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 transparatent 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.

1 Introduction

Zero-Knowledge Proof (ZKP) is one of the privacy-enhancing technologies spotlighted by many international organizations and others [48]. ZKP is a protocol allowing a prover to convince a verifier that a statement is true without leaking any additional information [35]. ZKP schemes are employed as foundational components in various cryptographic applications, including identification [28, 23], verifiable computation [9, 11, 56, 14], range proofs [18, 25], confidential transactions [54, 18, 40, 25, 34], and incrementally verifiable computation [17, 19]. To

* Corresponding Author.

Schemes	Comm.	Prover	Verifier	Assumption	Setup	Pairing
Updatable IPA[26]	$O(\log_2 N)$	$O(N)$	$O(\log_2 N)$	DL, DPair	Trusted	Yes
Dory[45]	$O(\log_2 N)$	$O(N)$	$O(\log_2 N)$	SXDH	Trustless	Yes
Bulletproofs[13, 18]	$O(\log_2 N)$	$O(N)$	$O(N)$	DL	Trustless	No
Leopard[43, 42]	$O(\log_2 N)$	$O(N)$	$O(\sqrt{N})$	DL	Trustless	Yes
Protocol4[43]	$O(\log_2 N)$	$O(N)$	$O(\sqrt{N}\log_2 N)$	DL	Trustless	No
TENET[44]	$O(\sqrt{\log_2 N})$	$O(N \cdot 2^{\sqrt{\log_2 N}})$	$O(N/2^{\sqrt{\log_2 N}})$	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

this end, ZKP for proving the satisfiability of the arithmetic (AC) is essential, and several constructions have been proposed [39, 13, 18, 8, 32, 21, 36, 24].

Among them, one notable approach is to utilize an inner product argument (IPA) as a building block of the ZKP scheme [13, 18, 21, 45, 25]. Bootle et al. [13] first proposed an IPA with logarithmic proof size under the discrete logarithm (DL) assumption, and later, Bünz et al. [18] improved the IPA, which is called Bulletproofs. In [21], 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. [26] proposed a sublinear verifier IPA using bilinear pairing. However, the sublinear IPA [26] 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 [45] 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. [43] 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 [42], 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 [13, 18, 43]. 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 paring-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 [43]. 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 is about the elliptic curve relation. The proof is constructed by decomposing the elliptic curve relation to the arithmetic relation over the embedding field and then adapting the proof system for the arithmetic relation. In [43], they adopted a projective representation for the arithmetization of the elliptic curve operation because of the simple expression of complete addition. On the other hand, if we employ affine representation for the second layer, then the length of the commitment key for the second layer can be reduced to $2n$. In particular, [60] proposed an efficient proof for the complete addition formula between two points with affine representation using a Plonkish proof system. However, group operations in affine representation are inefficient for the prover because of costly modular inversions.

To avoid this while benefiting from the reduced $2n$-length commitment key, we present a novel Plonkish-based proof for the *mixed* addition, *i.e.*, an addition between curve points of different projective and affine representations. Our construction is based on a well-known complete addition formula [53] that is widely used in many implementations of elliptic curves, so the additional computational overhead for constructing an execution trace becomes negligible.

Plonk-Friendly Extended Polynomial Commitment Scheme. In the proof of the second layer, the prover claims knowledge of vectors and the corresponding elliptic curve relation. Constructing a proof system that satisfies both conditions is intricate because the elliptic curve relation is associated with all committed vectors. To address this, we propose a Plonk-friendly extended PCS constructed from a homomorphic PCS compatible with Plonkish proof system for elliptic curve operations. Concretely, the new PCS helps to show the consistency of 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 $O(m + \|\mathcal{V}_{\mathsf{Eval}}(n \log_2 m)\|)$, where $\|\mathcal{V}_{\mathsf{Eval}}(n \log_2 m)\|$ 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 $O(m + \|\mathcal{V}_{\mathsf{Eval}}(n \log_2 m)\|) = O(\sqrt[3]{N} + \sqrt[3]{N}\sqrt{\log_2 N})$, which is the cubic root of N.

1.2 Related Works

ZK Argument based on Discrete Logarithm Setting. Groth [38] first proposed a sublinear ZK argument for AC under the DL assumption, and Seo [55] improved it. These works also feature constant round complexity. Later works [13, 18, 21, 45, 25, 43] focus on reducing communication complexity (to logarithmic scale) rather than round complexity (allowing logarithmic complexity). Starting from Bulletproofs [13, 18], various works have been proposed to improve the verifier complexity of Bulletproofs [45, 25, 43, 27]. In a different view point, Kim et al. [43] proposed a sublogarithmic communication ZK argument for the first time, and then Lee et al. [44] enhanced it from a linear verifier cost to a sublinear one with sublogarithmic communication. Furthermore, Zhang et al. [61] proposed an efficient framework for IPA to handle arbitrary length of vectors directly, and some works [33, 59] 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, cryptographic groups with an unknown order are also considered. In this setting, Bünz et al. constructed the first transparent ZK argument with logarithmic proof size and verifier cost simultaneously [21]. Under the generic group model on

unknown order setting, Arun et al. [4] proposed a ZK argument with a constant proof size. However, both schemes feature quasi-linear prover complexity.

ZK Argument based on Lattice Setting. To overcome vulnerability against quantum computer-aided attacks, several ZK arguments from cryptographic assumptions on lattice have been proposed. Baum [5] proposed the first ZK argument for AC having sub-linear communication cost under the hardness of the short integer solution (SIS) problem. Bootle [16] proposed a plausibly post-quantum non-PCP approach ZK argument that has first poly-logarithmic communication cost and improved [5]. Lyubashevsky [47] presented a ZK argument system for the computations between integer commitments. Recently, Bünz [20] showed that the protocol of Bulletproofs [18] can be instantiated with plausibly post-quantum commitments from lattice hardness assumptions (SIS/R-SIS/M-SIS). Some other works [6, 3] presented interactive ZK proof systems under the hardness of the SIS problem or ring learning with errors (RLWE), which improved the proof size or inefficiency of the sampling technique. Although all these works have improved the communication cost up to poly-logarithm, it is still larger than ZK arguments based on the DL assumption.

ZK Argument based on Collision Resistant Hash function. In another direction for post-quantum security, ZK schemes based on cryptographic hash functions [2, 10, 8] have also been proposed. These earlier works require quasi-linear prover complexity. Later, Bootle et al. [15] proposed a tensor IOP combined with a cryptographic hash function, achieving asymptotically linear prover time. Golovnev et al. [37] improved the idea of Bootle et al. by employing a new linear-time encodable code. Recently, Orion [58] is further proposed as a system that performs better than Brakedown in terms of the soundness, proof size, and verifier's complexity, and Zhang et al. [62] optimized the works [2, 10] to reduce the prover and verifier's computation cost and the proof size.

Organization. Section 2 provides some preliminary definitions about argument of knowledge and various types of commitment schemes, such as two-tier commitments, homomorphic commitments, or polynomial commitments, and a brief overview of Plonkish. In Section 3, we revisit and generalize Protocol4 to construct our cubic root verifier IPA, Cougar. Section 4 introduces further optimization techniques for instantiating Cougar in terms of both asymptotic and concrete efficiencies. Our implementation results are provided in Section 5.

2 Preliminary

2.1 Definitions and Notations

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 $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$, which satisfies that: for any $c \in \mathbb{N}$, there exists N_c such that $\mathsf{negl}(\lambda) < 1/\lambda^c$ for all $\lambda > N_c$. 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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$. 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 $\boldsymbol{a} \in \mathbb{Z}_p^m$ and $i \in [m]$, we use a_i(non-bold style letter with a subscript i) to denote the i-th element of $\boldsymbol{a}$. We use $\|$ notation to represent concatenation of two vectors, *i.e.*, for $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^m$, $\boldsymbol{a} \parallel \boldsymbol{b} = (a_1, \ldots, a_m, b_1, \ldots, b_m)$. For $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^m$ and $\boldsymbol{G}, \boldsymbol{H} \in \mathbb{G}^m$, we use the following vector notations:

- **Component-wise addition** : $\boldsymbol{a} + \boldsymbol{b} = (a_1 + b_1, \ldots, a_m + b_m) \in \mathbb{Z}_p^m$ and $\boldsymbol{G} + \boldsymbol{H} = (G_1 + H_1, \ldots, G_m + H_m) \in \mathbb{G}^m$.
- **Component-wise product** : $\boldsymbol{a} \circ \boldsymbol{b} = (a_1 b_1, \ldots, a_m b_m) \in \mathbb{Z}_p^m$.
- **Multi-Scalar Multiplication** : $[\boldsymbol{x}]\boldsymbol{G} = \sum_{i \in [m]} [x_i] G_i \in \mathbb{G}$.
- **Inner Pairing Product** : $\boldsymbol{E}(\boldsymbol{G}, \boldsymbol{H}) = \sum_{i \in [m]} e(G_i, H_i) \in \mathbb{G}_t$, where $\boldsymbol{G} \in \mathbb{G}_1^m$ and $\boldsymbol{H} \in \mathbb{G}_2^m$.

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

Argument System for Relation $\mathcal{R}$. 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), a key generation algorithm $\mathcal{K}$, a prover algorithm $\mathcal{P}$, and a verifier algorithm $\mathcal{V}$. The $\mathcal{K}$ algorithm takes the security parameter λ and outputs CRS, 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 or 1, which means reject or accept, respectively.

Argument of Knowledge. An argument of knowledge (AoK) is a special case of an argument system that satisfies the properties of completeness and witness extractability. As previous works did [18, 43], we consider witness-extended emulation [46] for the latter, which is equivalent to knowledge soundness.

Definition 1 (Perfect Completeness). *Let $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ be an argument system and $\mathcal{R}$ be a polynomial-time verifiable relation. We say that the argument system $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ for the relation $\mathcal{R}$ has perfect completeness if, for every PPT adversary $\mathcal{A}$, the following probability equation holds:*

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

Definition 2 (Computational Witness Extended Emulation). *Let $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ be an argument system and $\mathcal{R}$ be a polynomial-time verifiable relation. We say that the argument $(\mathcal{P}, \mathcal{V})$ has computational witness-extended emulation if, for every deterministic polynomial prover $\mathcal{P}^*$, which may not follow $\mathcal{P}$, and all pairs of interactive polynomial-time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, there exists a polynomial time emulator $\mathcal{E}$, the following probability equation holds:*

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

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 $\mathcal{P}^*$, which includes randomness.

Definition 3. *We say that the argument system $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ is an argument of knowledge for relation $\mathcal{R}$ if the argument has (perfect) completeness and (computational) witness-extended emulation.*

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.

Non-interactive Argument in the Random Oracle Model. We call an interactive argument a public coin if $\mathcal{V}$ outputs without decision bits constituting a uniformly random message without dependency of $\mathcal{P}$'s messages. Fiat and Shamir [29] proposed a method to transform any public coin interactive argument into a non-interactive one using the random oracle model. The approach involves replacing $\mathcal{V}$'s random messages with random oracle outputs, where the inputs are derived from previous messages at that point.

Assumptions. Let $\mathcal{G}$ be a group generator that takes security parameters λ and then outputs $\mathbb{G}$, describing a group of order p.

Definition 4 (Discrete Logarithm Relation Assumption). *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:*

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

It is well-known that the discrete logarithm relation (DLR) assumption is equivalent to the discrete logarithm (DL) assumption [18, 43].

Definition 5 (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 $\mathcal{C} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ over a message space M, a random space R, and a commitment space C is defined by:*

- $\mathsf{Gen}(1^\lambda, \ell) \to \mathsf{ck}$ *: On input security parameter λ and dimension of message space ℓ, sample commitment key* ck
- $\mathsf{Com}(\mathsf{ck}, m; r) \to C$ *: Take commitment key* ck*, message $m \in \mathsf{M}$, and randomness $r \in \mathsf{R}$, output commitment $C \in \mathsf{C}$*
- $\mathsf{Open}(\mathsf{ck}, m, r, C) \to 0/1$ *: Take commitment key* ck*, message $m \in \mathsf{M}$, randomness $r \in \mathsf{R}$, and commitment $C \in \mathsf{C}$ output 1 if $\mathsf{Com}(\mathsf{ck}, m; r) = C$, 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 $\mathcal{C} = (\mathsf{Gen}, \mathsf{Com})$ a commitment scheme if the following properties hold:*

Binding: *For any expected PPT adversary $\mathcal{A}$,*

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

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

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

Additionally, we call a commitment scheme $\mathcal{C}$ is (additively) homomorphic if the following property holds:
(Additive) Homomorphic: *For any commitment key $\mathsf{ck} \leftarrow \mathsf{Gen}(1^\lambda, \ell)$ and pairs of message-randomness $(m_0, r_0), (m_1, r_1) \in \mathsf{M} \times \mathsf{R}$, the following equality holds:* $\mathsf{Com}(\mathsf{ck}, m_0; r_0) + \mathsf{Com}(\mathsf{ck}, m_1; r_1) = \mathsf{Com}(\mathsf{ck}, m_0 + m_1; r_0 + r_1)$

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* [49] and *AFGHO group commitment* [1].

Pedersen vector commitment. Pedersen vector commitment $\mathcal{C}_{\mathsf{Ped}} = (\mathsf{Gen}_{\mathsf{Ped}}, \mathsf{Com}_{\mathsf{Ped}})$ is a commitment scheme over message space $\mathbb{Z}_p^N$. Pedersen vector commitment provides perfect hiding and computational binding under the DL assumption. 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 $\mathcal{C}_{\mathsf{GC}} = (\mathsf{Gen}_{\mathsf{GC}}, \mathsf{Com}_{\mathsf{GC}})$ is a commitment scheme over message space $\mathbb{G}_1^N$. $\mathcal{C}_{\mathsf{GC}}$ uses a bilinear pairing for the commitment algorithm.

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

Fig. 1. Homomorphic Vector Commitment Schemes

Two-tier Commitment Scheme. 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 [22, 45, 43, 42]. A two-tier commitment scheme is constructed by combining two distinct commitment schemes $\mathcal{C}_1 = (\mathsf{Gen}_1, \mathsf{Com}_1)$ and $\mathcal{C}_2 = (\mathsf{Gen}_2, \mathsf{Com}_2)$. 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 6 (Two-tier Commitment Scheme). *Let* $\mathcal{C}_1 = (\mathsf{Gen}_1, \mathsf{Com}_1)$ *and* $\mathcal{C}_2 = (\mathsf{Gen}_2, \mathsf{Com}_2)$ *be commitment schemes over (message,commitment,randomness) space* $(\mathbb{Z}_p^n, \mathsf{C}_1, \mathsf{R}_1)$ *and* $(\mathsf{C}_1^m, \mathsf{C}_2, \mathsf{R}_2)$ *respectively. Then, the commitment scheme* $\mathcal{C} = (\mathsf{Gen}, \mathsf{Com})$ *over space* $((\mathbb{Z}_p^{m \times n}, \mathsf{C}_2, \mathsf{R}_1 \times \mathsf{R}_2)$ *is called as a two-tier commitment scheme based on* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *defined by:*

- $\underline{\mathsf{Gen}(1^\lambda, mn) \to \mathsf{ck} = (\mathsf{ck}_1, \mathsf{ck}_2)}$*:*
  1. *Run* $\mathsf{Gen}_1(1^\lambda, n) \to \mathsf{ck}_1$
  2. *Run* $\mathsf{Gen}_2(1^\lambda, m) \to \mathsf{ck}_2$
  3. *Return* $\mathsf{ck} = (\mathsf{ck}_1, \mathsf{ck}_2)$
- $\underline{\mathsf{Com}(\mathsf{ck}, M; (\boldsymbol{r}, r_f)) \to \mathrm{C}}$*:*
  1. *Compute* $\mathsf{Com}_1(\mathsf{ck}_1, M_i; r_i) \to C_i, \forall i$
  2. *Compute* $\mathsf{Com}_2(\mathsf{ck}_2, \boldsymbol{C}; r_f) \to \mathrm{C}$
  3. *Return* 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 $\mathsf{Com}(\mathsf{ck}, M)$ to describe the commitment algorithm for a message M.

Pairing-based Two-tier Commitment Scheme. From two commitment schemes $\mathcal{C}_1 = (\mathsf{Gen}_{\mathsf{Ped}}, \mathsf{Com}_{\mathsf{Ped}})$ and $\mathcal{C}_2 = (\mathsf{Gen}_{\mathsf{GC}}, \mathsf{Com}_{\mathsf{GC}})$ 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 [22, 45, 43]. 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.

Polynomial Commitment Scheme. A polynomial commitment scheme (PCS) [41, 21] 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 $\mathsf{Eval} = (\mathcal{K}, \mathcal{P}, \mathcal{V})$ for the following relation:

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

The formal definition of PCS is given as below:

Definition 7 (Polynomial Commitment Scheme). *A polynomial commitment scheme* $\mathsf{PCS} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Eval})$ *consists of key generation algorithms* Gen*, commitment algorithm* Com*, and argument system* Eval *for the relation* $\mathcal{R}_{\mathsf{Eval}}$*. We call* $\mathsf{PCS} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Eval})$ *is a polynomial commitment scheme if the following properties hold:*

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

2.2 Plonkish: Proof for Elliptic Curve Relation

In Protocol4, one of the main bottlenecks was checking the relation between elliptic curve points. More precisely, for elliptic curve points $L_i, R_i, P_{i+1}, P_i \in E(\mathbb{Z}_p)$ which are in fact commitments of corresponding messages, and a scalar $x_i \in \mathbb{Z}_p$, the relation of the form $P_{i+1} \stackrel{?}{=} [x_i^{-1}]L_i + P_i + [x_i]R_i$ should be ensured during the protocol. However, due to its construction, the commitment scheme to produce each curve point is no longer *homomorphic*, so [43] took a strategy to convert the relation into the AC. To this end, rather than using the affine coordinate representation, they attempted to represent each elliptic curve point as the projective coordinate representation, where the complete point addition formula is known [53] for prime order short Weierstrass curves.

Plonk [32] is one of the well-known methods to represent the circuit satisfiability of the given AC as the constraints system. By Lagrange interpolation, the latter can be converted to showing the equality of two polynomials, which can be proved efficiently by PIOP instantiated by PCS [21]. As shown in [31, 60], Plonk-style arithmetization can efficiently deal with *custom gates*, which are arithmetic gates beyond addition or multiplication. For example, Plonkish [60] supports custom gates and look-up operations, which is an extension of Plonk by constructing a constraint system about the execution trace for running the given AC. By utilizing Plonkish with custom gates for elliptic curve operations, as proposed by [60] for affine representation, we can handle the elliptic curve relation in Protocol4 more efficiently than the approach of [43]. The design for the custom gate and execution trace is crucial for efficiency gain. Hence, we devised new custom gates tailored for Protocol4 and constructed an execution trace using them, each of which is presented in Section 4.1.

Throughout this paper, we will denote $\mathsf{Plonkish}_{\mathsf{Eval}}$ as the proof system for Plonkish with the newly designed custom gate for mixed elliptic curve addition. Let us denote M as the number of columns in the execution trace and $\{g_i(X_1, \dots X_M)\}_{i=0}^{N_g-1}$ as the custom gates used in the proof system. Here, N_g denotes the number of types of gates. $\mathsf{Plonkish}_{\mathsf{Eval}}$ takes a commitment key $\mathsf{ck}_{\mathsf{PC}}$ for the underlying PCS, selector polynomials $\{s_i(X)\}_{i=0}^{N_g-1}$, custom 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, $\mathsf{Plonkish}_{\mathsf{Eval}}$ takes wire polynomials $\{w^{(i)}(X)\}_{i=0}^{M-1}$ that encode each column in the execution trace. Here, polynomials in the public inputs encode the configuration of the given AC, whereas each wire polynomial corresponds to the intermediate value that appeared during execution. We set the witness polynomials corresponding to each coordinate of elliptic curve points. The detailed procedure of $\mathsf{Plonkish}_{\mathsf{Eval}}$ is provided in Appendix D.

3 Construction of Cubic Root Verifier IPA

3.1 Reconstruction of Protocol4

In this section, we generalize the IPA Protocol4 [43]. Before describing the protocol, we focus on the structure of the commitment scheme used in Protocol4.

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 $\boldsymbol{a} \in \mathbb{Z}_p^{m \times n}$ is as follows: First, commit each row vector of $\boldsymbol{a}$ using Pedersen vector commitment 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. Contrary to the homomorphic commitment schemes, such as Pedersen commitment and AFGHO group commitment, the doubly-Pedersen two-tier commitment scheme does not have a homomorphic property [43]. For this reason, to apply the folding technique [13, 18] on the doubly-Pedersen commitment-based IPA, the prover should send additional proofs to ensure the validity of group operations, which are brought by Pedersen commitment in the first layer. Because the homomorphic property of second commitments helps to construct additional proofs efficiently, we prefer to use homomorphic commitment at the second layer. Additionally, the role of the second commitment is to compress a large message into single

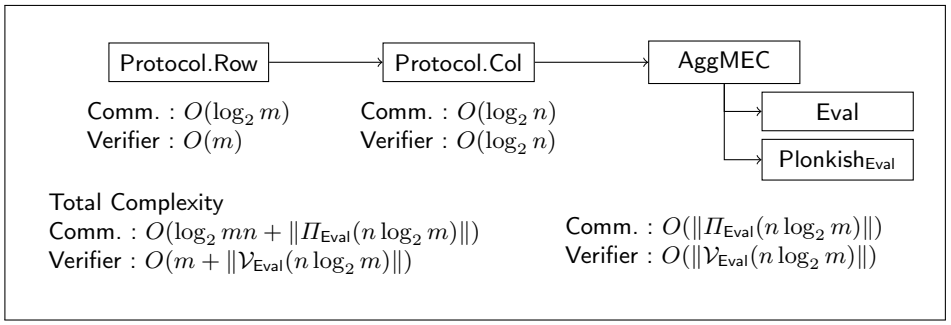

Fig. 2. Overall Process of the Protocol

commitment, e.g., from $\mathbb{Z}_q^N$ to C, so that the second commitment satisfies the *compression* property; converts a N-dimensional message into a single element.

For a precise description, let us consider the Pedersen commitment scheme $\mathcal{C}_1 = (\mathsf{Gen}_{\mathsf{Ped}}, \mathsf{Com}_{\mathsf{Ped}})$ over $(\mathbb{Z}_p^m, \mathbb{G}_p = E(\mathbb{Z}_q))$ at the first layer and a homomorphic commitment scheme $\mathcal{C}_2 = (\mathsf{Gen}_2, \mathsf{Com}_2)$ over $(\mathbb{Z}_q^{2n}, \mathsf{C})$ 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. Then, we can construct two-tier commitment scheme $\mathcal{C}_{\mathsf{TC}} = (\mathsf{Gen}_{\mathsf{TC}}, \mathsf{Com}_{\mathsf{TC}})$ as follows:

- $\mathsf{Gen}_{\mathsf{TC}}(1^\lambda, mn) \to \mathsf{ck} = (\boldsymbol{G}, \mathsf{ck}_2)$:
  1. Run $\mathsf{Gen}_{\mathsf{Ped},p}(1^\lambda, n) \to \boldsymbol{G} \in \mathbb{G}_p^n$
  2. Run $\mathsf{Gen}_2(1^\lambda, 2m) \to \mathsf{ck}_2 \in \mathbb{G}_q^{2m}$
  3. Return $\mathsf{ck} = (\boldsymbol{G}, \mathsf{ck}_2)$
- $\mathsf{Com}_{\mathsf{TC}}(\mathsf{ck} = (\boldsymbol{G}, \mathsf{ck}_2), \boldsymbol{a} \in \mathbb{Z}_p^{m\times n}) \to \mathrm{C} \in \mathbb{G}_q$:
  1. Compute $\mathsf{Com}_{\mathsf{Ped},p}(\boldsymbol{G}, \boldsymbol{a}_i) \to C_i \in \mathbb{G}_p, \forall i \in [m]$
  2. Compute $\mathsf{Com}_2(\mathsf{ck}_2, \boldsymbol{C}) \to \mathrm{C} \in \mathsf{C}$
  3. Return C

Using the commitment $\mathcal{C}_{\mathsf{TC}}$, we consider IPA for the following relation:

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

We intend to 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 $\mathcal{R}^{m,n}_{\mathsf{GenPT4}}$ to $\mathcal{R}^{m/2,n}_{\mathsf{GenPT4}}$(Row-reduction) or $\mathcal{R}^{1,n}_{\mathsf{GenPT4}}$ to $\mathcal{R}^{1,n/2}_{\mathsf{GenPT4}}$(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$_{\mathsf{Eval}}$ are used as subroutines. Notice that both have verifier complexity $\|\mathcal{V}_{\mathsf{Eval}}(n\log_2 m)\|$. We illustrate the overall process in Fig. 2.

Reduction and Store the States. 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 [13, 18] or Leopard [42], the prover and verifier store the history of reduction processes because

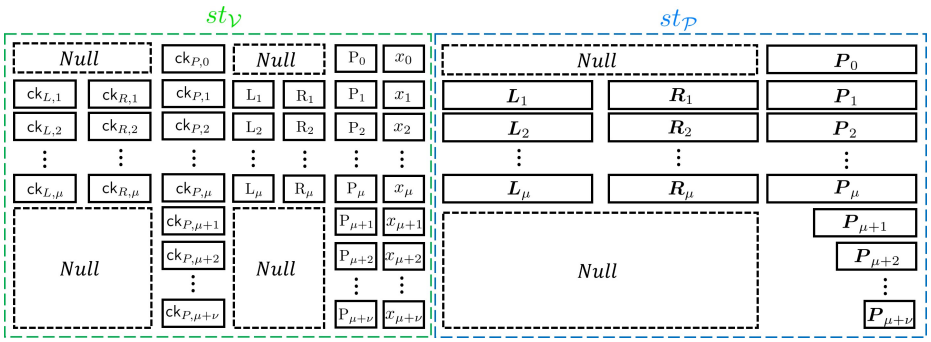

Fig. 3. Format of st_V and st_P

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, AggMEC, which guarantees the validity of the inner value of commitment for every round. We illustrate states st_V and st_P in Fig. 3. Hereafter, we denote $\mu = \log_2 m$ and $\nu = \log_2 n$.

Row-wise Reduction. In row-wise reduction, the $\mathcal{P}$ sends crossed inner product values c_L, c_R and 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 $\widehat{\mathrm{P}}$ for the next round. To resolve this issue, $\mathcal{P}$ sends an updated instance $\widehat{\mathrm{P}}$ to $\mathcal{V}$. In this phase, $\mathcal{V}$ should verify the well-construction of $\widehat{\mathrm{P}}$, but we postpone the verification of it and run the row-wise reduction recursively until $m = 1$. At $m = 1$, $\mathcal{P}$ and $\mathcal{V}$ run Protocol.Col. The description of Protocol.Row is given in Algorithm 1.

Column-wise Reduction. In column-wise reduction, the $\mathcal{P}$ sends crossed inner product values c_L and c_R. Then, $\mathcal{V}$ sends a challenge x to $\mathcal{P}$. The update process is different from that of row-wise reduction because the first commitment key is already compressed to a single element, G and H. In order to update the instance, $\mathcal{P}$ parses the vector $\boldsymbol{P}$ to 4 parts and then constructs the half-length updated vector $\widehat{\mathrm{P}}$. At the end of Protocol.Col, $\mathcal{P}$ and $\mathcal{V}$ additionally run AggMEC for knowledge of tuples of $(\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{P})$, which guarantees well-construction of $\widehat{\mathrm{P}}$ for all rounds in both row-wise and column-wise reduction. The description of Protocol.Col is given in Algorithm 2.

Theorem 1. *Assume that both* Protocol.Col *and* AggMEC *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 2 has perfect completeness and computational witness-extended emulation under the DL assumption.*

Algorithm 1 Protocol.Row

$\mathsf{Protocol.Row}(\boldsymbol{G}, \boldsymbol{H}, (\mathsf{ck}_k)_{k=s}^{\mu}, \mathsf{ck}_{\mathsf{Col}}, \mathrm{P}, c, st_V; \boldsymbol{a}, \boldsymbol{b}, st_P)$
where $\mathsf{ck}_k = (\mathsf{ck}_{L,k}, \mathsf{ck}_{R,k}, \mathsf{ck}_{P,k})$, $\mathsf{ck}_{\mathsf{Col}} = (\mathsf{ck}_{P,k})_{k=\mu+1}^{\mu+\nu+1}$
if $m = 1$, base case $s = \mu$ **then**:
  $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{Protocol.Col}(G, H, \mathsf{ck}_{\mathsf{Col}}, \mathrm{P}, c, st_V; \boldsymbol{a}, \boldsymbol{b}, st_P)$
else
  if $st_P = \perp$ and $st_V = \perp$ **then**
    $\mathcal{P}$ sets $\boldsymbol{P} = [\boldsymbol{a}]\boldsymbol{G} \parallel [\boldsymbol{b}]\boldsymbol{H}$ and adds $(\cdot, \cdot, \boldsymbol{P})$ into the bottom row of st_P.
    $\mathcal{V}$ adds $(\mathsf{ck}_{P,0}, \cdot, \cdot, \mathrm{P}, \cdot)$ into the bottom row of st_V.
  else
    $\mathcal{P}$ refers $\boldsymbol{P}$ in the bottom row of st_P
  end if
Set $\widehat{m} = \frac{m}{2}$ and $\boldsymbol{a} = [\boldsymbol{a}_L \| \boldsymbol{a}_R]$, $\boldsymbol{b} = [\boldsymbol{b}_L \| \boldsymbol{b}_R]$, $\boldsymbol{G} = \boldsymbol{G}_L \| \boldsymbol{G}_R$, $\boldsymbol{H} = \boldsymbol{H}_L \| \boldsymbol{H}_R$
  $\mathcal{P}$ computes c_L, c_R and L, R and sends them to $\mathcal{V}$:
  $\boldsymbol{L} = [\boldsymbol{a}_L]\boldsymbol{G}_R \parallel [\boldsymbol{b}_R]\boldsymbol{H}_L$, $\boldsymbol{R} = [\boldsymbol{a}_R]\boldsymbol{G}_L \parallel [\boldsymbol{b}_L]\boldsymbol{H}_R \in \mathbb{G}_p^{2n}$,
  $c_L = \langle \boldsymbol{a}_L, \boldsymbol{b}_R \rangle$, $c_R = \langle \boldsymbol{a}_R, \boldsymbol{b}_L \rangle \in \mathbb{Z}_p$,
  $\mathrm{L} = \mathsf{Com}_2(\mathsf{ck}_{L,s}, \boldsymbol{L})$, $\mathrm{R} = \mathsf{Com}_2(\mathsf{ck}_{R,s}, \boldsymbol{R}) \in \mathbb{G}_q$
  $\mathcal{V}$ chooses $x \xleftarrow{\$} \mathbb{Z}_p^*$ and returns it to $\mathcal{P}$
  $\mathcal{P}$ computes $\widehat{\mathrm{P}}$ and sends it to $\mathcal{V}$:
  $\widehat{\boldsymbol{P}} = [x^{-1}]\boldsymbol{L} + \boldsymbol{P} + [x]\boldsymbol{R} \in \mathbb{G}_p^{2n}$, $\widehat{\mathrm{P}} = \mathsf{Com}_2(\mathsf{ck}_{P,s}, \widehat{\boldsymbol{P}}) \in \mathbb{G}_q$
  Both $\mathcal{P}$ and $\mathcal{V}$ update:
  $\widehat{\boldsymbol{G}} = \boldsymbol{G}_L + [x^{-1}]\boldsymbol{G}_R$, $\widehat{\boldsymbol{H}} = \boldsymbol{H}_L + [x]\boldsymbol{H}_R \in \mathbb{G}_p^{\widehat{m}}$, $\widehat{c} = x^{-1}c_L + c + xc_R \in \mathbb{Z}_p$
  $\mathcal{P}$ updates $\widehat{\boldsymbol{a}} = \boldsymbol{a}_L + x\boldsymbol{a}_R$, $\widehat{\boldsymbol{b}} = \boldsymbol{b}_L + x^{-1}\boldsymbol{b}_R \in \mathbb{Z}_p^{\widehat{m} \times n}$.
  $\mathcal{V}$ adds $(\mathsf{ck}_s, \mathrm{L}, \mathrm{R}, \widehat{\mathrm{P}}, x)$ into the bottom row of st_V.
  $\mathcal{P}$ adds $(\boldsymbol{L}, \boldsymbol{R}, \widehat{\boldsymbol{P}})$ into the bottom row of st_P.
  Both $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{Protocol.Row}(\widehat{\boldsymbol{G}}, \widehat{\boldsymbol{H}}, (\mathsf{ck}_k)_{k=s+1}^{\mu}, \mathsf{ck}_{\mathsf{Col}}, \widehat{\mathrm{P}}, \widehat{c}, st_V; \widehat{\boldsymbol{a}}, \widehat{\boldsymbol{b}}, st_P)$
end if

Due to space constraints, we defer the full proof to Appendix. A and B.

Proof of Multi-Elliptic Curve Operation: Algorithm 3. We now explain how to construct multi-elliptic curve operation arguments AggMEC. Contrary to [43], we unify and aggregate row-wise and column-wise multi-elliptic curve operation proofs into a single protocol. That is, AggMEC guarantees the well-constructed updated instances $\widehat{\mathrm{P}}$ from every round of both row-wise and column-wise reductions. Concretely, AggMEC checks that the k-th row of state tuples $(st_V; st_P)_k = (\mathsf{ck}_k, (\mathrm{L}_k, \mathrm{R}_k, \mathrm{P}_k, x_k); (\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k))$ satisfy the following:

1. Commitment

$$\begin{aligned} &\mathrm{L}_k = \mathsf{Com}_2(\mathsf{ck}_{L,k}, \boldsymbol{L}_k), \mathrm{R}_k = \mathsf{Com}_2(\mathsf{ck}_{R,k}, \boldsymbol{R}_k) \text{ for } k = 1, \ldots, \mu \\ &\mathrm{P}_k = \mathsf{Com}_2(\mathsf{ck}_{P,k}, \boldsymbol{P}_k) \text{ for } k = 0, \ldots, \mu + \nu - 1 \\ &\mathrm{P}_{\mu+\nu} = \mathsf{Com}_2(\mathsf{ck}, [a]G \parallel [b]H) \end{aligned} \tag{3}$$

Algorithm 2 Protocol.Col

$\mathsf{Protocol.Col}(G, H, (\mathsf{ck}_{P,k+\mu})_{k=s}^{\nu}, \mathrm{P}, c, st_V; \boldsymbol{a}, \boldsymbol{b}, st_P)$

if $n = 1$, base case $s = \nu$ **then**:
  $\mathcal{P}$ sends a and b to $\mathcal{V}$
  $\mathcal{V}$ checks $c \stackrel{?}{=} a \cdot b$ and set $\boldsymbol{P}_{\mathsf{Pub}} = [a]G \parallel [b]H \in \mathbb{Z}_q^4$
  $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{AggMEC}(\boldsymbol{P}_{\mathsf{Pub}}, st_V; st_P)$
else
  if $st_P = \perp$ and $st_V = \perp$ **then**
    $\mathcal{P}$ sets $\boldsymbol{P} = [\boldsymbol{a}]G \parallel [\boldsymbol{b}]H$ and adds $(\boldsymbol{P})$ into the bottom row of st_P
    $\mathcal{V}$ adds $(\mathsf{ck}_{P,\mu}, \mathrm{P}, \cdot)$ into the bottom row of st_V.
  else
    $\mathcal{P}$ refers $\boldsymbol{P}$ in the bottom row of st_P
  end if
Set $\widehat{n} = \frac{n}{2}$ and $\boldsymbol{a} = \boldsymbol{a}_L \| \boldsymbol{a}_R$, $\boldsymbol{b} = \boldsymbol{b}_L \| \boldsymbol{b}_R$, $\boldsymbol{P} = \boldsymbol{P}^{(q_1)} \parallel \boldsymbol{P}^{(q_2)} \parallel \boldsymbol{P}^{(q_3)} \parallel \boldsymbol{P}^{(q_4)}$
  $\mathcal{P}$ computes c_L and c_R and sends them to $\mathcal{V}$:
  $c_L = \langle \boldsymbol{a}_L, \boldsymbol{b}_R \rangle \in \mathbb{Z}_p, \quad c_R = \langle \boldsymbol{a}_R, \boldsymbol{b}_L \rangle \in \mathbb{Z}_p.$
  $\mathcal{V}$ chooses $x \xleftarrow{\$} \mathbb{Z}_p^*$ and returns it to $\mathcal{P}$
  $\mathcal{P}$ computes $\widehat{\mathrm{P}}$ and sends it to $\mathcal{V}$:
  $\widehat{\boldsymbol{P}} = (\boldsymbol{P}^{(q_1)} + [x]\boldsymbol{P}^{(q_2)} \parallel \boldsymbol{P}^{(q_3)} + [x^{-1}]\boldsymbol{P}^{(q_4)}) \in \mathbb{G}_p^{2\widehat{n}}$, $\widehat{\mathrm{P}} = \mathsf{Com}_2(\mathsf{ck}_{P,\mu+s}, \widehat{\boldsymbol{P}}) \in \mathbb{G}_q$
  Both $\mathcal{P}$ and $\mathcal{V}$ compute $\widehat{c} = x^{-1}c_L + c + xc_R \in \mathbb{Z}_p$
  Additionally, $\mathcal{P}$ computes $\widehat{\boldsymbol{a}} = \boldsymbol{a}_L + x\boldsymbol{a}_R$, $\widehat{\boldsymbol{b}} = \boldsymbol{b}_L + x^{-1}\boldsymbol{b}_R \in \mathbb{Z}_p^{\widehat{n}}$.
  $\mathcal{V}$ adds $(\mathsf{ck}_{P,\mu+s}, \widehat{\mathrm{P}}, x)$ into the bottom row of st_V.
  $\mathcal{P}$ adds $(\widehat{\boldsymbol{P}})$ into the bottom row of st_P.
  Both $\mathcal{P}$ and $\mathcal{V}$ run $\mathsf{Protocol.Col}(G, H, (\mathsf{ck}_{P,k+\mu})_{k=s+1}^{\nu}, \widehat{\mathrm{P}}, \widehat{c}, st_V; \widehat{\boldsymbol{a}}, \widehat{\boldsymbol{b}}, st_P)$
end if

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

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

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

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 $\boldsymbol{L}_k$, $\boldsymbol{R}_k$, and $\boldsymbol{P}_k$ of L_k, R_k, and P_k. Since each $\boldsymbol{L}_k$, $\boldsymbol{R}_k$ consist of $2n$ elements for all $k \in [\mu]$, and $\boldsymbol{P}_k$ consists of $2n$ elements for $k = 0, \ldots \mu$ and $n/2^{k-\mu-1}$ for all $k = \mu + 1 \ldots \mu + \nu$, 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

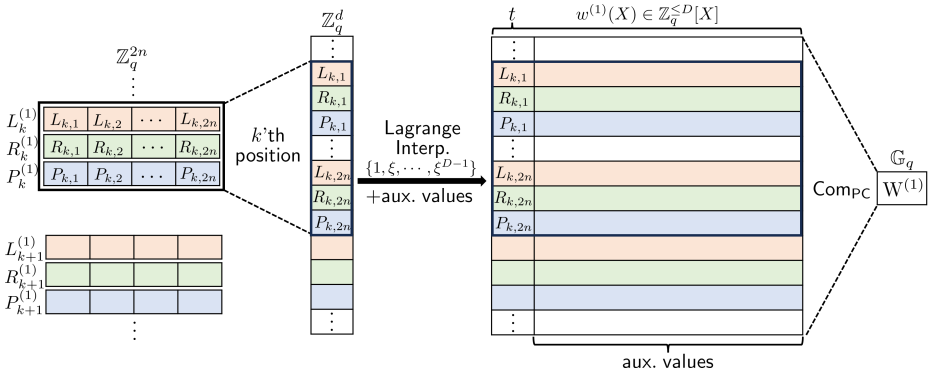

Fig. 4. Structure of Wire Polynomial. Best viewed in color.

$4n\mu+4n-2$ 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.

Plonk-Friendly Extended Polynomial Commitment Scheme. To prove the consistency of the wire polynomial and commitments $\mathrm{L}_k, \mathrm{R}_k, \mathrm{P}_k$, we construct a commitment scheme for the message vectors $\boldsymbol{L}_k, \boldsymbol{R}_k$ and $\boldsymbol{P}_k$ considering compatibility with the polynomial commitment scheme. To this end, we first encode vectors $\boldsymbol{L}_k, \boldsymbol{R}_k$ and $\boldsymbol{P}_k$ into polynomials $F_{L,k}, F_{R,k}, F_{P,k}$ and then commit them. The encoding function $\mathsf{Enc}_{\mathsf{type}}$ takes ξ, index k and a vector $\boldsymbol{a}$, returning a polynomial $F_{\mathsf{type},k}$ in $\mathbb{Z}_q[X]$, where $\mathsf{type} \in \{L, R, P\}$. 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 $(\mathsf{type}_1, k_1) \neq (\mathsf{type}_2, k_2)$, $F_{\mathsf{type}_1,k_1}(\xi^i)F_{\mathsf{type}_2,k_2}(\xi^i) = 0$ holds for all $i \in [D]$. In our setting, decoding of a polynomial $F_{\mathsf{type},k}$ can be performed uniquely when the type type and position k are determined. Furthermore, the sum of two encoded functions preserves their original non-zero evaluations at ξ^i.

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

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

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

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

Algorithm 3 AggMEC

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

– $\mathsf{Enc}_P(\xi, k \in \{0, \ldots, \mu + \nu + 1\}, \boldsymbol{a} \in \mathbb{Z}_q^{\leq 2n}) \to F_{P,k} \in \mathbb{Z}_q[X]$
Construct degree D polynomial $F_{P,k}(X)$ such that:

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

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

– $\mathsf{Com}_2(\mathsf{ck}_{\mathsf{type},k}, \boldsymbol{a} = (\boldsymbol{a}^{(1)}, \boldsymbol{a}^{(2)}) \in \mathbb{Z}_q^{4n}) \to \mathrm{A}$
1. $\mathsf{Enc}_{\mathsf{type}}(\xi, k, \boldsymbol{a}^{(i)}) \to F_{\mathsf{type},k}^{(i)}$ for $i \in \{1, 2\}$
2. $\mathsf{Com}_{\mathsf{PC}}(\mathsf{ck}_{\mathsf{PC}}, F_{\mathsf{type},k}^{(i)}) \to \mathrm{A}^{(i)}$ for $i \in \{1, 2\}$
3. Output $\mathrm{A} = (\mathrm{A}^{(1)}, \mathrm{A}^{(2)})$

Designated Execution Table. The execution table contains all values $\{\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k\}$ at some position. To construct AggMEC, we allocate each value $\{\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k\}$ at a specific position in two wire polynomials $w^{(1)}(X), w^{(2)}(X)$ following polynomial encoding Enc. Then the wire polynomials $w^{(1)}(X)$ contains x-coordinate of of the curve points in the affine representation as encoded values. Also, $w^{(2)}(X)$

contains y-coordinates of the curve points of the curve points. Thus, the non-zero evaluation of the encoding polynomials $F^{(1)}_{\mathsf{type},k}$ and $F^{(2)}_{\mathsf{type},k}$ at ξ^i is equal to the evaluation of the wire polynomials $w^{(1)}(X)$ and $w^{(2)}(X)$ for all k and type, respectively. Note that in this section, $w(X)$ denotes $w^{(1)}(X)$ and $w^{(2)}(X)$ as an integrated notation for convenience.

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 $\mathsf{Plonkish}_{\mathsf{Eval}}$, we now focus on checking Eq. (3) and the consistency between $\mathrm{L}_k, \mathrm{R}_k, \mathrm{P}_k$ and the wire polynomial of the execution trace. First, to ensure consistency of $\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k$, we first merge every commitment $\mathrm{L}_k, \mathrm{R}_k, \mathrm{P}_k$ to one commitment A, whose message polynomial is the sum of encoding polynomials, $a(X) = \sum F_{\mathsf{type},k}(X)$. 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 $\mathrm{L}_k, \mathrm{R}_k, \mathrm{P}_k$ corresponds to the commitment of $\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k$, respectively. Through the randomness ρ chosen by the verifier, these commitments and the encoding polynomials of $\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k$ 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.

Theorem 3. *Assume that a polynomial commitment scheme* $\mathsf{PCS} = (\mathsf{Gen}, \mathsf{Com}_{\mathsf{PC}}, \mathsf{Eval})$ *satisfies all properties of Definition 7 and the homomorphic property. Then,* AggMEC *in Algorithm 3 has perfect completeness and computational witness-extended-emulation.*

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

3.2 Cougar: Cubic Root Verifier IPA

From the above construction, we adopt the homomorphic polynomial commitment scheme $\mathsf{Leopard}_{\mathsf{PC}}$ in place of $\mathsf{Com}_{\mathsf{PC}}$. We call this IPA Cougar. The description of $\mathsf{Leopard}_{\mathsf{PC}}$ is given in Appendix E.

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

1. Row-reduction, Algorithm 1

- `[Prover Cost]`: For commitments L, R and $\widehat{\mathrm{P}}$ 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 $\widehat{\boldsymbol{G}}, \widehat{\boldsymbol{H}}$ and $\widehat{\boldsymbol{a}}, \widehat{\boldsymbol{b}}, \widehat{c}$ 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 $\widehat{\boldsymbol{G}}, \widehat{\boldsymbol{H}}$ and $\widehat{c}$ 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, $\widehat{\mathrm{P}}$, 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 2

– [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 $\widehat{\mathrm{P}}$, $\widehat{\boldsymbol{a}}$, $\widehat{\boldsymbol{b}}$, and $\widehat{c}$ 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 $\widehat{c}$ 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 $\widehat{\mathrm{P}}$, 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 3

– [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 $\mathsf{Plonkish}_{\mathsf{Eval}}$ is $O(\|\mathcal{P}_{\mathsf{Eval}}(D)\|)$. Then, total prover cost is $O(n\log_2 N + \|\mathcal{P}_{\mathsf{Eval}}(D)\|)$.

– [Verifier Cost]: From line 1 to 11, $\mathcal{V}$ computes $O(\log_2 N)$ $\mathbb{G}_q$ operations. And the cost of Eval and $\mathsf{Plonkish}_{\mathsf{Eval}}$ is $O(\|\mathcal{V}_{\mathsf{Eval}}(D)\|)$. Then the total verifier cost is $O(\log_2 N + \|\mathcal{V}_{\mathsf{Eval}}(D)\|)$.

– [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 $\mathsf{Plonkish}$ the prover sends $O(\|\Pi_{\mathsf{Eval}}(D)\|)$. Then total communication cost is $O(\|\Pi_{\mathsf{Eval}}(D)\|)$

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 $\mathsf{Leopard}$ features $(\|\mathcal{P}_{\mathsf{Eval}}(D)\|, \|\mathcal{V}_{\mathsf{Eval}}(D)\|, \|\Pi_{\mathsf{Eval}}(D)\|) = (O(D), O(\sqrt{D}), O(\log_2 D))$, 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 [42], Cougar satisfies perfect completeness and computational witness-extended-emulation under the DL assumption. □

4 Cougar-Friendly Constraint System and Optimizations

We present a Plonkish-type constraint system tailored for proving elliptic curve relations by introducing novel custom gates. In addition, we provide several optimization techniques for instantiating Cougar, including batching techniques for some dominant operations and a fine-grained parameter selection for concrete efficiency. Throughout this section, we denote the degree of an operation as the depth of the arithmetic circuit with respect to multiplication gates.

4.1 Efficient Constraint System for Elliptic Curve Relations

Custom Gate for Projective-Affine Mixed Operations. To ensure the elliptic curve relations Eq. (4) and (5), the prover should construct an execution trace consisting of elliptic curve additions and doublings. As presented in halo2 [60], the custom gates for them on the affine representation would reduce the number of input gates. However, many implementations of elliptic curve groups use group operations on projective coordinates [53] 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 [53]. 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 $y^2 = x^3 + b$[1], the mixed addition $P + Q = (X_3, Y_3, Z_3)$ can be computed by

$$\begin{aligned} X_3 &= (X_1Y_2 + X_2Y_1)(Y_1Y_2 - 3bZ_1) - 3b(Y_1 + Y_2Z_1)(X_1 + X_2Z_1), \\ Y_3 &= (Y_1Y_2 + 3bZ_1)(Y_1Y_2 - 3bZ_1) + 9bX_1X_2(X_1 + X_2Z_1), \\ Z_3 &= (Y_1 + Y_2Z_1)(Y_1Y_2 + 3bZ_1) + 3X_1X_2(X_1Y_2 + X_2Y_1). \end{aligned}$$

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

$$\begin{aligned} X_3 &= 2X_1Y_1(Y_1^2 - 9bZ_1^2), \\ Y_3 &= (Y_1^2 - 9bZ_1^2)(Y_1^2 + 3bZ_1^2) + 24bY_1^2Z_1^2, \\ Z_3 &= 8Y_1^3Z_1. \end{aligned}$$

[1] Of course, [53] provided general results for curves of the form $y^2 = x^3 + ax + b$. We presented a special case for the sake of efficiency.

Algorithm 4 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 $k \in \mathbb{Z}_q$.
Initialize $R \leftarrow (0, 1, 0)$
Compute $(k_1, \ldots, k_{\lceil \log_2 q \rceil}) \in \{0,1\}^{\lceil \log_2 q \rceil}$ such that $k = \sum_{i=1}^{\lceil \log_2 q \rceil} k_i 2^{i-1}$.
For i from $\lceil \log_2 q \rceil$ to 0 do:
    Update $R \leftarrow [2]R + [k_i]P$.
Return R.

Note that these operations are of degree 4, and [53] 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 4. 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. 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 $\boldsymbol{L}_i$, $\boldsymbol{R}_i$, and $\boldsymbol{P}_i$ 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,

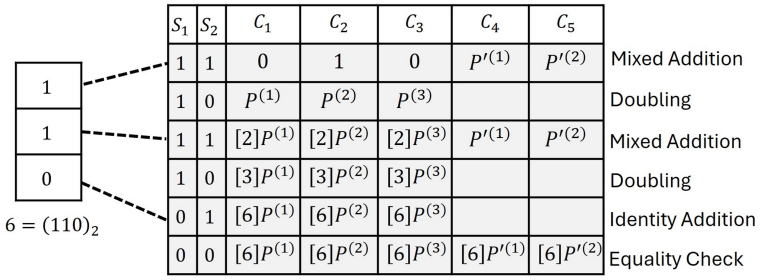

Fig. 5. Description of the execution trace for computing [6]P.

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. 5. In this figure, $[k]P^{(i)}$ means the ith 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 [53, 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 $\boldsymbol{L}$, $\boldsymbol{R}$, and $\boldsymbol{P}$ 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 $\boldsymbol{P}_{k+1}$, $\boldsymbol{L}_{k+1}$, $\boldsymbol{P}_k$, and $\boldsymbol{R}_{k+1}$, 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 $R_{k+1,i}$ 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 $\boldsymbol{P}_{k+1}$, $\boldsymbol{P}_k^{(q_1)}$, $\boldsymbol{P}_k^{(q_2)}$, $\boldsymbol{P}_k^{(q_3)}$, and $\boldsymbol{P}_k^{(q_4)}$, respectively. Here, we assumed that the length of $\boldsymbol{P}_{k+1}$ is $2l$ and that for each $\widetilde{\boldsymbol{P}_k^{(q_j)}}$ 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 $\boldsymbol{L}_k, \boldsymbol{R}_k$, and $\boldsymbol{P}_k$, 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 $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$ 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 $\boldsymbol{P}_k$ 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 $\boldsymbol{L}_k$, $\boldsymbol{R}_k$, and $\boldsymbol{P}_k$ on the wire polynomial. Thus, we place each $\boldsymbol{P}_k$ in the column reduction with the same period as $\boldsymbol{L}_k$, $\boldsymbol{R}_k$, and $\boldsymbol{P}_k$ 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 $\boldsymbol{L}_k, \boldsymbol{R}_k$, and $\boldsymbol{P}_k$ in the same order of their index to simplify the indexing. For the row reduction, we placed input vectors $\boldsymbol{L}_{k+1}, \boldsymbol{R}_{k+1}$, and $\boldsymbol{P}_k$ 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 $\boldsymbol{P}_{k+1}$ of the process with the same period. However, in this case, the output $\boldsymbol{P}_{\mu+1}$ 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 $\boldsymbol{P}_{\mu+1}$ 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 $\boldsymbol{L}_i$, $\boldsymbol{R}_i$, and $\boldsymbol{P}_i$, *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. 6. 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.

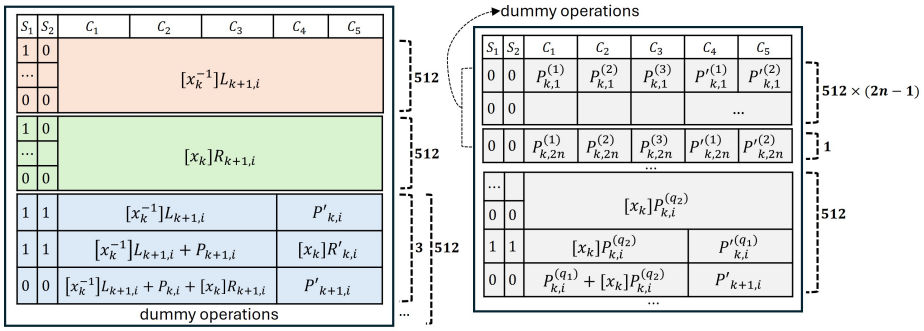

Fig. 6. Final construction of the execution trace.

4.2 More Optimization Tricks

Batched Execution of $\mathsf{Leopard_{PC}.Eval}$. During AggMEC, the prover needs to execute $\mathsf{Leopard_{PC}.Eval}$ on several polynomials and various evaluation points. In fact, by employing the techniques used in [17, 12], these processes can be done in a batched manner. To convince the verifier that the evaluations of several polynomials $f_1(X), \ldots, f_k(X)$ at different evaluation points $u_1, \ldots, u_k$ are $v_1, \ldots, v_k$, respectively, it suffices to show the existence of polynomials $q_1(X), \ldots, q_k(X)$ such that $f_i(X) - v_i = q_i(X)(X - u_i)$. Following the idea of [17, 12], 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 $f_1(X), \ldots, f_k(X)$, and $\widehat{q}(X)$ at another random point r_2 provided by the verifier, sending $w_1 := f_1(r_2), \ldots, w_k := f_k(r_2)$ and $w_{k+1} := \widehat{q}(r)$ 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 $\mathsf{Leopard_{PC}.Eval}$ 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 $\mathsf{Leopard_{PC}}$ is a homomorphic commitment, the commitment of $\widehat{f}$ can be computed from those of $f_1, \ldots, f_k$ and $\widehat{q}$, *i.e.*, the verifier can be computed itself.

Thus, only a single execution of $\mathsf{Leopard_{PC}.Eval}$ 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 3.2, 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.

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

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 $\mathsf{Protocol.Col}$ through $2\log_2 N$ field operations, (2) computing $\mathrm{A}^{(1)}$, $\mathrm{A}^{(2)}$, and V through multi-scalar multiplication of length $O(\log_2 N)$ each on $\mathbb{G}_t$ during AggMEC, (3) running several times of $\mathsf{Leopard_{PC}.Eval}$ during $\mathsf{Plonkish_{Eval}}$, and (4) checking the output of the AC for $\mathsf{Plonkish}$ 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 $\mathsf{Leopard_{PC}.Eval}$ 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 $\mathsf{Leopard_{PC}.Eval}$ 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 $\mathfrak{c}_p$, $\mathfrak{c}_1$, and $\mathfrak{c}_2$, respectively, finding the optimal parameter is equivalent to solving the following optimization problem:

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

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

$$\widehat{C} = A \cdot \frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2}\sqrt{\frac{B}{m}} + A \cdot \frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2}\sqrt{\frac{B}{m}} + \mathfrak{c}_p \cdot 2m. \tag{6}$$

Hence, applying the AM-GM inequality in Eq. (6) 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

$$m = \sqrt[3]{\frac{A^2 B}{4}\left(\frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2\mathfrak{c}_p}\right)^2} = \sqrt[3]{\left(\frac{\mathfrak{c}_1 + \mathfrak{c}_2}{2\mathfrak{c}_p}\right)^2 (3N\log_2 N + 2N)\log_2 q}. \tag{7}$$

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

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 [50].

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 $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$ 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 $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$, 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 $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$. Hence, the verifier can postpone these multiscalar multiplications and merge them to the last part of $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$. Moreover, note that V and $[\tau]\mathrm{A}^{(1)} + [\tau^2]\mathrm{A}^{(2)}$ share the same base group points: $\mathrm{L}_i^{(\mathsf{c})}, \mathrm{R}_i^{(\mathsf{c})}$, and $\mathrm{P}_i^{(\mathsf{c})}$ for $\mathsf{c} \in \{1, 2\}$. 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 $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$.

5 Implementation Results

We implemented Cougar with a famous half-pairing cycle of curves: BN254 and Grumpkin. For PCS, we utilize $\mathsf{Leopard}_{\mathsf{PC}}$ instantiated with BN254. We used the Fiat-Shamir transformation [29] 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.

5.1 Parameter Selection

For selecting m, n from the given N, we used the formula introduced in Eq. (7) with measuring coefficients $\mathfrak{c}_p$, $\mathfrak{c}_1$, and $\mathfrak{c}_2$ 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 $\mathfrak{c}_p = 1$, we obtain $\mathfrak{c}_1 = 1.0$ and $\mathfrak{c}_2 = 3.2$ 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.

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. (7).

5.2 Evaluation Results

We implemented Cougar by following the algorithms described in the paper, with optimization tricks in Section 4.2. For elliptic curve operations in BN254 and Grumpkin, we used `halo2curves` crate [51] of version 0.6.1. In addition, we utilized some parts of `halo2_proofs` crate [52] of version 0.3.0 to deal with polynomials during AggMEC. For more detailed information, we recommend the reader refer to our source code[2].

For comparison, we also implemented BulletProofs [18] and Leopard [43, 42] 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 [30] 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 [42].

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 N from 2^{10} to 2^{20}. The evaluation results for each IPA are presented and visualized in Table 3 and Fig. 7, 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.

[2] `https://github.com/Cryptology-Algorithm-Lab/Cougar`

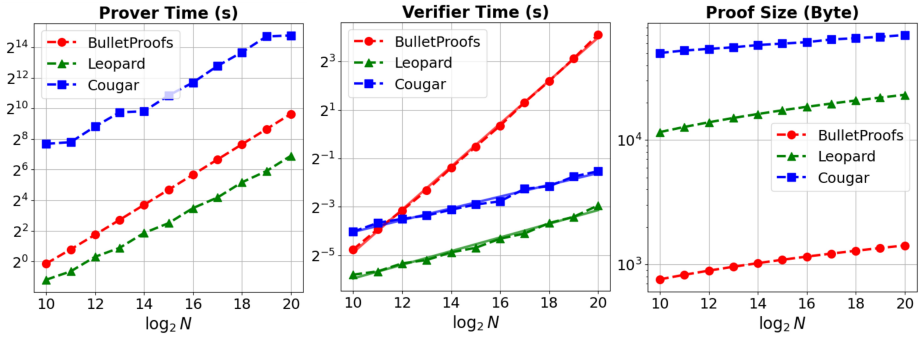

Fig. 7. Evaluation results of each IPA. We present log-log plots of each quantity for various N from 2^{10} to 2^{20}. The solid line in the verification time indicates the linear regression for each IPA in log-log plot. Best viewed in color.

N	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}.

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 mutliscalar 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. 7.

6 Concluding Remarks

We presented Cougar, the first cubic root verifier, logarithmic size proof IPA under the DL assumption and transparent setup. Our construction is based on two square root verifier IPAs, Protocol3 and Protocol4, by smoothly combining them with novel techniques for proving relations on elliptic curve points. In particular, we introduced a novel custom gate for mixed point addition, which is of independent interest. Through experiments, we demonstrated that the verification cost of Cougar increases much slower than two previous IPAs, BulletProofs and Leopard, under the same DL assumption and transparent setup.

We leave some challenges for improving Cougar. First, although the proposed Cougar improved the asymptotic complexity over prior works, it shows worse concrete efficiency for a practical choice of N. Although our results on linear regression indicate that the verification cost of Cougar becomes cheaper than Leopard when $\log_2 N > 60$, making this crossing point smaller would be a challenging yet important problem. In addition, since Cougar utilized Plonkish, it cannot handle a large N where the corresponding D exceeds the allowed number of roots of unity, *e.g.,* 2^{28} in BN254. To mitigate this, employing other constraint systems, such as SuperSpartan [57] or HyperPlonk [24], that do not rely on Lagrange interpolation would be possible. We leave it to future work. Finally, the proving cost of Cougar is too expensive compared to previous IPAs. We suspect that this is because of the expansion of the committed vectors led by Plonk-friendly encoding during Protocol.Row and Protocol.Col. Mitigating this would significantly improve the prover's cost, and we also leave it as future work.

References

1. Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. *J. Cryptol.*, 29(2):363–421, 2016.
2. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM CCS 2017*, pages 2087–2104. ACM, 2017.
3. Xavier Arnal, Abraham Cano, Tamara Finogina, and Javier Herranz. How to avoid repetitions in lattice-based deniable zero-knowledge proofs. In *Secure IT Systems*, pages 253–269. Springer, 2022.
4. Arasu Arun, Chaya Ganesh, Satya V. Lokam, Tushar Mopuri, and Sriram Sridhar. Dew: Transparent constant-sized zksnarks. Cryptology ePrint Archive, Report 2022/419, 2022. `https://eprint.iacr.org/2022/419.pdf`.
5. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafael del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *Advances in Cryptology – CRYPTO 2018*, pages 669–699. Springer, 2018.
6. Carsten Baum and Ariel 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.

7. Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, 2012.
8. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Annual International Cryptology Conference*, pages 701–732. Springer, 2019.
9. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO 2013*, volume 8043 of *LNCS*, pages 90–108. Springer, 2013.
10. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P Ward. Aurora: Transparent succinct arguments for r1cs. In *EUROCRYPT 2019*, volume 11476 of *LNCS*, pages 103–128. Springer, 2019.
11. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security 2014*, pages 781–796, 2014.
12. Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. *Cryptology ePrint Archive*, 2020.
13. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe 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.
14. Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary 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.
15. Jonathan Bootle, Alessandro Chiesa, and Jens 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.
16. Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. A non-pcp approach to succinct quantum-safe zero-knowledge. In *Advances in Cryptology – CRYPTO 2020*, pages 441–469. Springer, 2020.
17. Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.
18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy 2018*, pages 315–334. IEEE Computer Society, 2018.
19. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In *TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *LNCS*, pages 1–18. Springer, 2020.
20. Benedikt Bünz and Ben Fisch. Multilinear schwartz-zippel mod n and lattice-based succinct arguments. In *Theory of Cryptography Conference*, pages 394–423. Springer, 2023.
21. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 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.

22. Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi 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.
23. Mike Burmester, Yvo Desmedt, and Thomas Beth. Efficient zero-knowledge identification scheme for smart cards. *Comput. J.*, 35(1):21–29, 1992.
24. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei 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.
25. Heewon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger. *IEEE Access*, 10:42067–42082, 2022.
26. Vanesa Daza, Carla Ràfols, and Alexandros Zacharakis. Updateable inner product argument with logarithmic verifier and applications. In *PKC 2020*, volume 12110 of *LNCS*, pages 527–557. Springer, 2020.
27. Moumita Dutta, Chaya Ganesh, and Neha 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.
28. Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *J. Cryptol.*, 1(2):77–94, 1988.
29. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO 1986*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.
30. Filecoin. blstrs, 2024. `https://github.com/filecoin-project/blstrs`.
31. Ariel Gabizon and Zachary J Williamson. Proposal: The turbo-plonk program syntax for specifying snark programs, 2020.
32. Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. `https://eprint.iacr.org/2019/953.pdf`.
33. Shang Gao, Zhe Peng, Feng Tan, Yuanqing Zheng, and Bin Xiao. Symmeproof: Compact zero-knowledge argument for blockchain confidential transactions. *IEEE Transactions on Dependable and Secure Computing*, 2022.
34. Shang Gao, Zhe Peng, Feng Tan, Yuanqing Zheng, and Bin Xiao. Symmeproof: Compact zero-knowledge argument for blockchain confidential transactions. *IEEE Transactions on Dependable and Secure Computing*, 20(3):2289–2301, 2023.
35. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.
36. Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S Wahby. Brakedown: Linear-time and field-agnostic snarks for r1cs. In *Annual International Cryptology Conference*, pages 193–226. Springer, 2023.
37. Alexander Golovnev, Jonathan Lee, Srinath TV Setty, Justin Thaler, and Riad S Wahby. Brakedown: Linear-time and post-quantum snarks for r1cs. *IACR Cryptol. ePrint Arch.*, 2021:1043, 2021.
38. Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO 2009*, volume 5677 of *LNCS*, pages 192–208. Springer, 2009.
39. Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016.

40. Aram Jivanyan. Lelantus: Towards confidentiality and anonymity of blockchain transactions from standard assumptions. Cryptology ePrint Archive, Report 2019/373, 2019.
41. 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.
42. Sungwook Kim, Gwangwoon Lee, Hyeonbum Lee, and Jae Hong Seo. Leopard: Sublinear verifier inner product argument under discrete logarithm assumption. *IEEE Transactions on Information Forensics and Security*, 18:5332–5344, 2023.
43. Sungwook Kim, Hyeonbum Lee, and Jae Hong 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.
44. Hyeonbum Lee and Jae Hong 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.
45. Jonathan 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.
46. Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, 2003.
47. Vadim Lyubashevsky and Ngoc Khanh Nguyen. Practical lattice-based zero-knowledge proofs for integer relations. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1051–1070, 2020.
48. OECD. Emerging privacy-enhancing technologies. *OECD Digital Economy Papers*, (351), 2023.
49. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.
50. Nicholas Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250, 1980.
51. PSE. halo2curves, 2024. `https://github.com/privacy-scaling-explorations/halo2curves`.
52. PSE. halo2_proofs, 2024. `https://github.com/zcash/halo2/tree/main/halo2_proofs`.
53. Joost Renes, Craig Costello, and Lejla Batina. Complete addition formulas for prime order elliptic curves. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 403–428. Springer, 2016.
54. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE Symposium on Security and Privacy 2014*, pages 459–474. IEEE, 2014.
55. Jae Hong Seo. Round-efficient sub-linear zero-knowledge arguments for linear algebra. In *PKC 2011*, volume 6571 of *LNCS*, pages 387–402. Springer, 2011.
56. Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan 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.

57. Srinath Setty, Justin Thaler, and Riad Wahby. Customizable constraint systems for succinct arguments. *Cryptology ePrint Archive*, 2023.
58. Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *Annual International Cryptology Conference*, pages 299–328. Springer, 2022.
59. Tsz Hon Yuen, Muhammed F Esgin, Joseph K Liu, Man Ho Au, and Zhimin Ding. Dualring: generic construction of ring signatures with efficient instantiations. In *Annual International Cryptology Conference*, pages 251–281. Springer, 2021.
60. zcash. The halo2 book, 2022. `https://zcash.github.io/halo2/`.
61. Jianning Zhang, Ming Su, Xiaoguang Liu, and Gang 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.
62. Zongyang Zhang, Weihan Li, Ximeng Liu, Xin Chen, and Qihang Peng. Ligerolight: Optimized iop-based zero-knowledge argument for blockchain scalability. *IEEE Transactions on Dependable and Secure Computing*, 2023.

A Proof of Theorem 1

Proof. (Completeness) For a base case $m = 1$, the completeness is held by the completeness of $\mathsf{Protocol.Col}$ and AggMEC. Let us consider the case $m > 1$. In this case, we show that if the input $(\boldsymbol{G}, \boldsymbol{H}, \mathsf{ck}_{P,s}, \mathrm{P}, c; \boldsymbol{a}, \boldsymbol{b})$ belongs to $\mathcal{R}^{m,n}_{\mathsf{GenPT4}}$, then the updated input $(\widehat{\boldsymbol{G}}, \widehat{\boldsymbol{H}}, \mathsf{ck}_{P,s+1}, \widehat{\mathrm{P}}, \widehat{c}; \widehat{\boldsymbol{a}}, \widehat{\boldsymbol{b}})$ belongs to $\mathcal{R}^{m/2,n}_{\mathsf{GenPT4}}$. Following the $\mathcal{P}$ algorithm, we get the following equations:

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

Therefore, we can conclude that updated input $(\widehat{\boldsymbol{G}}, \widehat{\boldsymbol{H}}, \mathsf{ck}_{P,s+1}, \widehat{\mathrm{P}}, \widehat{c}; \widehat{\boldsymbol{a}}, \widehat{\boldsymbol{b}})$ belongs to $\mathcal{R}^{m/2,n}_{\mathsf{GenPT4}}$.
(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 [13], 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 $x_1, \ldots, x_\mu$ in sequence. Let $n_i \geq 1$ for $i \in [\mu]$. Consider an $(n_1, \ldots, n_\mu)$-tree of accepting transcripts with challenges in the following format. The tree has depth μ and $N = \prod_{i=1}^{\mu} n_i$ leaves. 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 extraction algorithm that succeeds with probability $1 - \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\lambda)$ in extracting a witness from an

$(n_1, \ldots, n_\mu)$-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 $(\boldsymbol{G}, \boldsymbol{H}, (\mathsf{ck}_k)_{k=1}^{\mu}, \mathsf{ck}_{\mathsf{Col}}, \mathrm{P}, c, st_V; \boldsymbol{a}, \boldsymbol{b}, st_P)$. By premise, $\mathcal{E}_{Row}$ exploits two PPT extractors $\mathcal{E}_{Col}$ and $\mathcal{E}_{MEC}$, that extract witness $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^{1\times n}$ and st_P respectively. Note that st_P consists of tuples of commitments $(\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k)$, which satisfies the Eq. (3) and (4).

We show how to extract witness $\boldsymbol{a}, \boldsymbol{b}$ 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 $(\underbrace{4, \ldots, 4}_{\log_2 m})$-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 $\boldsymbol{a}, \boldsymbol{b}$ from $\mathcal{E}_{Col}$ satisfies the desire condition so that $\mathcal{E}_{Row}$ outputs the $\boldsymbol{a}, \boldsymbol{b}$ in polynomial time.

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

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

$$\begin{bmatrix} x_1^{-1} & 1 & x_1 \\ x_2^{-1} & 1 & x_2 \\ x_3^{-1} & 1 & x_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{L} \\ \boldsymbol{P} \\ \boldsymbol{R} \end{bmatrix} = \begin{bmatrix} \widehat{\boldsymbol{P}}_1 \\ \widehat{\boldsymbol{P}}_2 \\ \widehat{\boldsymbol{P}}_3 \end{bmatrix} = \begin{bmatrix} [\widehat{\boldsymbol{a}}_1]\widehat{\boldsymbol{G}}_1 \parallel [\widehat{\boldsymbol{b}}_1]\widehat{\boldsymbol{H}}_1 \\ [\widehat{\boldsymbol{a}}_2]\widehat{\boldsymbol{G}}_2 \parallel [\widehat{\boldsymbol{b}}_2]\widehat{\boldsymbol{H}}_2 \\ [\widehat{\boldsymbol{a}}_3]\widehat{\boldsymbol{G}}_3 \parallel [\widehat{\boldsymbol{b}}_3]\widehat{\boldsymbol{H}}_3 \end{bmatrix} \tag{8}$$

Since the right-hand side of Eq. (8) is decomposed by $\widehat{\boldsymbol{G}} = \boldsymbol{G}_L + [x^{-1}]\boldsymbol{G}_R$ and $\widehat{\boldsymbol{H}} = \boldsymbol{H}_L + [x]\boldsymbol{H}_R$ and each $\boldsymbol{G}$ and $\boldsymbol{H}$ are not effected by challenge x, $\mathcal{E}_{Row}$ can get represented vectors $\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{p} \in \mathbb{Z}_p^{m/2^s \times 2n}$ of $\boldsymbol{L}, \boldsymbol{R}, \boldsymbol{P} \in \mathbb{G}^{2n}$ under base $\boldsymbol{G} \parallel \boldsymbol{H}$. Let $\mathcal{E}_{Row}$ parse $\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{p}$ to 4 segments $\boldsymbol{l}^{(t)}, \boldsymbol{p}^{(t)}, \boldsymbol{r}^{(t)} \in \mathbb{Z}_p^{m/2^s \times n/2}$ where $t \in [4]$. Let the representation vectors put on Eq. (8). Then

$$[x_i^{-1}\boldsymbol{l}^{(1)} + \boldsymbol{p}^{(1)} + x_i\boldsymbol{r}^{(1)}]\boldsymbol{G}_L = [\widehat{\boldsymbol{a}}]\boldsymbol{G}_L \tag{9}$$

$$[x_i^{-1}\boldsymbol{l}^{(2)} + \boldsymbol{p}^{(2)} + x_i\boldsymbol{r}^{(2)}]\boldsymbol{G}_R = [x_i^{-1}\widehat{\boldsymbol{a}}]\boldsymbol{G}_R \tag{10}$$

$$[x_i^{-1}\boldsymbol{l}^{(3)} + \boldsymbol{p}^{(3)} + x_i\boldsymbol{r}^{(3)}]\boldsymbol{H}_L = [\widehat{\boldsymbol{b}}]\boldsymbol{H}_L \tag{11}$$

$$[x_i^{-1}\boldsymbol{l}^{(4)} + \boldsymbol{p}^{(4)} + x_i\boldsymbol{r}^{(4)}]\boldsymbol{H}_R = [x_i\widehat{\boldsymbol{b}}]\boldsymbol{H}_R \tag{12}$$

By DL assumption on $\mathbb{G}$, the representation vectors of both side should be equivalent. From Eq.(9), Eq.(10) and Eq.(11), Eq.(12), we get

$$\begin{aligned} &-\boldsymbol{l}^{(1)} + x_i(\boldsymbol{l}^{(2)} - \boldsymbol{p}^{(1)}) + x_i^2(\boldsymbol{p}^{(2)} - \boldsymbol{r}^{(1)}) + x_i^3\boldsymbol{r}^{(2)} = 0 \\ &-\boldsymbol{l}^{(4)} + x_i(\boldsymbol{l}^{(3)} - \boldsymbol{p}^{(4)}) + x_i^2(\boldsymbol{p}^{(3)} - \boldsymbol{r}^{(4)}) + x_i^3\boldsymbol{r}^{(3)} = 0 \end{aligned}$$

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

$$\begin{aligned} x_i^{-1}c_L + c + x_i c_R = \widehat{c} &= \langle \widehat{\boldsymbol{a}}, \widehat{\boldsymbol{b}} \rangle \\ &= x_i^{-1}\langle \tilde{\boldsymbol{a}}_L, \tilde{\boldsymbol{b}}_R \rangle + \langle \tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}} \rangle + x_i \langle \tilde{\boldsymbol{a}}_R, \tilde{\boldsymbol{b}}_L \rangle \end{aligned} \tag{13}$$

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

B Proof of Theorem 2

Proof. (Completeness) For a base case $m = 1$, the completeness can get straightforward by our premise: completeness of AggMEC and $(G, H, \mathsf{ck}_1, \mathrm{P}, c; \boldsymbol{a}, \boldsymbol{b}) \in \mathcal{R}^{1,1}_{\mathsf{GenPT4}}$. Let consider the case $m > 1$. In this case, we show that if the input $(G, H, \mathsf{ck}_{P,\mu+s}, \mathrm{P}, c; \boldsymbol{a}, \boldsymbol{b})$ belongs to $\mathcal{R}^{1,n}_{\mathsf{GenPT4}}$, then the updated input $(G, H, \mathsf{ck}_{P,\mu+s+1}, \widehat{\mathrm{P}}, \widehat{c}; \widehat{\boldsymbol{a}}, \widehat{\boldsymbol{b}})$ belongs to $\mathcal{R}^{1,n/2}_{\mathsf{GenPT4}}$. Following the $\mathcal{P}$ algorithm, we get the following equations:

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

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

(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 $(\mathsf{pp}_\nu, G, H, \mathrm{P}, c, st_V; \boldsymbol{a}, \boldsymbol{b}, st_P)$. By premise, $\mathcal{E}_{Col}$ exploits a PPT extractor $\mathcal{E}_{MEC}$, that extract st_P which consists of commitments $(\boldsymbol{P}_k)_{k=\mu+1}^{\mu+\nu+1}$, which satisfies Eq. (5) and $\mathrm{P}_k = \mathsf{Com}_2(\mathsf{ck}_\nu, \boldsymbol{P}_k)$ In the similar way in proof of Theorem 1, we show that how to extract witness $\boldsymbol{a}, \boldsymbol{b}$ 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 $(\underbrace{3, \ldots, 3}_{\log_2 n})$-tree of accepting transcripts. 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 $\boldsymbol{a}^{(s)}, \boldsymbol{b}^{(s)} \in \mathbb{Z}_p^{1\times n/2^s}$ from next round witnesses $\boldsymbol{a}^{(s+1)}, \boldsymbol{b}^{(s+1)} \in \mathbb{Z}_p^{1\times n/2^{s+1}}$ recursively.

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

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{P}^{(q_1)} \\ \boldsymbol{P}^{(q_2)} \end{bmatrix} = \begin{bmatrix} [\widehat{\boldsymbol{a}}_1]G \\ [\widehat{\boldsymbol{a}}_2]G \end{bmatrix}, \begin{bmatrix} 1 & x_1^{-1} \\ 1 & x_2^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{P}^{(q_3)} \\ \boldsymbol{P}^{(q_4)} \end{bmatrix} = \begin{bmatrix} [\widehat{\boldsymbol{b}}_1]H \\ [\widehat{\boldsymbol{b}}_2]H \end{bmatrix} \tag{14}$$

By DL assumption, $\mathcal{E}_{Col}$ solves the linear equation and then get the representation $\boldsymbol{p}^{(q_1)}, \boldsymbol{p}^{(q_2)} \in \mathbb{Z}_p^{n/2^{s+1}}$ of $\boldsymbol{P}^{(q_1)}, \boldsymbol{P}^{(q_2)} \in \mathbb{G}^{n/2^{s+1}}$ under base G and $\boldsymbol{p}^{(q_3)}, \boldsymbol{p}^{(q_4)} \in \mathbb{Z}_p^{n/2^{s+1}}$ of $\boldsymbol{P}^{(q_3)}, \boldsymbol{P}^{(q_4)} \in \mathbb{G}^{n/2^{s+1}}$ under base H respectively. Then $\boldsymbol{p} = \boldsymbol{p}^{(q_1)} \parallel \boldsymbol{p}^{(q_2)} \parallel \boldsymbol{p}^{(q_3)} \parallel \boldsymbol{p}^{(q_4)}$ is naturally representation of $\boldsymbol{P}$. Let $\boldsymbol{p}^{(q_i)}$ denote $\tilde{\boldsymbol{a}}_L = \boldsymbol{p}^{(q_1)}$, $\tilde{\boldsymbol{a}}_R = \boldsymbol{p}^{(q_2)}$, $\tilde{\boldsymbol{b}}_L = \boldsymbol{p}^{(q_3)}$, $\tilde{\boldsymbol{b}}_R = \boldsymbol{p}^{(q_4)}$. In the similar way in Eq. (13) of Theorem 1, 3 distinct challenges guarantee extracted vectors $\langle \tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}} \rangle$ is equal to the value c. Therefore, the extracted witnesses $\tilde{\boldsymbol{a}}$ and $\tilde{\boldsymbol{b}}$ is valid witness for the relation $\mathcal{R}_{\mathsf{GenPT4}}^{1,n/2^s}$. By inductively retrieving process and general forking lemma, $\mathcal{E}_{Col}$ can extract witness vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. □

C Proof of Theorem 3

Proof. (Completeness) Assume that the input $\mathsf{ck}_k, (\mathrm{L}_k, \mathrm{R}_k, \mathrm{P}_k, x_k); (\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k)$ satisfies Eq. (3), (4), and (5). By the homomorphic property of polynomial commitment scheme and perfect completeness of Eval and $\mathsf{Plonkish}_{\mathsf{Eval}}$, $\mathcal{V}$ accepts both Eval and $\mathsf{Plonkish}_{\mathsf{Eval}}$. 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 $\mathsf{ck}_k, (\mathrm{L}_k, \mathrm{R}_k, \mathrm{P}_k, x_k)$ and returns witness vectors $(\boldsymbol{L}_k, \boldsymbol{R}_k, \boldsymbol{P}_k)$ 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 [18].

By our premise, one can construct a PPT extractor $\mathcal{E}_{\mathsf{Eval}}$ for $\mathsf{PCS.Eval}$. In addition, since the above premise implies that the $\mathsf{Plonkish}_{\mathsf{Eval}}$ is an AoK, one can construct a PPT extractor $\mathcal{E}_{\mathsf{Plonkish}}$ for $\mathsf{Plonkish}_{\mathsf{Eval}}$ 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 $\mathcal{E}_{\mathsf{Eval}}$ with $(\mathsf{ck}_{\mathsf{PC}}, \mathrm{P}, z, y)$ and $(\mathsf{ck}_{\mathsf{PC}}, W^{(i)}, z, r^{(i)})$, 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 $6 \log m + 2 \log n + 1$:

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

The extracted polynomials satisfy the following relation:

$$\mathrm{L}_k^{(i)} = \mathsf{Com}_{\mathsf{PC}}(\mathsf{ck}_{\mathsf{PC}}, F_{L,k}),\ \mathrm{R}_k^{(i)} = \mathsf{Com}_{\mathsf{PC}}(\mathsf{ck}_{\mathsf{PC}}, F_{R,k}),\ \mathrm{P}_k^{(i)} = \mathsf{Com}_{\mathsf{PC}}(\mathsf{ck}_{\mathsf{PC}}, F_{P,k}) \quad (15)$$

Finally, $\mathcal{E}_{MEC}$ outputs $\boldsymbol{L}_k^{(i)}, \boldsymbol{R}_k^{(i)}, \boldsymbol{P}_k^{(i)}$ 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 $\boldsymbol{L}_k^{(i)}, \boldsymbol{R}_k^{(i)}, \boldsymbol{P}_k^{(i)}$ satisfy Eq. (15), so does Eq. (3). In addition, by the construction of $\mathrm{A}^{(i)}$, 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 $\boldsymbol{L}_k^{(i)}, \boldsymbol{R}_k^{(i)}, \boldsymbol{P}_k^{(i)}$ at the corresponding positions for $i \in \{1, 2\}$. Furthermore, the wire polynomials $\{w^{(i)}(X)\}_{i=0}^{M-1}$ extracted from $\mathcal{E}_{\mathsf{Plonkish}}$ ensure that $\boldsymbol{L}_k^{(i)}, \boldsymbol{R}_k^{(i)}, \boldsymbol{P}_k^{(i)}$ satisfy the relation Eq. (4) and Eq. (5).

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

D Plonkish Arithmetization with Custom Gates

In this section, we provide supplementary description of Plonkish for elliptic curve operations.

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 $c_i(X) = g_i(w^{(0)}(X), w^{(1)}(X), \ldots, w^{(M-1)}(X))$ 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 $\sigma : [D] \times [M] \to [D] \times [M]$. $\sigma(i,j) = (\sigma(i,j)_1, \sigma(i,j)_2)$ 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 [60]. Define $\mathsf{ID}_i(\xi^j) = \delta^i \cdot \xi^j$ 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 $\prod_{h=0}^{D-1} \prod_{i=0}^{M-1} \frac{w^{(i)}(\xi^h) + u_1 \mathsf{ID}_i(\xi^h) + u_2}{w^{(i)}(\xi^h) + u_1 r_i(\xi^h) + u_2}$ 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 [7].

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) $\sum_{i=0}^{N_g-1} s_i(X) c_i(X) = 0 \mod X^D - 1$. As shown in several studies [31, 32, 60], the relations can be efficiently proved by the Polynomial IOP instantiated by PCS [21]. 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 [32] 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 $\prod_{i=0}^{M-1} \frac{w^{(i)}(\xi^h) + u_1 \mathsf{ID}_i(\xi^h) + u_2}{w^{(i)}(\xi^h) + u_1 r_i(\xi^h) + u_2}$ one by one for $h \in [D]$. Then, as shown by [7], $z(X)$ satisfies $z(\xi X)/z(X) = \prod_{i=0}^{M-1}(w^{(i)}(X) + u_1 \mathsf{ID}_i(X) + u_2) / \prod_{i=0}^{M-1}(w^{(i)}(X) + u_1 r_i(X) + u_2) \mod X^D - 1$ and $z(\xi^0) = 1$ for random challenges u_1, u_2. Hence, by combining these con-

straints 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 5.

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 $\mathsf{ck}_{\mathsf{PC}}$ would lead to inefficiency in the computational cost for both the prover and verifier. To avoid this, as did in `halo2` library [60], 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 $t_1(X), \ldots, t_{d_t}(X)$ and $Q_1(X), \ldots q_{d_t-1}(X)$ 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 $q_j(X)$ 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 $\widehat{t}_1 := t_1(u_4), \ldots, \widehat{t}_{d_t} := t_{d_t}(u_4)$ and $\widehat{q}_1 := q_1(u_4), \ldots, \widehat{q}_{d_t-1} := q_{d_t-1}(u_4)$ to the verifier. Then the verifier checks

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

and run $\mathsf{Eval}(\mathsf{ck}_{\mathsf{pc}}, T_i, u_4, \widehat{t}_i; t_i(X))$ and $\mathsf{Eval}(\mathsf{ck}_{\mathsf{pc}}, Q_j, u_4, \widehat{q}_j; q_j(X))$ 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 $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$. 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 4.2, the computational cost for the verifier would be significantly reduced compared to the case when we use the commitment key of length $d_t \cdot D$.

$\mathsf{Plonkish}_{\mathsf{Eval}}$. Using the constraints system described in Section 4.1, 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 $\mathsf{Plonkish}_{\mathsf{Eval}}$ as an instantiation of Plonkish with the custom gate polynomial g_{MA}.

Algorithm 5 $\mathsf{Plonkish}_{\mathsf{Eval}}$

$\mathsf{Plonkish}_{\mathsf{Eval}}(\mathsf{ck}_{\mathsf{PC}}, \{s_i(X), g_i(X_0, \ldots, X_{M-1})\}_{i=0}^{N_g-1}, \{r_i(X)\}_{i=0}^{M-1}; \{w^{(i)}(X)\}_{i=0}^{M-1})$

Precompute: $C_{\mathsf{ID}_i} = \mathsf{Com}_{\mathsf{PC}}(\mathsf{ck}_{\mathsf{PC}}, \mathsf{ID}_i(X))$, $C_{r_i} = \mathsf{Com}_{\mathsf{PC}}(\mathsf{ck}_{\mathsf{PC}}, r_i(X))$, $i \in [M]$

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

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

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

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

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

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

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

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

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

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

7: $\mathcal{P}$ sends $\{\alpha_i = w^{(i)}(u_4)\}_{i=0}^{M-1}$, $\beta = z(u_4)$, $\gamma = z(\xi u_4)$,
$\{\phi_i = \mathsf{ID}_i(u_4)\}_{i=0}^{M-1}$, 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 $\mathcal{P}$

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

9: $\mathcal{P}$ and $\mathcal{V}$ set run $\mathsf{Eval}(\mathsf{ck}_{\mathsf{PC}}, T, u_4, \rho_1; t(X))$, $\mathsf{Eval}(\mathsf{ck}_{\mathsf{PC}}, W^{(i)}, u_4, \alpha_i; w^{(i)}(X))_{i \in [M]}$,
$\mathsf{Eval}(\mathsf{ck}_{\mathsf{PC}}, Q, u_4, \rho_2; q(X))$, $\mathsf{Eval}(\mathsf{ck}_{\mathsf{PC}}, Z, u_4, \beta; z(X))$, $\mathsf{Eval}(\mathsf{ck}_{\mathsf{PC}}, Z, \xi u_4, \gamma; z(X))$,
$\mathsf{Eval}(\mathsf{ck}_{\mathsf{PC}}, C_{\mathsf{ID}_i}, u_4, \phi_i; \mathsf{ID}_i(X))_{i \in [M]}$, and $\mathsf{Eval}(\mathsf{ck}_{\mathsf{PC}}, C_{r_i}, u_4, \psi_i; r_i(X))_{i \in [M]}$.

E Polynomial Commitment Scheme from Leopard

In this section, we provide details about the $\mathsf{Leopard}_{\mathsf{PC}}$, which is a key ingredient to instantiate Cougar. Remark that [43, Section E.1.] provided a basic idea for constructing this; we present the full description for the sake of completeness.

$\mathsf{Leopard}_{\mathsf{PC}}$ is a natural tweak of Protocol3 [43] as a PCS. The construction idea is basically the same as the PCS introduced by [17], 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 $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$.

Following the above approach, we provide the full description of $\mathsf{Leopard}_{\mathsf{PC}}$ 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 $a(X) \in \mathbb{Z}_p^{<mn}[X]$ and positive integers $m, n \in \mathbb{N}$, we will denote its coefficient vector as $\boldsymbol{a} \in \mathbb{Z}_p^{mn}$, namely, $\boldsymbol{a} = (a_0, \ldots, a_{mn-1})$ such that $a(X) = \sum_{i=0}^{mn-1} a_i X^i$. $\mathsf{Leopard}_{\mathsf{PC}} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Eval})$ over a message space $\mathbb{Z}_p^{<mn}[X]$ and a commitment space $\mathbb{G}_t$ is defined as follows[3]:

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

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

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

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

We now show that $\mathsf{Leopard}_{\mathsf{PC}}$ is indeed the PCS, *i.e.*, satisfying the conditions in Definition 7, under the DL assumption. In fact, $\boldsymbol{G} \otimes \boldsymbol{H}$ 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 $\mathsf{Leopard}_{\mathsf{PC}}$ under a structure-aware version of the DLR assumption.

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

Algorithm 6 $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$

$\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}(\mathsf{ck}_{\mathsf{PC}} = (\boldsymbol{G}, \boldsymbol{H}) \in \mathbb{G}_1^m \times \mathbb{G}_2^n, P \in \mathbb{G}_t, z, y \in \mathbb{Z}_p; \boldsymbol{a} \in \mathbb{Z}_p^{mn})$
where $m = 2^\mu$ and $n = 2^\nu$

$\mathcal{V}$ picks $U \xleftarrow{\$} \mathbb{G}_t$ and sends it to $\mathcal{P}$
$\mathcal{P}$ and $\mathcal{V}$ set $P_0 = P + [y]U$, $\boldsymbol{G}_0 = \boldsymbol{G}, \boldsymbol{H}_0 = \boldsymbol{H}$.
Additionally, $\mathcal{P}$ set $\boldsymbol{a}_0 = \boldsymbol{a}$ and $\boldsymbol{z}_0 = [z^{m(i-1)+(j-1)}] \in \mathbb{Z}_p^{m \times n}$
for $i = 0, \ldots, \mu - 1$ **do**
  $\mathcal{P}$ parses $\boldsymbol{a}_i$, $\boldsymbol{z}_i$, and $\boldsymbol{G}_i$ to
  $\boldsymbol{a}_i = [\boldsymbol{a}_{i,L} \parallel \boldsymbol{a}_{i,R}], \quad \boldsymbol{z}_i = [\boldsymbol{z}_{i,L} \parallel \boldsymbol{z}_{i,R}], \quad \boldsymbol{G}_i = \boldsymbol{G}_{i,L} \| \boldsymbol{G}_{i,R}$
  $\mathcal{P}$ computes:
  $L_i = [\boldsymbol{a}_{i,L}](\boldsymbol{G}_{i,R} \otimes \boldsymbol{H}) + [\langle \boldsymbol{a}_{i,L}, \boldsymbol{z}_{i,R} \rangle]U \in \mathbb{G}_t$
  $R_i = [\boldsymbol{a}_{i,R}](\boldsymbol{G}_{i,L} \otimes \boldsymbol{H}) + [\langle \boldsymbol{a}_{i,R}, \boldsymbol{z}_{i,L} \rangle]U \in \mathbb{G}_t$
  $\mathcal{P}$ sends L_i, R_i to $\mathcal{V}$
  $\mathcal{V}$ chooses $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sends it to $\mathcal{P}$
  $\mathcal{P}$ computes:
  $\boldsymbol{a}_{i+1} = \boldsymbol{a}_{i,L} + r_i^{-1}\boldsymbol{a}_{i,R}, \quad \boldsymbol{z}_{i+1} = \boldsymbol{z}_{i,L} + r_i \boldsymbol{z}_{i,R} \in \mathbb{Z}_p^{m/2^{i+1} \times n}$
  $\boldsymbol{G}_{i+1} = \boldsymbol{G}_{i,L} + [r_i]\boldsymbol{G}_{i,R} \in \mathbb{G}_1^{m/2^{i+1}}$
  $P_{i+1} = [r_i]L_i + P_i + [r_i^{-1}]R_i \in \mathbb{G}_t$
end for
for $j = 0, \ldots, \nu - 1$ **do**
  $\mathcal{P}$ sets $i = j + \mu$ and then parses $\boldsymbol{a}_i$, $\boldsymbol{z}_i$, and $\boldsymbol{H}_j$ to
  $\boldsymbol{a}_i = \boldsymbol{a}_{i,L} \parallel \boldsymbol{a}_{i,R}, \quad \boldsymbol{z}_i = \boldsymbol{z}_{i,L} \parallel \boldsymbol{z}_{i,R}, \quad \boldsymbol{H}_j = \boldsymbol{H}_{j,L} \| \boldsymbol{H}_{j,R}$
  $\mathcal{P}$ computes:
  $L_i = [\boldsymbol{a}_{i,L}](G_\mu \otimes \boldsymbol{H}_{j,R}) + [\langle \boldsymbol{a}_{i,L}, \boldsymbol{z}_{i,R} \rangle]U \in \mathbb{G}_t$
  $R_i = [\boldsymbol{a}_{i,R}](G_\mu \otimes \boldsymbol{H}_{j,L}) + [\langle \boldsymbol{a}_{i,R}, \boldsymbol{z}_{i,L} \rangle]U \in \mathbb{G}_t$
  $\mathcal{V}$ chooses $r_i \xleftarrow{\$} \mathbb{Z}_p^*$ and sends it to $\mathcal{P}$
  $\mathcal{P}$ computes:
  $\boldsymbol{a}_{i+1} = \boldsymbol{a}_{i,L} + s_j^{-1}\boldsymbol{a}_{i,R}, \quad \boldsymbol{z}_{i+1} = \boldsymbol{z}_{i,L} + s_j \boldsymbol{z}_{i,R} \in \mathbb{Z}_p^{n/2^{j-1}}$
  $\boldsymbol{H}_{j+1} = \boldsymbol{H}_{j,L} + [s_j]\boldsymbol{H}_{j,R} \in \mathbb{G}_2^{n/2^{j+1}}$
  $P_{i+1} = [s_i]L_i + \cdot P_i + [s_i^{-1}]R_i \in \mathbb{G}_t$
end for
$\mathcal{P}$ sends $a = a_{\mu+\nu} \in \mathbb{Z}_p$ to $\mathcal{V}$
$\mathcal{V}$ computes:
$\boldsymbol{r}[k+1] = \langle \mathsf{bit}(k), (r_0, \ldots, r_{\mu+\nu-1}) \rangle$ for $k = 0, \ldots m+n-1$
Parse $\boldsymbol{r}$ to $\boldsymbol{r}_{row} \parallel \boldsymbol{r}_{col}$ where $\boldsymbol{r}_{row} \in \mathbb{Z}_p^m$ and $\boldsymbol{r}_{col} \in \mathbb{Z}_p^n$
$G = \langle \boldsymbol{r}_{row}, \boldsymbol{G}_0 \rangle, H = \langle \boldsymbol{r}_{col}, \boldsymbol{H}_0 \rangle, z = \boldsymbol{r}_{row} \boldsymbol{z}_0 \boldsymbol{r}_{col}$
$\mathcal{V}$ checks:
$P_0 + \sum_{i \in [\mu+\nu]} ([r_i]L_i + [r_i^{-1}]R_i) = e([az]G, H)$

For this reason, we first provide a definition of generalized discrete logarithm relation (GDLR) assumption, which was previously defined in [43, 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.

Definition 8. *For $m, n \in \mathbb{N}$ and the security parameter $\lambda \in \mathbb{N}$, let* GDLRsp *be a sampler defined by*

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

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:*

$$\Pr\left[\boldsymbol{a} \neq \boldsymbol{0} \ \wedge \ \boldsymbol{g}^{\boldsymbol{a}} = 1_{\mathbb{G}_t} \middle| \begin{matrix}(p, \boldsymbol{g} \in \mathbb{G}_t^{m \times n}, \mathbb{G}_t) \leftarrow \mathsf{GDLRsp}(1^\lambda) \\ \boldsymbol{a} \leftarrow \mathcal{A}(p, \boldsymbol{g}, \mathbb{G}_t)\end{matrix}\right]$$

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

As shown by [43, 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 $\mathsf{Leopard}_{\mathsf{PC}}$ holds immediately.

Now it remains to check that $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$ is an AoK for the relation $\mathcal{R}_{\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}}$. As we mentioned, this relation can be understood as a special case of that for $\mathsf{Protocol3}$, and Algorithm 6 is in fact almost identical to $\mathsf{Protocol3}$. We note that $\mathsf{Protocol3}$ satisfies perfect completeness and computational witness-extended emulation under the GDLR assumption [42]. In fact, we made the same modifications as [17] for constructing $\mathsf{Leopard}_{\mathsf{PC}}.\mathsf{Eval}$, 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 [42] and [17] for more detailed information.

To sum up, $\mathsf{Leopard}_{\mathsf{PC}}$ satisfies all conditions in Definition 7 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.