mathcal{E}_{Row} takes public inputs $(\mathbf{G}, \mathbf{H}, (\text{ck}_k)_{k=1}^\mu, \text{ck}_{\text{Col}}, \mathbf{P}, c, st_V; \mathbf{a}, \mathbf{b}, st_P)$. By premise, \mathcal{E}_{Row} exploits two PPT extractors \mathcal{E}_{Col} and \mathcal{E}_{MEC} , that extract witness $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^{1 \times n}$ and st_P respectively. Note that st_P consists of tuples of commitments $(\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k)$, which satisfies the Eq. (3) and (4).

We show how to extract witness \mathbf{a}, \mathbf{b} from accepting transcripts. By the general forking lemma, it is sufficient to construct an extractor \mathcal{E}_{Row} that extracts a witness from a suitable tree of accepting transcripts in probabilistic polynomial time. We begin with $(\underbrace{4, \dots, 4}_{\log_2 m})$ -tree of accepting transcripts. Since the number

of leaves of the tree is polynomially bound, $4^{\log_2 m}$, we can apply the general forking lemma.

First, for the base case $m = 1$, the extracted witness \mathbf{a}, \mathbf{b} from \mathcal{E}_{Col} satisfies the desire condition so that \mathcal{E}_{Row} outputs the \mathbf{a}, \mathbf{b} in polynomial time.

In the case $m > 1$, we construct extractor \mathcal{E}_{Row} by inductively extraction. That is, retrieves s -round witness $\mathbf{a}^{(s)}, \mathbf{b}^{(s)} \in \mathbb{Z}_p^{m/2^s \times n}$ from next steps $\mathbf{a}^{(s+1)}, \mathbf{b}^{(s+1)} \in \mathbb{Z}_p^{m/2^{s+1} \times n}$ recursively.

First, \mathcal{E}_{Row} run \mathcal{E}_{Col} and get extracted witnesses $\mathbf{a}^{(\mu)}, \mathbf{b}^{(\mu)} \in \mathbb{Z}_p^{1 \times n}$, which is valid witness for the relation $\mathcal{R}_{\text{GenPT4}}^{1,n}$. Now, we assume that $\widehat{\mathbf{a}}_i, \widehat{\mathbf{b}}_i \in \mathbb{Z}_p^{m/2^{s+1} \times n}$ is valid witness of instance $(\widehat{\mathbf{G}}_i, \widehat{\mathbf{H}}_i, \widehat{\mathbf{P}}_i, \widehat{c}_i)$, that are updated instance using challenge x_i for the relation $\mathcal{R}_{\text{GenPT4}}^{m/2^{s+1}, n}$. From the tree of accepting transcript, we can get 4 instance-witness pairs: $(\widehat{\mathbf{G}}_i, \widehat{\mathbf{H}}_i, \widehat{\mathbf{P}}_i, \widehat{c}_i; \widehat{\mathbf{a}}_i, \widehat{\mathbf{b}}_i)$. Furthermore, the \mathcal{E}_{Row} can get s -round prover's commitments L, R, P, \widehat{P} and their messages $\mathbf{L}, \mathbf{R}, \mathbf{P}, \widehat{\mathbf{P}}$ from s -round transcripts and \mathcal{E}_{MEC} respectively. From 3 distinct tuples, \mathcal{E}_{Row} can construct the following linear system:

$$\begin{bmatrix} x_1^{-1} & 1 & x_1 \\ x_2^{-1} & 1 & x_2 \\ x_3^{-1} & 1 & x_3 \end{bmatrix} \begin{bmatrix} \mathbf{L} \\ \mathbf{P} \\ \mathbf{R} \end{bmatrix} = \begin{bmatrix} \widehat{\mathbf{P}}_1 \\ \widehat{\mathbf{P}}_2 \\ \widehat{\mathbf{P}}_3 \end{bmatrix} = \begin{bmatrix} [\widehat{\mathbf{a}}_1] \widehat{\mathbf{G}}_1 \parallel [\widehat{\mathbf{b}}_1] \widehat{\mathbf{H}}_1 \\ [\widehat{\mathbf{a}}_2] \widehat{\mathbf{G}}_2 \parallel [\widehat{\mathbf{b}}_2] \widehat{\mathbf{H}}_2 \\ [\widehat{\mathbf{a}}_3] \widehat{\mathbf{G}}_3 \parallel [\widehat{\mathbf{b}}_3] \widehat{\mathbf{H}}_3 \end{bmatrix} \quad (6)$$

Since the right-hand side of Eq. (6) is decomposed by $\widehat{\mathbf{G}} = \mathbf{G}_L + [x^{-1}] \mathbf{G}_R$ and $\widehat{\mathbf{H}} = \mathbf{H}_L + [x] \mathbf{H}_R$ and each \mathbf{G} and \mathbf{H} are not effected by challenge x , \mathcal{E}_{Row} can get represented vectors $\mathbf{l}, \mathbf{r}, \mathbf{p} \in \mathbb{Z}_p^{m/2^s \times 2n}$ of $\mathbf{L}, \mathbf{R}, \mathbf{P} \in \mathbb{G}^{2n}$ under base $\mathbf{G} \parallel \mathbf{H}$. Let \mathcal{E}_{Row} parse $\mathbf{l}, \mathbf{r}, \mathbf{p}$ to 4 segments $\mathbf{l}^{(t)}, \mathbf{p}^{(t)}, \mathbf{r}^{(t)} \in \mathbb{Z}_p^{m/2^s \times n/2}$ where $t \in [4]$. Let the representation vectors put on Eq. (6). Then

$$[x_i^{-1} \mathbf{l}^{(1)} + \mathbf{p}^{(1)} + x_i \mathbf{r}^{(1)}] \mathbf{G}_L = [\widehat{\mathbf{a}}] \mathbf{G}_L \quad (7)$$

$$[x_i^{-1} \mathbf{l}^{(2)} + \mathbf{p}^{(2)} + x_i \mathbf{r}^{(2)}] \mathbf{G}_R = [x_i^{-1} \widehat{\mathbf{a}}] \mathbf{G}_R \quad (8)$$

$$[x_i^{-1} \mathbf{l}^{(3)} + \mathbf{p}^{(3)} + x_i \mathbf{r}^{(3)}] \mathbf{H}_L = [\widehat{\mathbf{b}}] \mathbf{H}_L \quad (9)$$

$$[x_i^{-1} \mathbf{l}^{(4)} + \mathbf{p}^{(4)} + x_i \mathbf{r}^{(4)}] \mathbf{H}_R = [x_i \widehat{\mathbf{b}}] \mathbf{H}_R \quad (10)$$

By DL assumption on \mathbb{G} , the representation vectors of both side should be equivalent. From Eq.(7), Eq.(8) and Eq.(9), Eq.(10), we get

$$-\mathbf{l}^{(1)} + x_i(\mathbf{l}^{(2)} - \mathbf{p}^{(1)}) + x_i^2(\mathbf{p}^{(2)} - \mathbf{r}^{(1)}) + x_i^3 \mathbf{r}^{(2)} = 0$$

$$-\mathbf{l}^{(4)} + x_i(\mathbf{l}^{(3)} - \mathbf{p}^{(4)}) + x_i^2(\mathbf{p}^{(3)} - \mathbf{r}^{(4)}) + x_i^3 \mathbf{r}^{(3)} = 0$$

for x_1, \dots, x_4 . Then each terms of x_i^k should be zero. Let $\mathbf{p}^{(i)}$ denote $\tilde{\mathbf{a}}_L = \mathbf{p}^{(1)}$, $\tilde{\mathbf{a}}_R = \mathbf{p}^{(2)}$, $\tilde{\mathbf{b}}_L = \mathbf{p}^{(3)}$, $\tilde{\mathbf{b}}_R = \mathbf{p}^{(4)}$. Then, we can obtain the following equation:

$$\begin{aligned} x_i^{-1} c_L + c + x_i c_R &= \widehat{c} = \langle \widehat{\mathbf{a}}, \widehat{\mathbf{b}} \rangle \\ &= x_i^{-1} \langle \tilde{\mathbf{a}}_L, \tilde{\mathbf{b}}_R \rangle + \langle \tilde{\mathbf{a}}, \tilde{\mathbf{b}} \rangle + x_i \langle \tilde{\mathbf{a}}_R, \tilde{\mathbf{b}}_L \rangle \end{aligned} \quad (11)$$

Similarly, x_1, \dots, x_4 guarantees $c = \langle \tilde{\mathbf{a}}, \tilde{\mathbf{b}} \rangle$. Therefore, the extracted witnesses $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ is valid witness for the relation $\mathcal{R}_{\text{GenPT4}}^{m/2^s, n}$. By inductively retrieving process and general forking lemma, \mathcal{E}_{Row} can extract witness vectors \mathbf{a} and \mathbf{b} . \square

C Proof of Theorem 2

Proof. (Completeness) For a base case $m = 1$, the completeness can get straightforward by our premise: completeness of AggMEC and $(G, H, \text{ck}_1, P, c; \mathbf{a}, \mathbf{b}) \in$

$\mathcal{R}_{\text{GenPT4}}^{1,1}$. Let consider the case $m > 1$. In this case, we show that if the input $(G, H, \text{ck}_{P, \mu+s}, P, c; \mathbf{a}, \mathbf{b})$ belongs to $\mathcal{R}_{\text{GenPT4}}^{1,n}$, then the updated input $(G, H, \text{ck}_{P, \mu+s+1}, \hat{P}, \hat{c}; \hat{\mathbf{a}}, \hat{\mathbf{b}})$ belongs to $\mathcal{R}_{\text{GenPT4}}^{1,n/2}$. Following the \mathcal{P} algorithm, we get the following equations:

$$\begin{aligned} \hat{c} &= x^{-1}c_L + c + xc_R = \langle \mathbf{a}_L, x^{-1}\mathbf{b}_R \rangle + \langle \mathbf{a}, \mathbf{b} \rangle + \langle x\mathbf{a}_R, \mathbf{b}_L \rangle = \langle \hat{\mathbf{a}}, \hat{\mathbf{b}} \rangle \\ \hat{P} &= (\mathbf{P}^{(q_1)} \parallel [x]\mathbf{P}^{(q_4)}) + (\mathbf{P}^{(q_2)} \parallel [x^{-1}]\mathbf{P}^{(q_3)}) \\ &= [\mathbf{a}_L]G \parallel [x^{-1}\mathbf{b}_R]H + [x\mathbf{a}_R]G \parallel [\mathbf{b}_L]H = [\hat{\mathbf{a}}]G \parallel [\hat{\mathbf{b}}]H \\ \hat{P} &= \text{Com}_2(\text{ck}_\nu, \hat{P}) = \text{Com}_{\text{TC}}((G \parallel H, \text{ck}_\nu), \mathbf{a} \parallel \mathbf{b}) \end{aligned}$$

Therefore, we can conclude that updated input $(G, H, \text{ck}_{P, \mu+s+1}, \hat{P}, \hat{c}; \hat{\mathbf{a}}, \hat{\mathbf{b}})$ belongs to $\mathcal{R}_{\text{GenPT4}}^{1,n/2}$.

(Witness-extended-emulation) For the computational witness-extended emulation, we construct an expected polynomial time extractor \mathcal{E}_{Col} whose goal is to extract a witness by using a polynomially bounded tree of accepting transcripts. \mathcal{E}_{Col} takes public inputs $(\text{pp}_\nu, G, H, P, c, st_V; \mathbf{a}, \mathbf{b}, st_P)$. By premise, \mathcal{E}_{Col} exploits a PPT extractor \mathcal{E}_{MEC} , that extract st_P which consists of commitments $(\mathbf{P}_k)_{k=\mu+1}^{\mu+\nu+1}$, which satisfies Eq. (5) and $P_k = \text{Com}_2(\text{ck}_\nu, \mathbf{P}_k)$. In the similar way in proof of Theorem 1, we show that how to extract witness \mathbf{a}, \mathbf{b} from accepting transcripts. By the general forking lemma, it is sufficient to construct an extractor \mathcal{E}_{Row} that extracts a witness from a suitable tree of accepting transcripts in probabilistic polynomial time. We begin with $\underbrace{(3, \dots, 3)}_{\log_2 n}$ -tree of accepting tran-

scripts. Since the number of leaves of the tree is polynomially bound, $3^{\log_2 n}$, we can apply the general forking lemma.

First, in the base case $n = 1$, the \mathcal{P} sends witnesses a, b to \mathcal{V} and \mathcal{V} check the relation directly. That means the witness a and b belongs to transcripts and \mathcal{E}_{Col} can extract them.

Now we consider the case $n > 1$. We construct extractor \mathcal{E}_{Col} by inductively extraction. That is, retrieves s -round witness $\mathbf{a}^{(s)}, \mathbf{b}^{(s)} \in \mathbb{Z}_p^{1 \times n/2^s}$ from next round witnesses $\mathbf{a}^{(s+1)}, \mathbf{b}^{(s+1)} \in \mathbb{Z}_p^{1 \times n/2^{s+1}}$ recursively.

First, \mathcal{E}_{Col} can extract final round witnesses $a^{(\nu+1)}$ and $b^{(\nu+1)}$. We assume that $\hat{\mathbf{a}}, \hat{\mathbf{b}} \in \mathbb{Z}_p^{1 \times n/2^{s+1}}$ is valid witness of instance $(G, H, \hat{P}_i, \hat{c}_i)$, that are affected by challenge x_i for the relation $\mathcal{R}_{\text{GenPT4}}^{1,n/2^{s+1}}$. From the tree of accepting transcript, we can get 3 instance-witness pairs: $(G, H, \hat{P}_i, \hat{c}_i; \hat{\mathbf{a}}_i, \hat{\mathbf{b}}_i)$. Furthermore, \mathcal{E}_{Col} can get k -round prover's commitments (P, \hat{P}) and their message $(\mathbf{P}, \hat{\mathbf{P}})$ from transcript and \mathcal{E}_{MEC} respectively. From 2 distinct tuples, \mathcal{E}_{Col} can construct following linear system:

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \begin{bmatrix} \mathbf{P}^{(q_1)} \\ \mathbf{P}^{(q_2)} \end{bmatrix} = \begin{bmatrix} [\hat{\mathbf{a}}_1]G \\ [\hat{\mathbf{a}}_2]G \end{bmatrix}, \begin{bmatrix} 1 & x_1^{-1} \\ 1 & x_2^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{P}^{(q_3)} \\ \mathbf{P}^{(q_4)} \end{bmatrix} = \begin{bmatrix} [\hat{\mathbf{b}}_1]H \\ [\hat{\mathbf{b}}_2]H \end{bmatrix} \quad (12)$$

By DL assumption, \mathcal{E}_{Col} solves the linear equation and then get the representation $\mathbf{p}^{(q_1)}, \mathbf{p}^{(q_2)} \in \mathbb{Z}_p^{n/2^{s+1}}$ of $\mathbf{P}^{(q_1)}, \mathbf{P}^{(q_2)} \in \mathbb{G}^{n/2^{s+1}}$ under base G and

$\mathbf{p}^{(q_3)}, \mathbf{p}^{(q_4)} \in \mathbb{Z}_p^{n/2^{s+1}}$ of $\mathbf{P}^{(q_3)}, \mathbf{P}^{(q_4)} \in \mathbb{G}^{n/2^{s+1}}$ under base H respectively. Then $\mathbf{p} = \mathbf{p}^{(q_1)} \parallel \mathbf{p}^{(q_2)} \parallel \mathbf{p}^{(q_3)} \parallel \mathbf{p}^{(q_4)}$ is naturally representation of \mathbf{P} . Let $\mathbf{p}^{(q_i)}$ denote $\tilde{\mathbf{a}}_L = \mathbf{p}^{(q_1)}, \tilde{\mathbf{a}}_R = \mathbf{p}^{(q_2)}, \tilde{\mathbf{b}}_L = \mathbf{p}^{(q_3)}, \tilde{\mathbf{b}}_R = \mathbf{p}^{(q_4)}$. In the similar way in Eq. (11) of Theorem 1, 3 distinct challenges guarantee extracted vectors $(\tilde{\mathbf{a}}, \tilde{\mathbf{b}})$ is equal to the value c . Therefore, the extracted witnesses $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ is valid witness for the relation $\mathcal{R}_{\text{GenPT4}}^{1, n/2^s}$. By inductively retrieving process and general forking lemma, \mathcal{E}_{Col} can extract witness vectors \mathbf{a} and \mathbf{b} . \square

D Proof of Theorem 3

Proof. (Completeness) Assume that the input $\text{ck}_k, (L_k, R_k, P_k, x_k); (\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k)$ satisfies Eq. (3), (4), and (5). By the homomorphic property of polynomial commitment scheme and perfect completeness of Eval and Plonkish_{Eval}, \mathcal{V} accepts both Eval and Plonkish_{Eval}. Therefore, we are shown the completeness of AggMEC. (Witness-Exetended Emulation) For the computational witness-extended emulation, we construct an expected polynomial-time extractor \mathcal{E}_{MEC} whose goal is to extract a witness by using a polynomially bounded tree of accepting transcripts. \mathcal{E}_{MEC} takes public inputs $\text{ck}_k, (L_k, R_k, P_k, x_k)$ and returns witness vectors $(\mathbf{L}_k, \mathbf{R}_k, \mathbf{P}_k)$ satisfying Eq. (3), Eq. (4), and Eq. (5).

By the general forking lemma, it is sufficient to construct an extractor \mathcal{E}_{MEC} that extracts a witness from a suitable tree of accepting transcripts in probabilistic polynomial-time. We begin with a $(6 \log m + 2 \log n + 2, 5)$ -tree of accepting transcripts. Since the number of leaves in the tree is polynomially bounded, we can apply the general forking lemma [16].

By our premise, one can construct a PPT extractor $\mathcal{E}_{\text{Eval}}$ for PCS.Eval. In addition, since the above premise implies that the Plonkish_{Eval} is an AoK, one can construct a PPT extractor $\mathcal{E}_{\text{Plonkish}}$ for Plonkish_{Eval} that extracts wire polynomials $\{w^{(i)}(X)\}_{i=0}^{M-1}$. The \mathcal{E}_{MEC} uses them as sub-routines.

First, for $i \in \{1, 2\}$, the \mathcal{E}_{MEC} gets $F_P(X)$ and $w^{(i)}(X)$ by feeding $\mathcal{E}_{\text{Eval}}$ with $(\text{ck}_{\text{PC}}, P, z, y)$ and $(\text{ck}_{\text{PC}}, W^{(i)}, z, r^{(i)})$, respectively. From the 5 transcripts from distinct challenge τ , the \mathcal{E}_{MEC} extracts F_V , a polynomial $a^{(i)}(X)$, and quotient polynomials $q^{(i)}(X)$ by regarding the following relation as a polynomial with respect to τ of degree 4: $F_P = F_V + \sum_{i=1}^2 (\tau^i a^{(i)} + \tau^{2+i} q^{(i)})$. Note that $w^{(i)}(X)$, $a^{(i)}(X)$, and $q^{(i)}(X)$ satisfy $a^{(i)}(z) = w^{(i)}(z) - q^{(i)}(z)(z^d - 1)$ for $i \in \{1, 2\}$.

From the $6 \log m + 2 \log n + 2$ transcripts from distinct challenge ρ , the \mathcal{E}_{MEC} extracts polynomials $F_{L,k}^{(i)}, F_{R,k}^{(i)}, F_{P,k}^{(i)}$ by regarding the following relationship as a polynomial with respect to ρ of degree $6 \log m + 2 \log n + 1$:

$$F_V = \sum_{i=1}^2 \left(\sum_{k=1}^{\mu} (\rho^{4k-2-i} F_{L,k}^{(i)} + \rho^{4k-i} F_{R,k}^{(i)}) + \rho^{4\mu} \left(\sum_{k=0}^{\mu+\nu} \rho^{2k-1+i} F_{P,k}^{(i)} \right) \right).$$

The extracted polynomials satisfy the following relation:

$$\mathbf{L}_k^{(i)} = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, F_{L,k}^{(i)}), \mathbf{R}_k^{(i)} = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, F_{R,k}^{(i)}), \mathbf{P}_k^{(i)} = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, F_{P,k}^{(i)}) \quad (13)$$

Finally, \mathcal{E}_{MEC} outputs $\mathbf{L}_k^{(i)}, \mathbf{R}_k^{(i)}, \mathbf{P}_k^{(i)}$ by decoding $F_{L,k}^{(i)}, F_{R,k}^{(i)}, F_{P,k}^{(i)}$ respectively.

It remains to check that these extracted vectors are valid witnesses satisfying all relations from Eq. (3) to (5). First, by the extraction process, the extracted vectors $\mathbf{L}_k^{(i)}, \mathbf{R}_k^{(i)}, \mathbf{P}_k^{(i)}$ satisfy Eq. (13), so does Eq. (3). In addition, by the construction of $\mathbf{A}^{(i)}$, the polynomial $a^{(i)}$ is equal to the sum of $F_{L,k}^{(i)}, F_{R,k}^{(i)}, F_{P,k}^{(i)}$, *i.e.* $a^{(i)} = \sum_{k=1}^{\mu} (F_{L,k}^{(i)} + F_{R,k}^{(i)}) + \sum_{k=0}^{\mu+\nu} F_{P,k}^{(i)}$. This implies that the evaluations of wire polynomial $w^{(i)}$ at appropriate ξ^i contain those of $F_{L,k}^{(i)}, F_{R,k}^{(i)}, F_{P,k}^{(i)}$, where each value matches with the values of $\mathbf{L}_k^{(i)}, \mathbf{R}_k^{(i)}, \mathbf{P}_k^{(i)}$ at the corresponding positions for $i \in \{1, 2\}$. Furthermore, the wire polynomials $\{w^{(i)}(X)\}_{i=0}^{M-1}$ extracted from $\mathcal{E}_{\text{Plonkish}}$ ensure that $\mathbf{L}_k^{(i)}, \mathbf{R}_k^{(i)}, \mathbf{P}_k^{(i)}$ satisfy the relation Eq. (4) and Eq. (5).

To sum up, the extracted vectors $\mathbf{L}_k^{(i)}, \mathbf{R}_k^{(i)}, \mathbf{P}_k^{(i)}$ are actually the valid witnesses, concluding that AggMEC satisfies computational witness extended emulation. \square

E Plonkish: Proof for Elliptic Curve Relation

Plonkish is a generalization of Plonk [29], one of the well-known methods to represent the circuit satisfiability of the given AC as the constraints system. Since it supports *custom gates*, such as elliptic curve operations, we employed it to check the elliptic curve relation at Eq. (4) and Eq. (5). In this section, we provide supplementary description of Plonkish. Here, we focus on introducing the general workflow of Plonkish only; details about the constraint systems will be provided in the next section.

We first provide a basic idea of Plonk arithmetization. For each gate in the circuit, Plonk constructs a constraint equation according to the type, *e.g.*, addition or multiplication, of the gate. To represent this, Plonk adopts an auxiliary variable called the *selector* that indicates which types of gates are enabled or not in the current gate. To ensure that the given two gates are connected, Plonk exploits the constraints using a permutation, which ensures that the values in the wires that connecting gates do not change after permutation on them. With Lagrange Interpolation for left inputs, right inputs, outputs, and selectors separately, using a cyclic group generated by the D -th root of unity ξ of \mathbb{Z}_q , all constraint equations except the permutation constraints can be expressed as a single polynomial equation. Note that in this section, $[\ell]$ denotes a set of integers from 0 to $\ell - 1$.

Plonkish generalizes Plonk by handling all the values occurred in the execution of the circuit as the *execution trace* $\mathbb{Z}_q^{D \times M}$. Each row represents the inputs, outputs, or auxiliary values occurred in the corresponding execution step. This execution trace can be represented as a sequence of polynomials by applying Lagrange interpolation with respect to each column. Each gate can be written as polynomial comprised of column polynomials that are engaged to the current gate. After then, arguments for gate identity and permutation can be constructed using these polynomials. Formally, let $\{w^{(i)}(X)\}_{i=0}^{M-1}$ be the polynomials that represents the execution trace of the given circuit, which correspond to the column polynomials mentioned. For the number N_g of the types of gates in the

circuit, we denote $\{c_i(X)\}_{i=0}^{N_g-1}$ as the gate polynomials for the circuit. Each gate polynomial can be represented as $c_i(X) = g_i(w^{(0)}(X), w^{(1)}(X), \dots, w^{(M-1)}(X))$ for some M -variate polynomial g_i . Let us define $\{s_i(X)\}_{i=0}^{N_g-1}$ as the selector polynomials.

In addition, for the permutation argument, we denote a permutation $\sigma : [D] \times [M] \rightarrow [D] \times [M]$. $\sigma(i, j) = (\sigma(i, j)_1, \sigma(i, j)_2)$ is equivalent to $w^{(i)}(\xi^j) = w^{(\sigma(i, j)_1)}(\xi^{\sigma(i, j)_2})$. Suppose $D = 2^k$ and δ is a T -th root of unity, where $T \cdot 2^S + 1 = q$ with odd T and $k \leq S$. We use $\delta^i \cdot \xi^j$ as the label for a value corresponding to $(\sigma(i, j)_1, \sigma(i, j)_2)$, as mentioned in [55]. Define $\text{ID}_i(\xi^j) = \delta^i \cdot \xi^j$ that is an identity polynomial of $w^{(i)}(\xi^j)$ and $r_i(\xi^j) = \delta^{\sigma(i, j)_1} \cdot \xi^{\sigma(i, j)_2}$. The idea behind the permutation argument technique is the fact that $\prod_{h=0}^{D-1} \prod_{i=0}^{M-1} \frac{w^{(i)}(\xi^h) + u_1 \text{ID}_i(\xi^h) + u_2}{w^{(i)}(\xi^h) + u_1 r_i(\xi^h) + u_2}$ is equal to 1 when $w^{(i)}(\xi^j) = w^{(\sigma(i, j)_1)}(\xi^{\sigma(i, j)_2})$ for random values u_1, u_2 . We can check the details for the technique in [5].

Plonkish is a protocol for arithmetic circuit satisfiability, and the circuit satisfiability is ensured when (1) $w^{(i)}(\xi^j) = w^{(\sigma(i, j)_1)}(\xi^{\sigma(i, j)_2})$ for $i \in [D], j \in [M]$ and (2) $\sum_{i=0}^{N_g-1} s_i(X)c_i(X) = 0 \pmod{X^D - 1}$. As shown in several studies [28, 29, 55], the relations can be efficiently proved by the Polynomial IOP instantiated by PCS [18]. In short, to check polynomial relations, the prover commits polynomial and then the verifier sends random point as challenge. After then, the prover responds evaluations. To verify the responds, the prover and verifier run Eval interactive proof. Thanks to the Fiat-Shamir transform, interactive proof Eval can be converted to non-interactive proof system.

We provide a brief idea of [29] to construct the polynomial relation covering both (1) and (2) as follows: First, in line 3, the prover computes $z(X)$, which is the interpolation of the values obtained by multiplying $\prod_{i=0}^{M-1} \frac{w^{(i)}(\xi^h) + u_1 \text{ID}_i(\xi^h) + u_2}{w^{(i)}(\xi^h) + u_1 r_i(\xi^h) + u_2}$ one by one for $h \in [D]$. Then, as shown by [5], $z(X)$ satisfies $z(\xi X)/z(X) = \prod_{i=0}^{M-1} (w^{(i)}(X) + u_1 \text{ID}_i(X) + u_2) / \prod_{i=0}^{M-1} (w^{(i)}(X) + u_1 r_i(X) + u_2) \pmod{X^D - 1}$ and $z(\xi^0) = 1$ for random challenges u_1, u_2 . Hence, by combining these constraints and the gate constraints by another random challenge u_3 , the prover computes $t(x)$, as described in line 5. Now, checking that $t(\xi^i) = 0$ for $i \in [D]$ is sufficient to convince the relations (1) and (2), which can be done by several runs of Eval. We describe Plonkish protocol in Algorithm 4.

Throughout the algorithm, note that committing $t(X)$ and $q(X)$ in line 5 would require a longer commitment key than D , namely, $\deg(t(X)) \cdot D$. Since the degree of all polynomials appearing in Cougar is at most D except them, directly increasing the length of ck_{PC} would lead to inefficiency in the computational cost for both the prover and verifier. To avoid this, as did in `halo2` library [55], we parse $t(X)$ and $q(X)$ into polynomials of degree at most D and commit them separately. More precisely, if we denote d_t as the larger one among the maximum degree of the used custom gate and the number of columns, then the degrees of $t(X)$ and $q(X)$ are $D \cdot d_t$ and $D \cdot (d_t - 1)$, respectively. In this case, we denote $t_1(X), \dots, t_{d_t}(X)$ and $Q_1(X), \dots, Q_{d_t-1}(X)$ as unique polynomials of degree at most D satisfying $t(X) = \sum_{i=1}^{d_t} X^{D(i-1)} t_i(X)$ and $q(X) = \sum_{i=1}^{d_t-1} X^{D(i-1)} q_i(X)$. In this setting, the prover sends commitments T_i and Q_j from each $t_i(X)$ and

Algorithm 4 Plonkish_{Eval}

Plonkish_{Eval}(ck_{PC}, {s_i(X), g_i(X₀, ..., X_{M-1})}_{i=0}^{N_g-1}, {r_i(X)}_{i=0}^{M-1}; {w⁽ⁱ⁾(X)}_{i=0}^{M-1})

Precompute: C_{ID_i} = Comp_{PC}(ck_{PC}, ID_i(X)), C_{r_i} = Comp_{PC}(ck_{PC}, r_i(X)), i ∈ [M]

1: \mathcal{P} sends $W^{(i)} = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, w^{(i)}(X))$ to \mathcal{V}

2: \mathcal{V} chooses $u_1, u_2 \xleftarrow{\mathbb{S}} \mathbb{Z}_q$ and sends it to \mathcal{P}

3: \mathcal{P} sends $Z = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, z(X))$ to \mathcal{V} where

$$z(X) = H_0(X) + \sum_{j=0}^{D-2} \left(H_{j+1}(X) \prod_{h=0}^j \prod_{i=0}^{M-1} \frac{w^{(i)}(\xi^h) + u_1 \text{ID}_i(\xi^h) + u_2}{w^{(i)}(\xi^h) + u_1 r_i(\xi^h) + u_2} \right).$$

$$H_j(X) = \prod_{i \neq j, i \in [D]} \frac{X - \xi^i}{\xi^j - \xi^i} \text{ for all } j \in [D].$$

4: \mathcal{V} chooses $u_3 \xleftarrow{\mathbb{S}} \mathbb{Z}_q$ and sends it to \mathcal{P} .

5: \mathcal{P} sends $T = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, t(X))$, $Q = \text{Comp}_{\text{PC}}(\text{ck}_{\text{PC}}, q(X))$ to \mathcal{V} where

$$t(X) = \sum_{i=0}^{N_g-1} s_i(X) g_i(w^{(0)}(X), \dots, w^{(M-1)}(X)) + u_3 \cdot z(X) \prod_{i=0}^{M-1} (w^{(i)}(X) + u_1 \text{ID}_i(X) + u_2) \\ - u_3 \cdot z(\xi X) \prod_{i=0}^{M-1} (w^{(i)}(X) + u_1 r_i(X) + u_2) + u_3^2 \cdot (z(X) - 1) H_0(X)$$

$$q(X) = \frac{t(X)}{z_H(X)}, \text{ where } z_H(X) = \prod_{i=0}^{D-1} (X - \xi^i).$$

6: \mathcal{V} chooses $u_4 \xleftarrow{\mathbb{S}} \mathbb{Z}_q$ and sends it to \mathcal{P} .

7: \mathcal{P} sends $\{\alpha_i = w^{(i)}(u_4)\}_{i=0}^{M-1}$, $\beta = z(u_4)$, $\gamma = z(\xi u_4)$,
 $\{\phi_i = \text{ID}_i(u_4)\}_{i=0}^{M-1}$, and $\{\psi_i = r_i(u_4)\}_{i=0}^{M-1}$ to \mathcal{V} .

8: \mathcal{V} evaluates ρ_1 and $\rho_2 = \rho_1 / z_H(u_4)$ using the values received from \mathcal{P}

$$\rho_1 = \sum_{i=0}^{N_g-1} s_i(u_4) \cdot g_i(\alpha_0, \dots, \alpha_{M-1}) + u_3 \cdot \beta \prod_{i=0}^{M-1} (\alpha_i + u_1 \cdot \phi_i + u_2) \\ - u_3 \cdot \gamma \prod_{i=0}^{M-1} (\alpha_i + u_1 \cdot \psi_i + u_2) + u_3^2 \cdot (\beta - 1) H_0(u_4)$$

9: \mathcal{P} and \mathcal{V} set run $\text{Eval}(\text{ck}_{\text{PC}}, T, u_4, \rho_1; t(X))$, $\text{Eval}(\text{ck}_{\text{PC}}, W^{(i)}, u_4, \alpha_i; w^{(i)}(X))_{i \in [M]}$,

$\text{Eval}(\text{ck}_{\text{PC}}, Q, u_4, \rho_2; q(X))$, $\text{Eval}(\text{ck}_{\text{PC}}, Z, u_4, \beta; z(X))$, $\text{Eval}(\text{ck}_{\text{PC}}, Z, \xi u_4, \gamma; z(X))$,

$\text{Eval}(\text{ck}_{\text{PC}}, C_{\text{ID}_i}, u_4, \phi_i; \text{ID}_i(X))_{i \in [M]}$, and $\text{Eval}(\text{ck}_{\text{PC}}, C_{r_i}, u_4, \psi_i; r_i(X))_{i \in [M]}$.

$q_j(X)$ in **line 5**, and later be requested to open each of them at the random challenge u_4 provided by the verifier.

From the perspective of the verifier, it can compute values expected to be $T(u_4)$ and $Q(u_4)$ by combining received evaluation points in **line 6**. However, Eval cannot directly proceed with $t(X)$ and $q(X)$ because they are not committed. To mitigate this, we assume that the prover additionally sends evaluation points $\hat{t}_1 := t_1(u_4), \dots, \hat{t}_{d_t} := t_{d_t}(u_4)$ and $\hat{q}_1 := q_1(u_4), \dots, \hat{q}_{d_t-1} := q_{d_t-1}(u_4)$ to the verifier. Then the verifier checks

$$\rho_1 \stackrel{?}{=} \sum_{i=1}^{d_t} \hat{t}_i \cdot u_4^{d(i-1)}, \quad \rho_2 \stackrel{?}{=} \sum_{i=1}^{d_t-1} \hat{q}_i \cdot u_4^{d(i-1)}$$

and run $\text{Eval}(\text{ck}_{\text{PC}}, T_i, u_4, \hat{t}_i; t_i(X))$ and $\text{Eval}(\text{ck}_{\text{PC}}, Q_j, u_4, \hat{q}_j; q_j(X))$ for $i \in [d_t]$ and $j \in [d_t - 1]$, respectively.

Indeed, this modification carries additional communications of $(2d_t - 3)$ group elements in \mathbb{G}_t and additional $(2d_t - 3)$ executions of $\text{Leopard}_{\text{PC}}.\text{Eval}$. In particular, the prover needs to compute each T_i and Q_j , *i.e.*, requiring more computations. Nevertheless, with the batching technique presented in Section H, the computational cost for the verifier would be significantly reduced compared to the case when we use the commitment key of length $d_t \cdot D$.

Plonkish_{Eval}. Using the constraints system that will be described in the next subsection, we can construct the polynomial g_{MA} for mixed elliptic point addition, which takes five polynomials $\{w^{(i)}(X)\}_{i=0}^4$ corresponding to each column as inputs. With two selector polynomials $s_1(X), s_2(X)$, we denote $\text{Plonkish}_{\text{Eval}}$ as an instantiation of Plonkish with the custom gate polynomial g_{MA} .

F Cougar-Friendly Constraint System

We present a Plonkish-type constraint system tailored for proving elliptic curve relations by introducing novel custom gates. Throughout this section, we denote the degree of an operation as the depth of the arithmetic circuit with respect to multiplication gates.

Constraints and Affine Representation. To ensure the elliptic curve relations Eq. (4) and (5), the prover should construct an execution trace consisting of elliptic curve additions and doublings. The authors of [39] addressed this by considering an elliptic curve point as a triplet in \mathbb{Z}_q^3 using the projective coordinate representation, which demands large communication and computation costs. By using the constraint system for elliptic curve operation in affine representation [55], we do not need to stick to projective representation like [39].

Custom Gate for Projective-Affine Mixed Operations. As presented in halo2 [55], the custom gates for them on the affine representation would reduce the number of input gates. However, many implementations of elliptic curve

Algorithm 5 Double-and-Add for Scalar Multiplication

-
- Input:** $P = (X, Y, 1)$ on $E : y^2z = x^3 + axz^2 + bz^3$ defined over \mathbb{Z}_p and $k \in \mathbb{Z}_q$.
- 1: Initialize $R \leftarrow (0, 1, 0)$
 - 2: Compute $(k_1, \dots, k_{\lceil \log_2 q \rceil}) \in \{0, 1\}^{\lceil \log_2 q \rceil}$ such that $k = \sum_{i=1}^{\lceil \log_2 q \rceil} k_i 2^{i-1}$.
 - 3: For i from $\lceil \log_2 q \rceil$ to 0 do:
 - 4: Update $R \leftarrow [2]R + [k_i]P$.
 - 5: **Return** R .
-

groups use group operations on projective coordinates [47] to improve efficiency by avoiding modular inversions. In particular, they utilize *mixed* addition to further reduce the number of field operations needed, which is an elliptic curve addition for two curve points of a projective representation and an affine representation, respectively, returning their sum in projective representation. Therefore, using custom gates for affine representation would be less efficient compared to the case without considering constructing an execution trace.

We address this issue by constructing a custom gate for mixed elliptic curve operations presented in [47]. For a projective point $P = (X_1, Y_1, Z_1)$ and an affine point $Q = (X_2, Y_2)$ on the prime-order short Weierstrass elliptic curve $y^2 = x^3 + b^2$, the mixed addition $P + Q = (X_3, Y_3, Z_3)$ can be computed by

$$\begin{aligned} X_3 &= (X_1 Y_2 + X_2 Y_1)(Y_1 Y_2 - 3bZ_1) - 3b(Y_1 + Y_2 Z_1)(X_1 + X_2 Z_1), \\ Y_3 &= (Y_1 Y_2 + 3bZ_1)(Y_1 Y_2 - 3bZ_1) + 9bX_1 X_2 (X_1 + X_2 Z_1), \\ Z_3 &= (Y_1 + Y_2 Z_1)(Y_1 Y_2 + 3bZ_1) + 3X_1 X_2 (X_1 Y_2 + X_2 Y_1). \end{aligned}$$

In addition, the point doubling $[2]P = (X_3, Y_3, Z_3)$ can also be performed by

$$\begin{aligned} X_3 &= 2X_1 Y_1 (Y_1^2 - 9bZ_1^2), \\ Y_3 &= (Y_1^2 - 9bZ_1^2)(Y_1^2 + 3bZ_1^2) + 24bY_1^2 Z_1^2, \\ Z_3 &= 8Y_1^3 Z_1. \end{aligned}$$

Note that these operations are of degree 4, and [47] provided efficient algorithms for computing them via field additions and multiplications. By using these operations, we can compute the scalar multiplication on the elliptic curve through the well-known double-and-add algorithm, which is presented in Algorithm 5. Since the doubling for the first iteration of the loop always results in the identity point, we omit it when configuring an execution trace for this operation.

However, the custom gates for mixed elliptic curve operations should be carefully combined with our two-tier commitment scheme. This is because in the second layer of the two-tier commitment, each elliptic curve group element is converted to a tuple of field elements by viewing them as an affine coordinate.

² Of course, [47] provided general results for curves of the form $y^2 = x^3 + ax + b$. We presented a special case for the sake of efficiency.

	S_1	S_2	C_1	C_2	C_3	C_4	C_5	
1	1	1	0	1	0	$p^{(1)}$	$p^{(2)}$	Mixed Addition
	1	0	$p^{(1)}$	$p^{(2)}$	$p^{(3)}$			Doubling
1	1	1	$[2]P^{(1)}$	$[2]P^{(2)}$	$[2]P^{(3)}$	$p^{(1)}$	$p^{(2)}$	Mixed Addition
	1	0	$[3]P^{(1)}$	$[3]P^{(2)}$	$[3]P^{(3)}$			Doubling
0	0	1	$[6]P^{(1)}$	$[6]P^{(2)}$	$[6]P^{(3)}$			Identity Addition
	0	0	$[6]P^{(1)}$	$[6]P^{(2)}$	$[6]P^{(3)}$	$[6]P^{(1)}$	$[6]P^{(2)}$	Equality Check

$6 = (110)_2$

Fig. 3. Description of the execution trace for computing $[6]P$.

That is, the final result should be presented in affine point representation. Fortunately, checking the equality of two points $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2)$ can be efficiently done by the following degree 2 operation:

$$X_1 = X_2 Z_1, \quad Y_1 = Y_2 Z_1$$

In addition, since at least one of the two input points should be represented as an affine coordinate for all operations, such a position would be utilized to hold values of L_i , R_i , and P_i for the sake of consistency check procedures during AggMEC. Therefore, all of these operations can be implemented at most degree 4 ACs, harmonizing well with our Cougar construction.

Efficient Construction of Execution Trace for Scalar Multiplication.

We now construct a Plonkish-style execution trace for these *basic* operations. By using them as building blocks, our goal is to give an efficient construction of an execution trace for the scalar multiplication, which is a dominating operation in both relations Eq. (4) and (5).

We consider the execution trace for seven columns, where two columns are assigned for two selectors, S_1 and S_2 , and the remaining ones are for one projective point $P = (X_1, Y_1, Z_1)$ and one affine point $Q = (X_2, Y_2)$, respectively. Here, we used two selectors for representing four types of operations, namely, addition, addition by identity, doubling, and equality check, that would be encountered during the scalar multiplication. For each operation, we assign the values of selectors as $(S_1, S_2) = (1, 1), (0, 1), (1, 0),$ and $(0, 0)$, respectively. Of course, it is more natural to employ four selectors corresponding to these four operations; we decided to use two selectors to reduce the number of columns. Moreover, because the outputs of mixed addition and doubling are points in projective coordinate, we assign the output of these operations to the next row of columns for storing projective points. This also saves the number of columns. To help understand, we provide an execution trace for checking $Q = [6]P$ in Fig. 3. In this figure, $[k]P^{(i)}$ means the i th coordinate of the point $[k]P$, and the prime symbol means the affine representation of the point. The points without the prime symbol are the projective representations. For the general case $Q = [k]P$ for $k \in \mathbb{Z}_q$, exactly $2\lceil \log q \rceil$ rows are sufficient for constructing the execution trace.

We note that these custom gates can be efficiently calculated by running the algorithms introduced by [47, Algorithm 8, 9] in the Lagrange domain.

Compatibility with Plonk-Friendly Encoding. By using the above construction, the prover can construct the intended execution trace for ensuring the elliptic curve relations. However, in order to make the execution trace compatible with the plonk-friendly encoding technique, the prover should carefully place each \mathbf{L} , \mathbf{R} , and \mathbf{P} in the right position for each ζ^i .

To address this, we take a closer look at the equations in each relation, Eq. (4) and (5). First, in the k 'th round of the row reduction (Eq. (4)), the prover should construct an execution trace for ensuring the relation

$$P_{k+1,i} = [x_k^{-1}]L_{k+1,i} + P_{k,i} + [x_k]R_{k+1,i}$$

for the challenge x_k and each i 'th coordinate of \mathbf{P}_{k+1} , \mathbf{L}_{k+1} , \mathbf{P}_k , and \mathbf{R}_{k+1} , respectively. To this end, the prover can compute this in order of computing (1) $\widetilde{L}_{k+1,i} := [x_k^{-1}]L_{k+1,i}$, (2) $\widetilde{R}_{k+1,i} := [x_k]R_{k+1,i}$, (3) $\widetilde{P}_{k+1,i} := \widetilde{L}_{k+1,i} + P_{k,i} + \widetilde{R}_{k+1,i}$, and checking (4) $P_{k+1,i} = \widetilde{P}_{k+1,i}$. Here, the permutation argument is required for processing the third step because the outputs of the first step ($\widetilde{L}_{k+1,i}$) and the second step ($\widetilde{R}_{k+1,i}$) should be referred to. According to our construction, each first and second step requires $2\lceil \log q \rceil$ rows, and each $\widetilde{L}_{k+1,i}$ and $\widetilde{R}_{k+1,i}$ appear at the first row of each position. Hence, by computing $\widetilde{L}_{k+1,i} + P_{k,i}$ first during the third step, the prover can coordinate each $L_{k+1,i}$, $R_{k+1,i}$, and $P_{k,i}$ with the period of $2\lceil \log q \rceil$. Of course, the remaining rows after the fourth step can be filled with an appropriate number of *dummy* operations, *e.g.*, copying the check operations in the fourth step. Therefore, the execution trace for convincing the whole Eq. (4) can be constructed by successively iterating the above process over each coordinate and reduction stage.

The execution trace for the column reduction (Eq. (5)) can also be constructed in a similar manner, or even simpler. For the k 'th round of reduction, the prover needs to convince the following relation

$$P_{k+1,i} = P_{k,i}^{(q_1)} + [x_k]P_{k,i}^{(q_2)}, \quad P_{k+1,i+l} = P_{k,i}^{(q_3)} + [x_k^{-1}]P_{k,i}^{(q_4)},$$

for the challenge x_k and each coordinate of \mathbf{P}_{k+1} , $\mathbf{P}_k^{(q_1)}$, $\mathbf{P}_k^{(q_2)}$, $\mathbf{P}_k^{(q_3)}$, and $\mathbf{P}_k^{(q_4)}$, respectively. Here, we assumed that the length of \mathbf{P}_{k+1} is $2l$ and that for each $\mathbf{P}_k^{(q_j)}$ is l . Similar to above, the prover can compute this by computing (1) $\widetilde{P}_{k+1,i} := [x_k]P_{k,i}^{(q_2)} + P_{k,i}^{(q_1)}$, and checking (2) $P_{k+1,i} = \widetilde{P}_{k+1,i}$. The same procedure suffices for computing $P_{k+1,i+l}$. In this case, the permutation argument is not required, and the construction of the execution trace can be done in the same way as above using dummy operations appropriately.

To sum up, by aggressively exploiting dummy operations to adjust the positions of the coordinates of each \mathbf{L}_k , \mathbf{R}_k , and \mathbf{P}_k , the prover can construct an execution trace compatible with the proposed Plonk-friendly encoding technique.

Reducing Dummy Rows. Although the above-mentioned method can construct the desirable execution trace, the excessive use of dummy operations makes it longer, which increases the computational cost for running `LeopardPC.Eval`

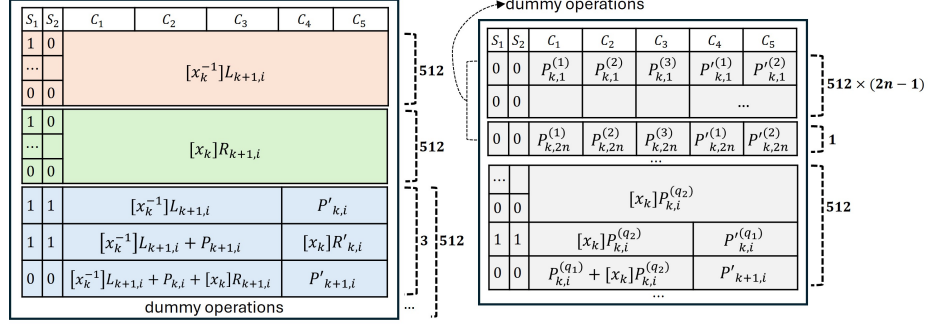


Fig. 4. Final construction of the execution trace.

on both the prover and verifier. For this reason, we present a more efficient construction of the execution trace by reducing dummy rows as much as possible.

We note that in the column reduction process, each component of \mathbf{P}_k is not needed to be placed with the same period as that for the row reduction. Recall that the goal of Plonk-friendly encoding is to facilitate the check of the membership of each \mathbf{L}_k , \mathbf{R}_k , and \mathbf{P}_k on the wire polynomial. Thus, we place each \mathbf{P}_k in the column reduction with the same period as \mathbf{L}_k , \mathbf{R}_k , and \mathbf{P}_k in the row reduction. This would not require modification in the previous AggMEC algorithm, while saving $2dn$ rows during constructing the execution trace.

In addition, we place each \mathbf{L}_k , \mathbf{R}_k , and \mathbf{P}_k in the same order of their index to simplify the indexing. For the row reduction, we placed input vectors \mathbf{L}_{k+1} , \mathbf{R}_{k+1} , and \mathbf{P}_k with a period of $2\lceil\log_2 q\rceil$. On the other hand, because the order of recorded components for column reduction does not match with the input vector, we instead place the output \mathbf{P}_{k+1} of the process with the same period. However, in this case, the output $\mathbf{P}_{\mu+1}$ of the last row reduction, *i.e.*, the input of the first column reduction, is not recorded. We mitigate this by manually recording each component of $\mathbf{P}_{\mu+1}$ into the execution trace with dummy rows containing these components. To match the period, we add an appropriate number of dummy rows except for the last component $P_{\mu+1,2n}$. This is because the next component $P_{\mu+2,1}$, the output of the first column reduction, should be separated from $P_{\mu+1,2n}$ with the period.

Remark that the execution trace should contain at least $(6n \log_2 m + 4n - 2)d$ rows because it should contain each \mathbf{L}_i , \mathbf{R}_i , and \mathbf{P}_i , *i.e.*, this approach gives an execution trace with an optimal number of rows. We illustrated the final construction of the execution trace in Fig. 4. This figure depicts a concrete case when $\log_2 q = 256$, *i.e.*, each $L_{k,i}$, $R_{k,i}$, and $P_{k,i}$ is placed with a period of 512.

G Polynomial Commitment Scheme from Leopard

In this section, we provide details about the $\text{Leopard}_{\text{PC}}$, which is a key ingredient to instantiate Cougar. Remark that [39, Section E.1.] provided a basic idea for constructing this; we present the full description for the sake of completeness.

$\text{Leopard}_{\text{PC}}$ is a natural tweak of Protocol3 [39] as a PCS. The construction idea is basically the same as the PCS introduced by [15], which was built upon BulletProofs. More precisely, we can construct PCS from IPA by regarding the point evaluation of the polynomial as an inner product between the coefficient vector and the vector comprised by the powers of the evaluation point. The asymptotic communication and computation complexities of Eval from this approach are the same as those of the underlying IPA. Note that Protocol3 features square root verifier cost and logarithmic communication cost; hence so does $\text{Leopard}_{\text{PC}}.\text{Eval}$.

Following the above approach, we provide the full description of $\text{Leopard}_{\text{PC}}$ as follows: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$ be a bilinear group, where $\mathbb{G}_1 = E(\mathbb{Z}_p)$. For a polynomial $a(X) \in \mathbb{Z}_p^{\leq mn}[X]$ and positive integers $m, n \in \mathbb{N}$, we will denote its coefficient vector as $\mathbf{a} \in \mathbb{Z}_p^{mn}$, namely, $\mathbf{a} = (a_0, \dots, a_{mn-1})$ such that $a(X) = \sum_{i=0}^{mn-1} a_i X^i$. $\text{Leopard}_{\text{PC}} = (\text{Gen}, \text{Com}, \text{Eval})$ over a message space $\mathbb{Z}_p^{\leq mn}[X]$ and a commitment space \mathbb{G}_t is defined as follows³:

- $\text{Gen}(1^\lambda) \rightarrow \text{ck}_{\text{PC}} \in \mathbb{G}_1^m \times \mathbb{G}_2^n$.
- $\text{Com}(\text{ck}_{\text{PC}} = (\mathbf{G}, \mathbf{H}), a(X)) \rightarrow P := (\mathbf{G} \otimes \mathbf{H})^{\mathbf{a}} \in \mathbb{G}_t$.

In addition, $\text{Eval} = (\mathcal{K}, \mathcal{P}, \mathcal{V})$ is an interactive argument system for the following relation:

$$\mathcal{R}_{\text{Leopard}_{\text{PC}}.\text{Eval}} = \left\{ \left(\begin{array}{l} \text{ck}_{\text{PC}} = (\mathbf{G}, \mathbf{H}) \in \mathbb{G}_1^m \times \mathbb{G}_2^n, \\ C \in \mathbb{G}_t, z, y \in \mathbb{Z}_p, d \in [mn]; \\ a(X) \in \mathbb{Z}_p^{\leq mn}[X] \end{array} \right) : \begin{array}{l} C = (\mathbf{G} \otimes \mathbf{H})^{\mathbf{a}} \\ \wedge \\ y = a(z) \end{array} \right\}$$

A typical strategy to cope with the above relation is to modify the above relation into that for IPA: For $\mathbf{z} = (1, z, \dots, z^{mn-1})$, we can rewrite $a(z) = \langle \mathbf{a}, \mathbf{z} \rangle$. For this reason, the construction of Eval is almost identical to Protocol3 except for some modifications regarding the fact that \mathbf{z} is also known to the verifier. The precise description of $\text{Leopard}_{\text{PC}}.\text{Eval}$ is given in Algorithm 6. Here, $\text{bit}(k)$ refers to the bit decomposition of a number k .

We now show that $\text{Leopard}_{\text{PC}}$ is indeed the PCS, *i.e.*, satisfying the conditions in Definition 4, under the DL assumption. In fact, $\mathbf{G} \otimes \mathbf{H}$ in the above relation can be seen as the commitment key of the Pedersen vector commitment defined over the group \mathbb{G}_t , along with a certain structure. Since the binding property of the Pedersen vector commitment depends on the DLR assumption, one can expect that the same holds for $\text{Leopard}_{\text{PC}}$ under a structure-aware version of the DLR assumption.

For this reason, we first provide a definition of generalized discrete logarithm relation (GDLR) assumption, which was previously defined in [39, Definition 8]. For simplicity, we denote \mathcal{G}_b as a bilinear group generator that takes the security parameter λ and outputs a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t)$ of order p , generators g, h for \mathbb{G}_1 and \mathbb{G}_2 , respectively, and a pairing operator e .

³ We will not consider hiding property because zero-knowledge property is unnecessary in our context.

Algorithm 6 Leopard_{PC}.Eval

-
- Leopard_{PC}.Eval(ck_{PC} = $(\mathbf{G}, \mathbf{H}) \in \mathbb{G}_1^m \times \mathbb{G}_2^n, P \in \mathbb{G}_t, z, y \in \mathbb{Z}_p; \mathbf{a} \in \mathbb{Z}_p^{mn}$)
 where $m = 2^\mu$ and $n = 2^\nu$
- 1: \mathcal{V} picks $U \xleftarrow{\$} \mathbb{G}_t$ and sends it to \mathcal{P}
 - 2: \mathcal{P} and \mathcal{V} set $P_0 = P + [y]U, \mathbf{G}_0 = \mathbf{G}, \mathbf{H}_0 = \mathbf{H}$.
 Additionally, \mathcal{P} set $\mathbf{a}_0 = \mathbf{a}$ and $\mathbf{z}_0 = [z^{m(i-1)+(j-1)}] \in \mathbb{Z}_p^{m \times n}$
 - 3: **for** $i = 0, \dots, \mu - 1$ **do**
 - 4: \mathcal{P} parses $\mathbf{a}_i, \mathbf{z}_i,$ and \mathbf{G}_i to
 $\mathbf{a}_i = [\mathbf{a}_{i,L} \parallel \mathbf{a}_{i,R}], \mathbf{z}_i = [\mathbf{z}_{i,L} \parallel \mathbf{z}_{i,R}], \mathbf{G}_i = \mathbf{G}_{i,L} \parallel \mathbf{G}_{i,R}$
 - 5: \mathcal{P} computes:
 $L_i = [\mathbf{a}_{i,L}](\mathbf{G}_{i,R} \otimes \mathbf{H}) + [\langle \mathbf{a}_{i,L}, \mathbf{z}_{i,R} \rangle]U \in \mathbb{G}_t$
 $R_i = [\mathbf{a}_{i,R}](\mathbf{G}_{i,L} \otimes \mathbf{H}) + [\langle \mathbf{a}_{i,R}, \mathbf{z}_{i,L} \rangle]U \in \mathbb{G}_t$
 - 6: \mathcal{P} sends L_i, R_i to \mathcal{V}
 - 7: \mathcal{V} chooses $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sends it to \mathcal{P}
 - 8: \mathcal{P} computes:
 $\mathbf{a}_{i+1} = \mathbf{a}_{i,L} + r_i^{-1} \mathbf{a}_{i,R}, \mathbf{z}_{i+1} = \mathbf{z}_{i,L} + r_i \mathbf{z}_{i,R} \in \mathbb{Z}_p^{m/2^{i+1} \times n}$
 $\mathbf{G}_{i+1} = \mathbf{G}_{i,L} + [r_i] \mathbf{G}_{i,R} \in \mathbb{G}_1^{m/2^{i+1}}$
 $P_{i+1} = [r_i]L_i + P_i + [r_i^{-1}]R_i \in \mathbb{G}_t$
 - 9: **end for**
 - 10: **for** $j = 0, \dots, \nu - 1$ **do**
 - 11: \mathcal{P} sets $i = j + \mu$ and then parses $\mathbf{a}_i, \mathbf{z}_i,$ and \mathbf{H}_j to
 $\mathbf{a}_i = \mathbf{a}_{i,L} \parallel \mathbf{a}_{i,R}, \mathbf{z}_i = \mathbf{z}_{i,L} \parallel \mathbf{z}_{i,R}, \mathbf{H}_j = \mathbf{H}_{j,L} \parallel \mathbf{H}_{j,R}$
 - 12: \mathcal{P} computes:
 $L_i = [\mathbf{a}_{i,L}](\mathbf{G}_\mu \otimes \mathbf{H}_{j,R}) + [\langle \mathbf{a}_{i,L}, \mathbf{z}_{i,R} \rangle]U \in \mathbb{G}_t$
 $R_i = [\mathbf{a}_{i,R}](\mathbf{G}_\mu \otimes \mathbf{H}_{j,L}) + [\langle \mathbf{a}_{i,R}, \mathbf{z}_{i,L} \rangle]U \in \mathbb{G}_t$
 - 13: \mathcal{V} chooses $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sends it to \mathcal{P}
 - 14: \mathcal{P} computes:
 $\mathbf{a}_{i+1} = \mathbf{a}_{i,L} + s_j^{-1} \mathbf{a}_{i,R}, \mathbf{z}_{i+1} = \mathbf{z}_{i,L} + s_j \mathbf{z}_{i,R} \in \mathbb{Z}_p^{n/2^{j-1}}$
 $\mathbf{H}_{j+1} = \mathbf{H}_{j,L} + [s_j] \mathbf{H}_{j,R} \in \mathbb{G}_2^{n/2^{j+1}}$
 $P_{i+1} = [s_i]L_i + P_i + [s_i^{-1}]R_i \in \mathbb{G}_t$
 - 15: **end for**
 - 16: \mathcal{P} sends $a = a_{\mu+\nu} \in \mathbb{Z}_p$ to \mathcal{V}
 - 17: \mathcal{V} computes:
 $\mathbf{r}[k+1] = \langle \text{bit}(k), (r_0, \dots, r_{\mu+\nu-1}) \rangle$ for $k = 0, \dots, m+n-1$
 Parse \mathbf{r} to $\mathbf{r}_{row} \parallel \mathbf{r}_{col}$ where $\mathbf{r}_{row} \in \mathbb{Z}_p^m$ and $\mathbf{r}_{col} \in \mathbb{Z}_p^n$
 $G = \langle \mathbf{r}_{row}, \mathbf{G}_0 \rangle, H = \langle \mathbf{r}_{col}, \mathbf{H}_0 \rangle, z = \mathbf{r}_{row} \mathbf{z}_0 \mathbf{r}_{col}$
 - 18: \mathcal{V} checks:
 $P_0 + \sum_{i \in [\mu+\nu]} ([r_i]L_i + [r_i^{-1}]R_i) = e([az]G, H)$
-

Definition 6. For $m, n \in \mathbb{N}$ and the security parameter $\lambda \in \mathbb{N}$, let GDLRsp be a sampler defined by

$$\begin{aligned} \text{GDLRsp}(1^\lambda) : (p, g, h, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e) &\leftarrow \mathcal{G}_b(1^\lambda); \mathbf{G} \stackrel{\$}{\leftarrow} \mathbb{G}_1^m; \mathbf{H} \stackrel{\$}{\leftarrow} \mathbb{G}_2^n; \\ \text{Output } (p, \mathbf{G} \otimes \mathbf{H}, \mathbb{G}_t), \end{aligned}$$

Then, we say that GDLRsp satisfies the general discrete logarithm relation (GDLR) assumption if all non-uniform polynomial-time adversaries \mathcal{A} , the following inequality holds:

$$\Pr \left[\mathbf{a} \neq \mathbf{0} \wedge \mathbf{g}^{\mathbf{a}} = 1_{\mathbb{G}_t} \mid (p, \mathbf{g} \in \mathbb{G}_t^{m \times n}, \mathbb{G}_t) \leftarrow \text{GDLRsp}(1^\lambda) \right. \\ \left. \mathbf{a} \leftarrow \mathcal{A}(p, \mathbf{g}, \mathbb{G}_t) \right]$$

where $1_{\mathbb{G}_t}$ is the identity of \mathbb{G}_t and $\text{negl}(\lambda)$ is a negligible function in λ .

As shown by [39, Theorem 5], if the DL assumption on both \mathbb{G}_1 and \mathbb{G}_2 hold, then the GDLR assumption also holds. In addition, by assuming the GDLR assumption, the binding property of $\text{Leopard}_{\text{PC}}$ holds immediately.

Now it remains to check that $\text{Leopard}_{\text{PC}}.\text{Eval}$ is an AoK for the relation $\mathcal{R}_{\text{Leopard}_{\text{PC}}.\text{Eval}}$. As we mentioned, this relation can be understood as a special case of that for Protocol3 , and Algorithm 6 is in fact almost identical to Protocol3 . We note that Protocol3 satisfies perfect completeness and computational witness-extended emulation under the GDLR assumption [38]. In fact, we made the same modifications as [15] for constructing $\text{Leopard}_{\text{PC}}.\text{Eval}$, without considering zero-knowledge. That is, the proof strategies for computational witness-extended emulation of ours and theirs are identical, except for replacing the DLR assumption with the GDLR assumption. We refer to [38] and [15] for more detailed information.

To sum up, $\text{Leopard}_{\text{PC}}$ satisfies all conditions in Definition 4 under the DL assumption on \mathbb{G}_1 and \mathbb{G}_2 . In addition, it does not require the trusted setup and features squared root verification cost and logarithmic communication cost with respect to the length of the witness. Therefore, it is a desirable PCS for instantiating Cougar .

H Optimization Tricks

In this section, we provide the optimization tricks to improve the prover and verifier cost of Cougar , which were omitted in the main text due to page limit.

Batched Execution of $\text{Leopard}_{\text{PC}}.\text{Eval}$. During AggMEC , the prover needs to execute $\text{Leopard}_{\text{PC}}.\text{Eval}$ on several polynomials and various evaluation points. In fact, by employing the techniques used in [15,10], these processes can be done in a batched manner. To convince the verifier that the evaluations of several polynomials $f_1(X), \dots, f_k(X)$ at different evaluation points u_1, \dots, u_k are v_1, \dots, v_k , respectively, it suffices to show the existence of polynomials $q_1(X), \dots, q_k(X)$ such that $f_i(X) - v_i = q_i(X)(X - u_i)$. Following the idea of [15,10], the prover

(1) constructs $\widehat{q}_i(X) := \sum_{i=1}^k r_1^{i-1} q_i(X)$ for a random challenge r_1 received from the verifier, (2) commits $\widehat{q}(X)$, (3) opens $f_1(X), \dots, f_k(X)$, and $\widehat{q}(X)$ at another random point r_2 provided by the verifier, sending $w_1 := f_1(r_2), \dots, w_k := f_k(r_2)$ and $w_{k+1} := \widehat{q}(r_2)$ to the verifier. Now, the verifier convinces the prover's claim if $w_{k+1} = \sum_{i=1}^k r_1^{i-1} \frac{w_i - v_i}{r_2 - u_i}$ holds.

We note that a single run of `LeopardPC.Eval` suffices for the third step through random linear combination on the commits of each polynomial. More precisely, for a random challenge r_3 received from the verifier, it suffices to ensure that the evaluation of $\widehat{f}(X) = \sum_{i=1}^k r_3^{i-1} f_i(X) + r_3^k \widehat{q}(X)$ at r_2 is $\sum_{i=1}^k r_3^{i-1} w_i + r_3^k w_{k+1}$. Since `LeopardPC` is a homomorphic commitment, the commitment of \widehat{f} can be computed from those of f_1, \dots, f_k and \widehat{q} , *i.e.*, the verifier can be computed itself.

Thus, only a single execution of `LeopardPC.Eval` suffices for `AggMEC`, along with a constant number of group operations on \mathbb{G}_t for merging commitments.

Concrete Parameter Selection for Verifier's Computation Cost. Under the suggested parameter in Section 3.4, the verifier cost of `Cougar` becomes cubic root with respect to the length of the witness vectors. However, when we consider the efficiency on concrete parameters, the huge constant in front of $D = O(n \log_2 m)$ leads to inefficiency, which requires the proper choice of m and n parameters to reduce D . To give a way to select m and n , we provide a concrete size of D , which is determined by m , n , and q .

$$D = \underbrace{12n \log_2 m \log_2 q + 4n \log_2 q}_{\text{Protocol.Row}} + \underbrace{(4n - 4) \log_2 q}_{\text{Protocol.Col}}.$$

That is, for small N , the term $\log_2 q$ would dominate the verifier's computation cost, so the parameters (m, n) should be selected carefully.

To this end, first recall that the verifier's computational costs consist of (1) checking $c = a \cdot b$ in the last of `Protocol.Col` through $2 \log_2 N$ field operations, (2) computing $A^{(1)}$, $A^{(2)}$, and V through multi-scalar multiplication of length $O(\log_2 N)$ each on \mathbb{G}_t during `AggMEC`, (3) running several times of `LeopardPC.Eval` during `PlonkishEval`, and (4) checking the output of the AC for `Plonkish` is $[a]G || [b]H$ through multi-scalar multiplication of length $2m$ on \mathbb{G}_p . Among them, we will focus on the third and fourth terms, whose computational cost depends on the choice of m, n for a fixed $N = mn$. Note that a single run of `LeopardPC.Eval` for a polynomial of length D requires multi-scalar multiplication of \sqrt{D} on both \mathbb{G}_1 and \mathbb{G}_2 . With the batching technique, a single execution of `LeopardPC.Eval` suffices during the entire `AggMEC`. Thus, if we denote the unit cost of each group operation in \mathbb{G}_p , \mathbb{G}_1 , and \mathbb{G}_2 as \mathbf{c}_p , \mathbf{c}_1 , and \mathbf{c}_2 , respectively, finding the optimal parameter is equivalent to solving the following optimization problem:

$$\begin{aligned} & \text{minimize } C := (\mathbf{c}_1 + \mathbf{c}_2) \sqrt{12n \log_2 q \log_2 m + (8n - 4) \log_2 q} + \mathbf{c}_p \cdot 2m, \\ & \text{subject to } N = mn. \end{aligned}$$

Since dealing with C directly would be rather cumbersome because of the $\log_2 m$ and $(8n - 4) \log_2 q$ in the radical symbol, we instead consider minimizing

$$\widehat{C} = (\mathfrak{c}_1 + \mathfrak{c}_2) \sqrt{12n \log_2 q \log_2 N + 8n \log_2 q} + \mathfrak{c}_p \cdot 2m,$$

by replacing each term with $\log_2 N$ and $8n \log_2 q$, respectively. Now if we set $n = \frac{N}{m}$, $A = 2\sqrt{\log_2 q}$, and $B = 3N \log_2 N + 2N$, then we obtain

$$\widehat{C} = A \cdot \frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2} \sqrt{\frac{B}{m}} + A \cdot \frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2} \sqrt{\frac{B}{m}} + \mathfrak{c}_p \cdot 2m. \quad (14)$$

Hence, applying the AM-GM inequality in Eq. (14) yields the following inequality:

$$\widehat{C} \geq 3 \cdot \sqrt[3]{2\mathfrak{c}_p \cdot \left(\frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2}\right)^2 A^2 B} = 6\mathfrak{c}_p \sqrt[3]{3 \left(\frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2\mathfrak{c}_p}\right)^2 \log_2 q \sqrt[3]{N \left(\log_2 N + \frac{2}{3}\right)}},$$

and equality holds when

$$m = \sqrt[3]{\frac{A^2 B}{4} \left(\frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2\mathfrak{c}_p}\right)^2} = \sqrt[3]{\left(\frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2\mathfrak{c}_p}\right)^2 (3N \log_2 N + 2N) \log_2 q}. \quad (15)$$

We note that such parameter choice ensures that the computational complexity of the verifier is $O(\sqrt[3]{N \log_2 N})$, which is a slight improvement from $O(\sqrt[3]{N} \sqrt{\log_2 N})$ in Theorem 4. In addition, to avoid the case when $m > N$, we select $m = N$ if the R.H.S. of Eq. (15) surpasses N .

Merging Multiscalar Multiplications on \mathbb{G}_t . The previous paragraph addressed the number of required group operations on \mathbb{G}_p , \mathbb{G}_1 , and \mathbb{G}_2 . We now move our focus to that on \mathbb{G}_t . Although the asymptotic required number of \mathbb{G}_t during the whole protocol is $O(\log_2 N)$, the effect of them on the concrete efficiency of the verifier would become non-negligible because of the relatively expensive unit cost for the group operation. We can precisely count the input length of each multiscalar multiplication on \mathbb{G}_t during `AggMEC` as follows:

- $3 \log_2 m + \log_2 n + 1$ for computing each $A^{(i)}$, $i \in \{1, 2\}$.
- $6 \log_2 m + 2 \log_2 n + 2$ for computing V .
- 5 for computing $P = V + [\tau]A^{(1)} + [\tau^2]A^{(2)} + [\tau^3]Q^{(1)} + [\tau^4]Q^{(2)}$.
- 36 for random linear combination during the batching technique.
- $2 \log_2 D$ for the final process of `LeopardPC.Eval`.

Our strategy to address this is to merge multiscalar multiplications on \mathbb{G}_t into a single but longer multiscalar multiplication. We note that the cost of multiscalar multiplication is sublinear to the length of the input vector [44].

To this end, we observed that with the batched evaluation technique, the verifier does not need to compute P earlier. Computing them before running the batched `LeopardPC.Eval` at the end of `AggMEC` is sufficient. In addition, if we

take a closer look at the batching process, we can figure out that the verifier computes the linear combination of a bulk of commitments before running the final process. Finally, at the end of `LeopardPC.Eval`, the verifier needs to check the consistency of the final commitment by using challenges and other commitments communicated during the protocol (line 18 in Algorithm 6). We note that the former two processes can be delayed because they do not affect the process of the remaining protocol except for the last batched `LeopardPC.Eval`. Hence, the verifier can postpone these multiscalar multiplications and merge them to the last part of `LeopardPC.Eval`. Moreover, note that V and $[\tau]A^{(1)} + [\tau^2]A^{(2)}$ share the same base group points: $L_i^{(c)}$, $R_i^{(c)}$, and $P_i^{(c)}$ for $c \in \{1, 2\}$. Thus, it suffices for the verifier to compute a multiscalar multiplication of length $(2 \log_2 D + 6 \log_2 m + 2 \log_2 n + 40)$ at the last of `LeopardPC.Eval`.

Precomputation for Efficient Commitment. Our homomorphic commitment `Com2` requires $O(D)$ computation independent of the length of the vectors. To avoid the prover’s large computation, we apply the precomputation technique in [54]. That is, the committer computes commitment $(H_i = \text{Com}_{PC}(\text{ck}_{PC}, H_i(X)))$, where $H_i(X) = \prod_{j \neq i} (\frac{1}{\zeta^j - \zeta^i}) \cdot \frac{X^D - 1}{X - \zeta^i}$ for all i . Then, the commitment to $\mathbf{L} \in \mathbb{Z}_q^N$ can be computed by N linear computation of (H_i) . Hereafter, we consider the computation cost of `Com2` for N -length message vector as $O(N)$.

I Implementation Results

We implemented `Cougar` with a famous half-pairing cycle of curves: BN254 and Grumpkin. For PCS, we utilize `LeopardPC` instantiated with BN254. We used the Fiat-Shamir transformation [26] to make them non-interactive. Every code was written in Rust, and every experiment was done in the following setting: a single thread of an AMD EPYC 7543P (2.8GHz) CPU with 512GB RAM.

I.1 Parameter Selection

For selecting m, n from the given N , we used the formula introduced in Eq. (15) with measuring coefficients \mathbf{c}_p , \mathbf{c}_1 , and \mathbf{c}_2 on our choice of elliptic curves. To this end, we first measured the ratio of unit group operations in our experimental experiments, namely, \mathbb{G}_p for Grumpkin and $\mathbb{G}_1, \mathbb{G}_2$ for BN254. We evaluate the time elapsed for 1 million group operations in each group. For $\mathbf{c}_p = 1$, we obtain $\mathbf{c}_1 = 1.0$ and $\mathbf{c}_2 = 3.2$ with an 1-sigma error from 100 runs of the experiments. In addition, since points in the Grumpkin curve can be represented as a tuple of 254-bit integers, we set $\log_2 q = 256$, which is the closest power of two from 254. We report the choice of m, n , and D for $N = 2^{10}$ to $N = 2^{20}$ in Table 2.

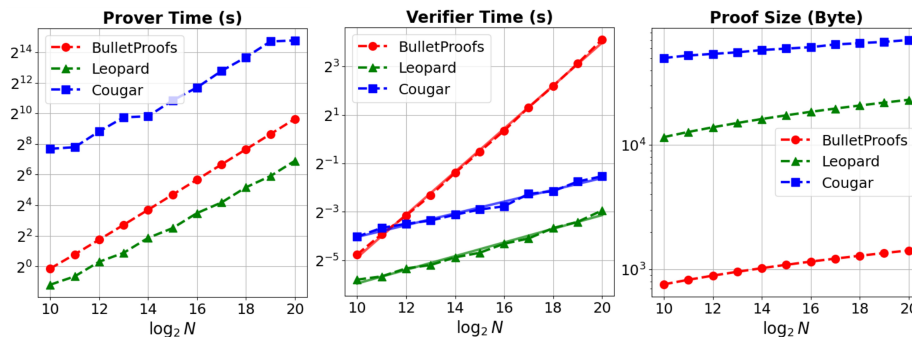


Fig. 5. Evaluation results of each IPA. We present log-log plots of each quantity for various N from 2^{10} to 2^{20} . The solid line in the verification time indicates the linear regression for each IPA in log-log plot. Best viewed in color.

N	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
m	2^8	2^9	2^9	2^9	2^{10}	2^{10}	2^{10}	2^{11}	2^{11}	2^{11}	2^{13}
n	2^2	2^3	2^3	2^4	2^4	2^5	2^6	2^6	2^7	2^8	2^8
D	2^{17}	2^{17}	2^{18}	2^{19}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{24}

Table 2. Selected parameters m, n and corresponding D for witnesses of length N . We set N around 2^{10} to 2^{20} . Each parameter is chosen by the formula in Eq. (15).

I.2 Evaluation Results

We implemented *Cougar* by following the algorithms described in the paper, with optimization tricks in Section H. For elliptic curve operations in BN254 and Grumpkin, we used `halo2curves` crate [45] of version 0.6.1. In addition, we utilized some parts of `halo2_proofs` crate [46] of version 0.3.0 to deal with polynomials during *AggMEC*. For more detailed information, we recommend the reader refer to our source code⁴.

Comparison with IPAs based on the DL Assumption. For comparison, we also implemented *BulletProofs* [16] and *Leopard* [39,38] under our experimental setting and conducted the same experiments as above. In order to implement *Bulletproofs*, we chose the *Secp256k1* curve, which is used in many cryptocurrencies. On the other hand, to implement *Leopard* we chose the *BLS12-381* curve, which is a well-known pairing friendly curve. We used `halo2curve` crate of the same version as above and `blstrs` crate [27] of version 0.7.1 for *Secp256k1* and *BLS12-381*, respectively. We note that *Secp256k1*, *BLS12-381*, and *BN254* are known to provide 127-bit, 117-bit, and 102-bit security for the DL assumption, respectively. For implementing *Leopard*, we followed several optimization tricks introduced by [38].

We report the time elapsed for proof generation and verification on each IPA. We also provide the proof size for each scheme. We conducted for various

⁴ <https://github.com/Cryptography-Algorithm-Lab/Cougar>

N	Proving time (s)			Verification time (s)			Proof Size (KB)		
	BPs	Leopard	Cougar	BPs	Leopard	Cougar	BPs	Leopard	Cougar
2^{10}	0.91	0.44	206.5	0.04	0.02	0.06	0.76	11.58	50.14
2^{11}	1.73	0.64	221.8	0.07	0.02	0.08	0.82	12.74	52.51
2^{12}	3.37	1.24	452.6	0.11	0.02	0.09	0.89	13.89	54.11
2^{13}	6.59	1.86	854.8	0.20	0.03	0.10	0.95	15.04	55.71
2^{14}	13.03	3.64	902.4	0.38	0.03	0.12	1.02	16.19	58.08
2^{15}	25.83	5.58	1842	0.70	0.04	0.13	1.09	17.34	59.68
2^{16}	51.10	11.25	3315	1.27	0.05	0.15	1.15	18.50	61.28
2^{17}	101.5	18.14	7014	2.48	0.06	0.21	1.22	19.65	64.42
2^{18}	201.8	36.11	13051	4.55	0.08	0.23	1.28	20.80	66.02
2^{19}	401.6	60.12	26646	8.71	0.09	0.30	1.35	21.95	67.62
2^{20}	801.2	120.1	27866	17.34	0.13	0.35	1.42	23.10	69.98

Table 3. Evaluation results of each IPA for various N from 2^{10} to 2^{20} .

N from 2^{10} to 2^{20} . The evaluation results for each IPA are presented and visualized in Table 3 and Fig. 5, respectively. From this figure, we can observe that the verification time of **Cougar** increases slowly than that for **BulletProofs** and **Leopard**, though the time for small $N = 2^{10}$ or 2^{11} surpasses the cost for them. To further support this, we also conducted linear regression on the verification time in log scale for each $\log_2 N$. We note that for the regression coefficients $\hat{\alpha}, \hat{\beta}$ such that $\log_2 y = \hat{\alpha} \log_2 x + \hat{\beta}$, we have that $y = 2^{\hat{\beta}} \cdot x^{\hat{\alpha}}$, *i.e.*, the coefficient $\hat{\alpha}$ for slope indicates the exponent of the verification cost with respect to N . As shown in the figure, the regression results fit well with the measured data. Concretely, the mean squared error for each scheme is given by 0.006, 0.013, and 0.009, respectively. The regression coefficients $(\hat{\alpha}, \hat{\beta})$ for **BulletProofs**, **Leopard** and **Cougar**, are $(0.887, -13.758)$, $(0.282, -8.777)$, and $(0.243, -6.465)$, respectively. We guess that the coefficient $\hat{\alpha}$ is less than the theoretically estimated values ($\hat{\alpha} = 1, 1/2$ and $1/3$ for each scheme, respectively) because the computational complexity for multi-scalar multiplication is $O\left(\frac{N}{\log_2 N}\right)$ for input vectors of length N . Nevertheless, this result indicates that the rate of increase in the verification cost for **Cougar** is slower than that of **BulletProofs** and **Leopard**.

In contrast to **BulletProofs** and **Leopard**, one can figure out that the proving time of **Cougar** increases stepwise. This is because the proving time of **Leopard** highly depends on D . As shown in Table 2, the rate of the increase in D is slower than N , and more importantly, this tendency exactly coincides with the tendency of the proof generation time depicted in Fig. 5.

Comparison with Hyrax as a PCS. Recall that one of the main usage of IPA is to instantiate Poly IOPs, and the **Cougar** also can be naturally utilized as a PCS as well. For this reason, we also compare the PCS version of **Cougar** with **Hyrax-PCS** [51], which is another PCS based on the DL assumption and featuring squared root verifier and communication costs. Although both **BulletProofs** and **Leopard** can be converted to PCS, we will not compare **Hyrax-PCS** with them because they are already compared with ours in terms of IPA.

N	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
\mathcal{P} (s)	0.057	0.093	0.141	0.271	0.465	0.845	1.420	2.771	4.747	9.323	17.117
\mathcal{V} (s)	0.007	0.009	0.007	0.013	0.012	0.018	0.020	0.032	0.036	0.049	0.057
C (KB)	1.45	2.48	2.54	4.61	4.67	8.79	8.86	17.11	17.17	33.67	33.74

Table 4. Evaluation results of Hyrax-PCS. \mathcal{P} , \mathcal{V} , C, refers to proving time, verification time, and communication cost, respectively.

Since Hyrax-PCS does not require pairing operations, we used the Secp256k1 curve to implement it. We conducted the same experiments and `halo2curves` library as above under the same environment. The results are given in Tab. 4. We can observe that the verifier time of Hyrax-PCS is faster than that of Cougar, even faster than that of Leopard, in our experiment. Moreover, the communication cost is lower in all experimental settings. This is because Hyrax-PCS does not rely on pairing operations, so they do not require sending heavy group elements in \mathbb{G}_t . Nevertheless, we note that Hyrax-PCS features square root communication cost, so the communication cost increases extremely fast when N becomes large. According to our experimental result, the communication cost of Hyrax-PCS surpasses that of Leopard when $N \geq 2^{19}$, and the (expected) crossing over point between Hyrax-PCS and Cougar is $N = 2^{23}$.

In addition, we also estimate the verification cost by conducting linear regression on the data in Tab. 4, as we did for comparing IPAs. With the same notation as the previous paragraph, we obtain the regression coefficients $(\hat{\alpha}, \hat{\beta}) = (0.325, -10.650)$. Since the regression coefficient for Cougar is $(0.243, -6.465)$, i.e., having a smaller $\hat{\alpha}$ than Hyrax-PCS, we can conclude that the verification time of the proposed Cougar is asymptotically faster than that of Hyrax-PCS.