Dinocchio: Distributed Prover for Ring Arithmetic

Katerina Sotiraki
Yale University

Yunhao Wang
Yale University

Fan Zhang
Yale University

Abstract

Nearly all existing SNARK systems are optimized for arithmetic over finite fields. Using them to prove statements involving ring arithmetic, which underlies lattice-based cryptography and fully homomorphic encryption (FHE), incurs substantial overhead. Consequently, practical deployments of FHE often rely on the *honest-but-curious* assumptions, leaving a gap in the verifiability of the FHE computation. Several recent works have explored zero-knowledge proofs tailored to lattice-based schemes, yet still suffer from high prover costs and limited scalability.

In this work, we introduce Dinocchio, the first distributed SNARK for rings with constant proof size and constant verification time. For a setting with m sub-provers, Dinocchio achieves an approximately m-fold speedup in prover time compared to Rinocchio (JoC'23), while preserving the constant verification time independent of m.

We demonstrate the practicality of Dinocchio through matrix multiplication, which is a crucial building block to large-scale lattice-based applications. With matrices of size $2^{12} \times 2^{12}$, the corresponding arithmetic circuit contains $\sim 2^{32}$ constraints, which is beyond the reach of all existing works. Our microbenchmarks show that Dinocchio can generate a succinct proof in around 9.23 hours with 128 sub-provers, more than 108× faster than the prior work, and the verifier completes the verification in under 16 seconds.

1 Introduction

Zero-knowledge proofs (ZKPs) have emerged as an indispensable tool in numerous applications, such as private blockchain transactions and scalable rollups, verifiable machine learning, and secure cloud computation. In particular, succinct non-interactive zero-knowledge proofs (SNARKs) offer great efficiency both in terms of proof size and verification time. However, proof generation remains their dominant cost. This bottleneck poses severe challenges in the real-world deployment of SNARKs, where proofs must be generated at high throughput and low latency.

A promising direction to address this challenge is through *distributed proof generation* [63]. In this setting, the prover's workload is distributed among multiple parties, who jointly produce a final proof with as little communication as possible. Recent works [64, 42, 39, 63, 56, 61, 65] have demonstrated substantial speedup for distributed SNARKs, making them a compelling approach for improving the proof generation time. However, all works on distributed SNARKs focus on proving computation performed over finite fields. This leaves a crucial gap in constructing distributed SNARKs for ring arithmetic, which have become increasingly relevant in emerging cryptographic applications such as lattice-based cryptography and fully homomorphic encryption.

Fully Homomorphic Encryption (FHE) is a powerful tool for privacy-preserving tasks and secure cloud computing by allowing a server to perform computation under encryption, including applications such as privacy-preserving ML [36, 33, 37, 34, 21], private information retrieval [4, 51,

26, 71, 31] and private set intersection [17, 16, 20], oblivious message retrieval [44, 45, 46, 38, 43], and single secret leader election [8, 62]. Substantial progress has been made in the performace of FHE itself [13, 23, 19, 18, 12, 10, 24]. However, verifying that an encrypted computation was performed correctly is a major challenge. Recently, [60] introduced the notion of verifiable FHE (vFHE), where the server has to prove that the encrypted computation was performed correctly. The authors of [60] also observed that certifying the correctness of a computation using a ZKP over rings leads to significantly faster protocols than using ZKPs over finite fields.

In this work, we construct a distributed SNARK over rings, denoted as Dinocchio, where the prover computation is distributed among multiple subprovers. Specifically, we build on [27] to construct an SNARK over rings with constant proof size and verification time, where for proving a statement of size N with m subprovers, each sub-prover performs $O(\frac{N}{m} \log \frac{N}{m})$ operations. In addition, the total communication between the subprovers is only $O(m)$.

1.1 Our Contributions

- We introduce Dinocchio, the first construction that enables *distributed* proof generation natively over rings extending the Rinocchio framework [27]. Under a distributed setup with m sub-provers, our prover runtime is approximately m times faster than Rinocchio, while maintaining constant verification time, independent of the number of sub-provers. For a circuit of size N, each sub-prover performs only $O(\frac{N}{m} \log \frac{N}{m})$ computation.

- Our design introduces a novel aggregation mechanism that enables sub-provers to independently locally generate sub-proofs, which are subsequently aggregated into a single, succinct proof. This aggregation protocol has runtime $O(m \log m)$, independent of N. Different from all prior works on the distributed prover setting, Dinocchio operates over rings, requiring a novel adaptation of the partitioning and aggregation procedures to structured Quadratic Ring Programs (QRP). We emphasize that our adaptation preserves both succinctness and soundness while operating entirely within the arithmetic structure of the ring.

- We demonstrate the practicality of Dinocchio by applying it to matrix multiplication, a pivot building block in privacy-preserving computation tasks, such as secure machine learning. For matrices of size $2^{12} \times 2^{12}$, the corresponding arithmetic circuits contain more than 2^{32} constraints. To the best of our knowledge, no existing SNARK system has been demonstrated to handle circuits at this scale. Our microbenchmarks show that Dinocchio can efficiently generate a succinct proof in 9.23 hours using 128 sub-provers. Compared to Rinocchio [27], this yields a $\sim 108\times$ speedup, which is slightly below the number of sub-provers due to the overhead introduced by the aggregation. The verifier completes the verification in under 16 seconds. Each sub-prover produces a sub-proof of size 1.8 MB, and the final aggregated proof sent to the verifier is 11.4 MB.

1.2 Related Works

ZKPs over rings. Zero-knowledge proofs over ring arithmetic have been extensively studied in the context of lattice-based assumptions and FHE-friendly circuits. LaBRADOR [7] defines a general R1CS-style relation over Module-SIS and constructs a Bulletproof-inspired proof system with sublinear proof size and linear prover and verifier runtime. Greyhound [53] extends LaBRADOR to sublinear verifier runtime via polynomial commitments. Similarly, a recent work DV-zkSNARK

[1] adapts the framework of LaBRADOR based on the standard assumption under the random oracle model. Those works require a short witness, which might incur overhead when proving honest evaluation for a general FHE circuit. Instead, ZHE [72] directly targets HE evaluation proofs, but their final runtime improvement is now unclear due to (now fixed) security problems in their original scheme. There are also works tailored to specific FHE schemes, like Approx-VC [14] over CKKS, HasteBoots [41] and [58] for TFHE bootstrapping, and Heliopolis [5] and Laminate [55] over BG/FV, all achieving sublinear verification. It is also unclear if these optimizations tailored to a specific FHE scheme can be adapted to other FHE schemes. Other works [69, 6, 2], though efficient for general FHE operations, still do not achieve a constant proof size or have a sublinear verifier. In fact, none of the above achieves a constant proof size. Rinocchio [27] has constant proof size but suffers from heavy prover-side computation.[1]

Distributed provers. Distributed SNARKs aim to split prover computation across multiple parties or threads to reduce latency. Recent distributed SNARK systems, including deVergo [64], Pianist [42], HyperPianist [39], DIZK [63], Hekaton [56], Cirrus [61], and FRIttata [65], scale prover computation by partitioning the circuit and aggregating sub-proofs, achieving prover runtime improvement proportional to the number of sub-provers. While these field-based distributed SNARKs achieve substantial speedups, they rely on building blocks that are inherently field-specific, such as KZG commitments and bilinear pairings, thus making their adaptation to ring arithmetic non-trivial.

Verifiable FHE computation. A number of works address the integrity of outsourced homomorphic or encrypted computations via methods other than ZKP. DataSeal [57] is the SOTA work that applies an algorithm-based-fault-tolerance-style (ABFT) scheme. I.e., it embeds redundant computations to detect errors, but only supports plaintext-ciphertext matrix multiplications, and requires communication between the verifier and prover at each level of the circuit. MAC-based approaches embed error-detection encodings into FHE ciphertexts, yet they either rely on highly expressive homomorphic MAC [40] or suffer from large computation overhead [15]. TEE-hybrid protocols such as CHE-MIX [52] delegate part of the HE computation into a secure enclave, ensuring integrity under the assumption of trusted hardware. Fherret [32] leverages MPC to produce proofs of honest FHE evaluation with circuit privacy, but requires a linear-time verifier. A more recent work, MatVecMul [59], targets verifiable matrix-to-vector homomorphic multiplication by bridging the gap between the sumcheck protocol over fields and the polynomial rings used in FHE.

2 Preliminary

2.1 Notation

We use $\boxed{x}$ and $\boxed{\dot{x}}$ to denote the encoding of plaintext x, with solid boxes $\boxed{x}$ for the encodings provided in the crs and dotted boxes $\boxed{\dot{x}}$ for the encodings derived by the prover. We use two encoding schemes Π_1, Π_2; we write encodings by Π_i as $\boxed{x}_i$ and $\boxed{\dot{x}}_i$.

We denote by "$\diamond$" the multiplication between a plaintext (i.e., the witness value in the protocol) and an encoding, i.e., $a \diamond \boxed{x} := \boxed{\dot{ax}}$. We use $\diamond'$ to denote the *double* multiplication between two plaintexts and one encoding, i.e., for $(a_1, a_2) \diamond' \boxed{x} := \big(a_1 \diamond \boxed{x}, a_2 \diamond \boxed{x}\big)$. We refer to the result of

[1] As pointed out in [22], the linear-only assumption underlying the encoding scheme used in the proof has been shown *not* to be post-quantum secure. Accordingly, we do *not* claim post-quantum security for our construction. Rather, our focus is on the efficiency of proof generation for ring-based arithmetic.

this operation as an *extended* encoding and denote it by $\boxed{\boxed{z}}_1 := \boxed{x}_1 \otimes \boxed{y}_1$. We define the operation $\circ$ between two encodings $\boxed{x}_1, \boxed{y}_1$ as $\boxed{x}_1 \circ \boxed{y}_1 := ((a_1a_2, a_1b_2), (b_1a_2, b_1b_2)) \in (\mathcal{R}^2)^2$.

In our protocol, the encodings of Π_1 can be treated as plaintexts for Π_2. We use $\sharp\boxed{\cdot}\sharp_{1\to 2}$ or $\sharp\boxed{\cdot}\sharp_{1\to 2}$ to represent the casting from an encoding of Π_1 to a plaintext of Π_2, e.g., if $\boxed{x}_1$ is an encoding by Π_1 and $\boxed{y}_2$ is an encoding by Π_2, then $\sharp\boxed{x}\sharp_{1\to 2} \diamond \boxed{y}_2 = \boxed{\sharp\boxed{x}\sharp_{1\to 2} \cdot y}_2$. We abuse notation and use $\natural \cdot \natural_{1\to 2}$ to denote the casting for computations on an encoding, e.g., $\left\natural\ a \diamond \boxed{x}_1 + b \diamond \boxed{y}_1 \right\natural_{1\to 2} = \boxed{\sharp ax + by \sharp}_{1\to 2}$. To simplify the notation, we omit the subscript of a specific encoding scheme when clear from context. Since we *only* cast the encoding under Π_1 to a plaintext under Π_2, we also drop the subscript "$1 \to 2$" in later sections. For a ring element $a \in \mathcal{R}_t$, we let $[a]_q$ denote the canonical lifting of a from modulus t to modulus q.

Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. We use calligraphic capital letters (e.g., $\mathcal{I}, \mathcal{J}$) to denote sets of wire indices. We denote polynomials with capital letters, e.g., $V(x)$, and their evaluations on specific points with lowercase letters, e.g., $v := V(s)$. Additionally, we use primed lowercase letters, e.g., v', to represent the evaluations scaled by auxiliary monomials (e.g., $v' := V'(s) := s^j V(s)$). We use $\cdot$ to denote the standard ring multiplication.

2.2 Rings

Definition 2.1 (Decisional ring learning with error problem [50])**.** Let $n, q, \mathcal{D}, \chi$ be parameters dependent on λ and n being a power of two. Let $\mathcal{R} = \mathbb{Z}[X]/(X^D + 1)$, where D is the ring dimension. The ring learning with error (RLWE) problem $\mathsf{RLWE}_{n,q,\mathcal{D},\chi}$ is the following: distinguish $(a, a \cdot s + e)$ and (a, b) (with noticeable advantage), where $a \xleftarrow{\$} \mathcal{R}_q, s \leftarrow \mathcal{D}, e \leftarrow \chi$ and $b \xleftarrow{\$} \mathcal{R}_q$.

Exceptional set. For a ring $\mathcal{R} := \mathbb{Z}[X]/(X^N + 1)$, let $\mathcal{R}^* \subseteq \mathcal{R}$ denote the unit elements, and let $\mathbb{A}$ denote the exceptional set, i.e., for all $a_i, a_j \in \mathbb{A}$ with $a_i \neq a_j$, it holds that $a_i - a_j \in \mathcal{R}^*$. Let $\mathbb{A}^* \subseteq \mathbb{A}$ denote the subset of $\mathbb{A}$ containing elements that are units.

Quadratic ring program. We represent a circuit that evaluates ring arithmetic as a *quadratic ring program* as proposed in [27], which is adapted from its non-ring arithmetic version in [54].

Definition 2.2 (Quadratic Ring Program)**.** Let C be an arithmetic circuit over a finite commutative ring $\mathcal{R}$ with n inputs and n' outputs, and we associate each input of C and each output of a multiplication gate with an index $k \in \{0, \ldots, \mathsf{w}\}$. Without loss of generality, we assume the input wires have indices $(1, \ldots, n)$ and output wires $(\mathsf{w} - n' + 1, \ldots, \mathsf{w})$.

A Quadratic Ring Program (QRP) $\mathcal{Q}$ encoding C consists of three sets of polynomials, $\{V_k(x)\}$, $\{W_k(x)\}$, and $\{Y_k(x)\}$ for $k \in \{0, \ldots, \mathsf{w}\}$, and a target polynomial $t(x)$. We say that $\mathcal{Q}$ computes C if: $(c_1, \ldots, c_n, c_{\mathsf{w}-n'+1}, \ldots, c_{\mathsf{w}}) \in \mathcal{R}^{n+n'}$ is a valid input/output assignment of the circuit C if and only if there exists an assignment $(c_{n+1}, \ldots, c_{\mathsf{w}-n'}) \in \mathcal{R}^{\mathsf{w}-n-n'}$ such that $t(x)$ divides $p(x) = V(x) \cdot W(x) - Y(x)$, where:

$$V(x) = V_0(x) + \sum_{k \in [\mathsf{w}]} c_k V_k(x),$$

$$W(x) = W_0(x) + \sum_{k \in [\mathsf{w}]} c_k W_k(x),$$

$$Y(x) = Y_0(x) + \sum_{k \in [\mathsf{w}]} c_k Y_k(x).$$

As in prior work [27, 54], a QPR for a circuit C can be constructed as follows. We pick a root $r_g \in \mathcal{R}$ for each multiplication gate g and define the target polynomial as $t(x) = \prod_g (x - r_g)$. For each wire index k, a polynomial $V_k(x)$ is defined so that $V_k(r_g) = 1$ if the k-th wire is the left input of the multiplication gate g; and $V_k(r_g) = 0$ otherwise. Similarly, $W_k(x)$ encodes the right inputs, and $Y_k(x)$ encodes the outputs. We denote by $H(x)$ the quotient polynomial $H(x) := \frac{V(x)W(x)-Y(x)}{t(x)}$.

We also use the following notation. Let $\mathcal{I} := \{0, \cdots, \mathsf{w}\}$, let $\mathcal{I}_{\mathsf{io}} \subseteq \mathcal{I}$ denote the indices for public input and output values, and let $\mathcal{I}_{\mathsf{mid}} := \mathcal{I} \backslash \mathcal{I}_{\mathsf{io}}$ denote the set of all other wire indices in $\mathcal{I}$. Setting $c_0 = 1$, we can write $V(x) = V_{\mathsf{io}}(x) + V_{\mathsf{mid}}(x)$, where $V_{\mathsf{io}}(x) := \sum_{\kappa \in \mathcal{I}_{\mathsf{io}}} c_\kappa V_{\mathsf{io},\kappa}(x)$ and $V_{\mathsf{mid}}(x) = \sum_{k \in \mathcal{I}_{\mathsf{mid}}} c_k V_{\mathsf{mid},k}(x)$. Similarly, we define W_{io}, W_{mid}, Y_{io}, and Y_{mid}.

(Fully) homomorphic encryption. A (fully) homomorphic encryption scheme, first constructed by [29], consists of algorithms $(\mathsf{GenParam}, \mathsf{KeyGen}, \mathsf{E}, \mathsf{D}, \mathsf{Eval})$. The $\mathsf{GenParam}$ interface takes as input the security parameter and outputs the (F)HE parameters pp, which include the ring dimension D, the ciphertext modulus q, the plaintext modulus t, the secret key distribution $\mathcal{D}$ and the error distribution χ. The parameters pp are input to all other algorithms. The KeyGen algorithm generates the public key, the secret key, and the evaluation key $(\mathsf{pk}, \mathsf{sk}, \mathsf{ek}_{\mathsf{eval}})$. The E interface takes a public key pk and a plaintext message $m \in \mathcal{R}_t$ as inputs, and outputs a ciphertext $\mathsf{ct} := \boxed{m} \in \mathcal{R}_q$. Decryption recovers the plaintext accordingly: $m \leftarrow \mathsf{D}(\mathsf{ct}, \mathsf{sk})$. Given a function f and ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n) := \left(\boxed{m_1}, \ldots, \boxed{m_n}\right)$, $\mathsf{Eval}(f, (\mathsf{ct}_1, \ldots, \mathsf{ct}_n), \mathsf{ek}_{\mathsf{eval}})$ outputs ct', s.t., $\mathsf{D}(\mathsf{ct}') = f(m_1, \ldots, m_n)$.

Looking ahead, to demonstrate the practicality of our protocol, we estimate the runtime for large-scale ciphertext-ciphertext matrix multiplications under the BFV scheme [11, 25]. Below, we briefly recall several subtleties of BFV that are relevant to our evaluation.

In BFV, an input message is treated as a vector in $\mathbb{Z}_t^D$, which is then encoded into a ring element $\in \mathcal{R}_t$ via the Inverse Number Theoretic Transform (INTT) before encryption. Similarly, decryption produces a ring element that is decoded back into a vector in $\mathbb{Z}_t^D$. For simplicity and readability, we treat the encoding and decoding procedures as being wrapped inside the E and D interfaces. Moreover, BFV supports Single Instruction Multiple Data (SIMD), enabling homomorphic operations to be applied component-wise to all entries (slots) of the encrypted vector. This SIMD capability is a key advantage of BFV for efficient matrix multiplication. (See Section 8 for further details.) Finally, we formally define the three homomorphic operations (i.e., multiplication, addition, and rotation) under BFV, which are used in our evaluation section.

- $\mathsf{ct} \leftarrow \mathsf{BFV.Eval}(+, \{\mathsf{ct}_i\}_{i \in [m]})$: given a list of ciphertexts, outputs a single ciphertext ct, s.t., $\mathsf{BFV.D}(\mathsf{ct}) = \sum_{i \in [m]} \mathsf{BFV.D}(\mathsf{ct}_i)$.
- $\mathsf{ct} \leftarrow \mathsf{BFV.Eval}(\cdot, \mathsf{ct}_1, \mathsf{ct}_2)$: given two input ciphertexts, outputs a single ciphertext ct, s.t., $\forall i \in [N], \mathsf{BFV.D}(\mathsf{ct})[i] = \mathsf{BFV.D}(\mathsf{ct}_1)[i] \cdot \mathsf{BFV.D}(\mathsf{ct}_2)[i]$ (element-wise multiplication).
- $\mathsf{ct}' \leftarrow \mathsf{BFV.Rot}(\mathsf{ct}, k)$: given an input ciphertexts, outputs a ciphertext ct', s.t., $\forall i \in [N], \mathsf{BFV.D}(\mathsf{ct}')[i] = \mathsf{BFV.D}(\mathsf{ct})[i + k \mod N]$.

2.3 Distributed SNARK

Definition 2.3 (Indexed Relation for Circuit Satisfiability [56])**.** We denote by $\mathrm{R}_{\mathsf{SAT}} \ni (\mathbb{i}, \mathbb{x}, \mathbb{w})$ the circuit satisfiability relation, i.e., for all $(\mathbb{i} := (\mathsf{pp}, C), \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$, where pp is the public parameters and C is the description of an arithmetic circuit, $\mathbb{x}$ is the public inputs, and $\mathbb{w}$ is the

secret inputs, we have that $(\mathbb{x}, \mathbb{w})$ is a satisfying assignment of the circuit C under public parameter pp, i.e., $C(\mathsf{pp}, \mathbb{x}, \mathbb{w}) = 1$.

Looking ahead, in our construction, pp includes auxiliary information, such as the ring $\mathcal{R}$. Notice that for an arithmetic circuit, given the public and secret inputs, there is an efficient algorithm that recovers the complete assignment of all wires.

We now define a distributed SNARK for $\mathrm{R}_{\mathsf{SAT}}$, where the proof generation is distributed among m sub-provers.

Definition 2.4 (Distributed SNARK)**.** For $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$ and a parameter $m \in \mathbb{N}$, a distributed SNARK scheme contains the following PPT algorithms:

- $(\mathsf{VK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{i}, m)$: generate the verification key VK and the proving key PK.
- $(\overline{\mathsf{PK}}, \{(\mathsf{PK}_i, \mathbb{x}_i, \mathbb{w}_i)\}_{i\in[m]}) \leftarrow \mathsf{Distribute}(\mathsf{PK}, \mathbb{x}, \mathbb{w})$: create m sub-circuits $\{(\mathsf{PK}_i, \mathbb{x}_i, \mathbb{w}_i)\}_{i\in[m]}$ and a master proving key $\overline{\mathsf{PK}}$ based on $(\mathsf{PK}, \mathbb{x}, \mathbb{w})$.
- $\pi_i \leftarrow \mathsf{SubProve}(\mathsf{PK}_i, \mathbb{x}_i, \mathbb{w}_i)$: generate proof π_i for subcircuit $(\mathbb{x}_i, \mathbb{w}_i)$ using the proving key PK_i.
- $\pi \leftarrow \mathsf{AggProve}(\overline{\mathsf{PK}}, \{\pi_i\}_{i\in[m]})$: generate aggregated proof π given the master proving key $\overline{\mathsf{PK}}$ and the sub-proofs $\{\pi_i\}_{i\in[m]}$.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{VK}, \mathbb{x}, \pi)$: verify the proof π for instance $\mathbb{x}$ with verification key VK; output a boolean value $\{0, 1\}$, indicating if the proof is valid or not.

A distributed SNARK for $\mathrm{R}_{\mathsf{SAT}}$ satisfies the following properties:

Completeness. For all $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$, let

$$\begin{aligned}(\mathsf{VK}, \mathsf{PK}) &\leftarrow \mathsf{Setup}(1^\lambda, \mathbb{i}),\\ (\overline{\mathsf{PK}}, \{(\mathsf{PK}_i, \mathbb{x}_i, \mathbb{w}_i)\}_{i\in[m]}) &\leftarrow \mathsf{Distribute}(\mathsf{PK}, \mathbb{x}, \mathbb{w}),\\ \text{for } i \in [m], \pi_i &\leftarrow \mathsf{SubProve}(\mathsf{PK}_i, \mathbb{x}_i, \mathbb{w}_i),\\ \pi &\leftarrow \mathsf{AggProve}(\overline{\mathsf{PK}}, \{\pi_i\}_{i\in[m]}),\end{aligned}$$

we have that

$$\mathsf{Verify}(\mathsf{VK}, \mathbb{x}, \pi) = 1.$$

Knowledge Soundness. For any PPT adversary $\mathcal{A}$ there is a PPT extractor $\mathcal{E}$ such that if $(\mathbb{i}', \mathsf{aux}) \leftarrow \mathcal{A}(1^\lambda)$, $(\mathsf{VK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{i}')$, and $(\mathbb{x}', \mathbb{w}', \pi') \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{PK}, \mathsf{aux})$, where the extractor might have access to the random tape of $\mathcal{A}$, it holds that:

$$\Pr\left(\begin{array}{c}\mathsf{Verify}(\mathsf{VK}, \mathbb{x}', \pi') = 1\\ \wedge(\mathbb{i}', \mathbb{x}', \mathbb{w}') \notin \mathrm{R}_{\mathsf{SAT}}\end{array}\right) = \mathsf{negl}(\lambda)$$

Succinctness. For any $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$, we have that

$$|\pi| = \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(|\mathbb{x}| + |\mathbb{w}|),$$

which is independent of m and the circuit description included in $\mathbb{i}$.

2.4 Linear-only Encoding

Definition 2.5 (Encoding Scheme [27])**.** An encoding scheme Π_E over a ring $\mathcal{R}$, consists of algorithms $(\mathsf{Gen}, \mathsf{E})$ defined as follows:

- $(\mathsf{ek}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$: generate the secret key sk and the corresponding encoding key ek.
- $c \leftarrow \mathsf{E}(\mathsf{ek}, m)$: compute the encoding c for a message m.

An encoding scheme has to satisfy the following properties:

- *ℓ-linearly homomorphic*: There exists an efficient algorithm, denoted as "$\diamond$", that takes in the evaluation key ek, $(\mathsf{E}(a_1), \ldots, \mathsf{E}(a_\ell))$ and coefficients $c_1, \ldots, c_\ell \in \mathcal{R}$, and computes $\sum_{i\in[\ell]} c_i \diamond \mathsf{E}(a_i) := \mathsf{E}\left(\sum_{i\in[\ell]} c_i \cdot a_i\right)$.
- *Quadratic root detection*: Given the secret key sk, and the encodings $(\mathsf{E}(a_1), \ldots, \mathsf{E}(a_k))$, there exists an efficient algorithm that checks whether $Q(a_1, \ldots, a_k) = 0$ for a quadratic program $Q(x_1, \ldots, x_k) \in \mathcal{R}[X_1, \ldots, x_k]$.
- *Image verification*: There exists an efficient algorithm, denoted as IV, that takes in the secret key sk and an element c, and outputs 1 if c is a valid encoding of some element in $\mathcal{R}$; 0 otherwise.

Secure encoding. As in Rinocchio [27], an encoding scheme is *secure* if it satisfies Assumption 2.1 and 2.2 discussed below. Looking ahead, in our construction, we only consider secure encodings. In Section 5, we provide a concrete instantiation that fits our needs. Note that if E is the encoding algorithm of a secure encoding scheme, then E_r, defined as $\mathsf{E}_r(a) := \mathsf{E}(r \cdot a)$ for ring elements $r, a \in \mathcal{R}$, is also a secure encoding scheme as claimed in [27].

The two assumptions below are inherited from Rinocchio [27], which extend the classical bilinear-group knowledge assumptions formalized in the QSP framework [28]. At a high level, the falsifiable q-PDH assumption posits that, given encodings of successive powers of a secret point s for exponents in $\{0, \ldots, 2q\}\backslash\{q+1\}$, one cannot compute an encoding at the intermediate exponent $q+1$ with non-negligible probability. The non-falsifiable knowledge assumption q-PKE postulates that if the adversary is able to produce encodings satisfying certain algebraic relations, it must *know* the underlying linear combination used to construct the encodings. Looking ahead, this assumption manifests as the existence of an extractor that recovers the witness vector from our succinct proof.

Assumption 2.1 (Generalized q-PDH [27]). The generalized q-power Diffie-Hellman assumption holds for an encoding scheme Π_E if for every non-uniform PPT algorithm $\mathcal{A}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi_E.\mathsf{Gen}(1^\lambda)$, $s \xleftarrow{\$} \mathbb{A}^*$, and $\mathsf{pp} := (\mathsf{pk}, \Pi_E.\mathsf{E}(1), \Pi_E.\mathsf{E}(s), \ldots, \Pi_E.\mathsf{E}(s^q), \Pi_E.\mathsf{E}(s^{q+2}), \ldots, \Pi_E.\mathsf{E}(s^{2q}))$, let $(a, y) \leftarrow \mathcal{A}(\mathsf{pp})$, we have that:

$$\Pr\left(a \neq 0 \wedge y \in \{\Pi_E.\mathsf{E}(a \cdot s^{q+1})\}\right) \leq \frac{2q}{|\mathbb{A}^*|} + \mathsf{negl}(\lambda).$$

Assumption 2.2 (Generalized Augmented q-PKE [27]). The generalized augmented q-power knowledge of encoding assumption for an encoding scheme Π_E states the following: For all non-uniform PPT algorithms $\mathcal{A}$, let $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi_E.\mathsf{Gen}(1^\lambda)$, $s \xleftarrow{\$} \mathbb{A}^*, \alpha \xleftarrow{\$} \mathcal{R}^*$, $\mathsf{pp} := (\mathsf{pk}, \Pi_E.\mathsf{E}_r(1), \Pi_E.\mathsf{E}_r(s), \ldots, \Pi_E.\mathsf{E}_r(s^q), \Pi_E.\mathsf{E}_r(\alpha), \Pi_E.\mathsf{E}_r(\alpha s), \ldots, \Pi_E.\mathsf{E}_r(\alpha s^q))$ for an arbitrary $r \in \mathcal{R}^*$, the auxiliary input

$z \leftarrow Z(\mathsf{pp}, r)$ for any PPT algorithm Z [2], and let $(\Pi_E.\mathsf{E}_r(c), \Pi_E.\mathsf{E}_r(c')) \leftarrow \mathcal{A}(\mathsf{pp}, z)$, then there exists a non-uniform PPT extractor $\mathcal{E}_\mathcal{A}$ s.t. $(a_0, \ldots, a_q) \leftarrow \mathcal{E}_\mathcal{A}(\mathsf{pp}, z; r_\mathcal{A})$, where $r_\mathcal{A}$ is the random tape of $\mathcal{A}$ and:

$$\Pr\left(c' - \alpha c = 0 \wedge c \neq \sum_{i=0}^{q} a_i s^i\right) = \mathsf{negl}(\lambda).$$

This assumption essentially says that given the encoding tuple $(\mathsf{E}_r(c), \mathsf{E}_r(\alpha c))$, together with pp and the auxiliary input z, the extractor is able to extract a polynomial $f(x) = \sum_{i=0}^{q} a_i x^i$ of degree q s.t. $f(s) = c$. Crucially, our definition *does not* impose any stronger requirements than those in Rinocchio [27, Theorem 3], but rather it clarifies previously implicit points made in [27] and also used in the original knowledge soundness proof.

1. The public parameter pp contains encodings under the secure encoding scheme E_r instead of E. Notice that both E_r and E are secure encoding schemes, since $\mathsf{E}_r(a)$ is simply $\mathsf{E}(r \cdot a)$.

2. We remove the class $\mathcal{Z}$ of *benign* auxiliary input generators of [27] from the definition. Instead, we assume that the auxiliary input z is anything that can be computed by some PPT algorithm Z with access to pp and r (but without knowing sk, s, α). We stress that this modification is also essential for the original soundness proof of Rinocchio to hold (see [27, section 5.2]). Looking ahead, in our knowledge soundness proof (presented in Section 7), when applying the q-PKE assumption, given the encodings under E_r and r, the algorithm Z constructs the crs which contains encodings under E.

Remark 2.1. In our soundness proof (see Section 7), the adversary, which plays the role of the prover in our protocol, sometimes has access to only a subset of these encodings. I.e., instead of seeing $\big(\mathsf{E}(s^i), \mathsf{E}(\alpha s^i)\big)$ for $i \in \{0, \ldots, q\}$, the adversary receives $\big(\mathsf{E}(s^i), \mathsf{E}(\alpha s^i)\big)$ for $i \in \mathcal{S} \subseteq \{0, \ldots, q\}$. In this case, the extractor returns $(a_0, \ldots, q_q)$ s.t. $\forall i' \notin \mathcal{S}$, $a_{i'} = 0$. This is because by the linear-only nature of q-PKE assumption, the output of the adversary should only depend on the subset of the encodings it receives. This should be true even when the extractor $\mathcal{E}_\mathcal{A}$ takes as input all encodings for $i \in \{0, \ldots, q\}$. This generalization of Assumption 2.1 is used implicitly in the security proof of Rinocchio [27, section 5.2].

We present the following lemma from [27], used in our knowledge soundness proof in Section 7.

Lemma 2.1. [27, Lemma 7] Let $\mathcal{R}[x]_{\leq t}$ denote the polynomials in $\mathcal{R}[x]$ of degree at most t. Let $\mathcal{R}[x]_{\neg t}$ denote the polynomials over $\mathcal{R}[x]$ that have a zero coefficient for x^t. Let $\mathbb{A}^* \subseteq \mathcal{R}^*$ be an exceptional set. We define $\mathbb{A}^*[x]_{\leq t}, \mathbb{A}^*[x]_{\neg t}$ analogously.

Given a set $\mathcal{U} = \{u_i(x)\} \subset \mathcal{R}[x]_{\leq t}$ such that $|\mathcal{U}| = m$, let $\mathsf{span}(\mathcal{U})$ denote the set of polynomials that can be generated as $\mathcal{R}$-linear combinations of the polynomials in $\mathcal{U}$. Let $a(x) \in \mathbb{A}^*[x]_{\leq t+1}$ be generated uniformly at random subject to the constraint that $\{a(x) \cdot u_i(x) \mid u_i(x) \in \mathcal{U}\} \subset \mathcal{R}[x]_{\neg t+1}$. Let $s \xleftarrow{\$} \mathbb{A}^*$. If $t > m - 1$, for all PPT $\mathcal{A}$, we have:

$$\Pr\left(\begin{array}{r} r(x) \in \mathcal{R}[x]_{\leq t} \wedge \\ r(x) \notin \mathsf{span}(\mathcal{U}) \wedge \\ a(x) \cdot r(x) \in \mathcal{R}[x]_{\neg t+1} \end{array} \;\middle|\; r(x) \leftarrow \mathcal{A}(\mathcal{U}, s, a(s))\right) \leq \frac{1}{|\mathbb{A}^*|}.$$

[2] Notice that the auxiliary input z has a dependency on sk, s, and α limited to the extent that it can be generated efficiently from (pp, r), while sk, s, and α are *not* explicitly given to Z.

3 Technical Overview

Dinocchio is a distributed proof system for circuits over a *ring*. We begin by revisiting Rinocchio [27], the starting point of Dinocchio; then we show how to extend it to a distributed prover setting, starting from naive attempts inspired by the Hekaton paradigm [56]. A direct adaptation of these techniques, however, does not work for rings. We then present our solution, Dinocchio.

3.1 Recap of Rinocchio

Rinocchio is a designated-verifier SNARK for proving the satisfiability of QRPs. As recalled in Definition 2.2, the QRP corresponding to an arithmetic circuit C specifies $t(x)$, the target polynomial, and three polynomials for each wire, grouped into $\{V_\kappa(x), W_\kappa(x), Y_\kappa(x)\}_{\kappa\in\mathcal{I}_{\mathsf{io}}}$ and $\{V_k(x), W_k(x), Y_k(x)\}_{k\in\mathcal{I}_{\mathsf{mid}}}$, for public inputs/outputs wires and private wires respectively. All polynomials have degree d, where d is the total number of multiplication gates in the circuit. Throughout the paper, we use κ to iterate through $\mathcal{I}_{\mathsf{io}}$ and k to iterate through $\mathcal{I}_{\mathsf{mid}}$.

In Rinocchio, given a QRP specified as above, the prover wants to convince the verifier that it knows an *assignment* of wire values $\{c_k\}_{k\in\mathcal{I}_{\mathsf{mid}}}$, such that $t(x)$ divides $V(x)W(x) - Y(x)$; recall that $V(x) := V_{\mathsf{io}}(x) + V_{\mathsf{mid}}(x)$, $V_{\mathsf{mid}}(x) = \sum_{k\in\mathcal{I}_{\mathsf{mid}}} c_k V_k(x)$, and $V_{\mathsf{io}}(x) = \sum_{\kappa\in\mathcal{I}_{\mathsf{io}}} c_\kappa V_\kappa(x)$; $W(x), Y(x)$ are defined similarly. This involves proving that the polynomial $V(x)W(x) - Y(x)$ is divisible by $t(x)$, and that $V(x)_{\mathsf{mid}}, W_{\mathsf{mid}}(x), Y_{\mathsf{mid}}(x)$ are constructed from the same set of wire values.

Proof generation. First, to prove divisibility, the prover computes the quotient polynomial $H(x) = \frac{V(x)W(x)-Y(x)}{t(x)}$; the verifier samples a random $s \in \mathbb{A}^*$ and lets the prover compute $H(s), V_{\mathsf{mid}}(s)$, $W_{\mathsf{mid}}(s), Y_{\mathsf{mid}}(s)$. The verifier computes $t(s), V_{\mathsf{io}}(s), W_{\mathsf{io}}(s), Y_{\mathsf{io}}(s)$ from s, and checks that $H(s)t(s) = (V_{\mathsf{io}}(s) + V_{\mathsf{mid}}(s))(W_{\mathsf{io}}(s) + W_{\mathsf{mid}}(s)) - (Y_{\mathsf{io}}(s) + Y_{\mathsf{mid}}(s))$.

Since the random point $s \in \mathbb{A}^*$ should be hidden from the prover side, to facilitate proof generation, the crs includes the encodings $\left\{\boxed{V_k(s)}, \boxed{W_k(s)}, \boxed{Y_k(s)}\right\}_{k\in\mathcal{I}_{\mathsf{mid}}}$, and $\left\{\boxed{s^j}\right\}_{j\in[d]}$, where the solid box $\boxed{\cdot}$ is used to represent an encoding provided in the crs. These encodings are generated using a secure encoding scheme (Definition 2.5) that we denote by $\Pi_E^{(1)}$, to distinguish it from another encoding scheme, $\Pi_E^{(2)}$, to be introduced later.

The prover computes the encodings $\boxdot{V_{\mathsf{mid}}(s)} = \sum_{k\in\mathcal{I}_{\mathsf{mid}}} c_k \diamond \boxed{V_k(s)}$ where $\diamond$ denotes the multiplication operation between a plaintext and an encoding, and the dotted box $\boxdot{\cdot}$ is used to represent an encoding derived by the prover based on crs. $\boxdot{W_{\mathsf{mid}}(s)}$ and $\boxdot{Y_{\mathsf{mid}}(s)}$ are computed similarly. The prover first derives $H(x)$, then uses $\left\{\boxed{s^j}\right\}_{j\in[d]}$ and the coefficients of $H(x)$ to compute $\boxdot{H(s)}$.

To guarantee knowledge soundness, during setup we sample $\alpha_v, \alpha_w, \alpha_y$, and add $\boxed{\alpha_v V_k(s)}$, $\boxed{\alpha_w W_k(s)}, \boxed{\alpha_y Y_k(s)}$ to crs for every $k \in \mathcal{I}_{\mathsf{mid}}$. The prover must include pairs $\left(\boxdot{V_{\mathsf{mid}}(s)}, \boxdot{\alpha_v V_{\mathsf{mid}}(s)}\right)$ in the proof (same for W and Y). These pairs serve for the *α-relation check* so as to guarantee that $\{Z_{\mathsf{mid}}(x)\}$ all have degree at most d based on Assumption 2.2 (and Remark 2.1).

Secondly, to ensure that $V_{\mathsf{mid}}(s), W_{\mathsf{mid}}(s), Y_{\mathsf{mid}}(s)$ are computed from the same assignments $\{c_k\}$, during setup we sample $\gamma_v, \gamma_w, \gamma_y$ and add $\left\{\boxed{\gamma_v V_k(s) + \gamma_w W_k(s) + \gamma_y Y_k(s)}\right\}$ to crs for each $k \in \mathcal{I}_{\mathsf{mid}}$. [3] The proof then also includes a random linear combination $\boxdot{L(s)}$ that must be equal to

[3] We note that in the formal protocol, an extra randomness β is embedded in the random linear combination to facilitate the soundness proof, similar to our protocol. See Section 7.

$\boxed{\gamma_v V_{\mathsf{mid}}(s) + \gamma_w W_{\mathsf{mid}}(s) + \gamma_y Y_{\mathsf{mid}}(s)}$. Since the only way to compute $\boxed{L(s)}$ is $\sum_k c_k \diamond \boxed{\gamma_v V_k(s) + \gamma_w W_k(s) + \gamma_y Y_k(s)}$, by checking $\boxed{L(s)}$ against $\boxed{V_{\mathsf{mid}}(s)}, \boxed{W_{\mathsf{mid}}(s)}, \boxed{Y_{\mathsf{mid}}(s)}$, the verifier is convinced that the same set of $\{c_k\}_{k\in\mathcal{I}_{\mathsf{mid}}}$ is used.

Verification. After receiving a proof $\Big(\boxed{V_{\mathsf{mid}}(s)}, \boxed{\alpha_v V_{\mathsf{mid}}(s)}, \boxed{W_{\mathsf{mid}}(s)}, \boxed{\alpha_w W_{\mathsf{mid}}(s)}, \boxed{Y_{\mathsf{mid}}(s)}, \boxed{\alpha_y Y_{\mathsf{mid}}(s)}, \boxed{H(s)}, \boxed{\alpha H_{\mathsf{mid}}(s)}, \boxed{L(s)}\Big)$, the verifier first uses the secret decoding key and the *quadratic root detection* property of the underlying encoding scheme to check that all pairs of the form $\Big\{\boxed{Z_{\mathsf{mid}}(s)}, \boxed{\alpha_\zeta Z_{\mathsf{mid}}(s)}\Big\} = \big\{\boxed{x_1}, \boxed{x_2}\big\}$ satisfy the relation $\alpha_\zeta \cdot x_1 = x_2$, for $(Z, \zeta) \in \{(V, v), (W, w), (Y, y)\}$. Similarly for $\Big\{\boxed{H(s)}, \boxed{\alpha H(s)}\Big\}$. Then, the verifier computes $t(s), V_{\mathsf{io}}(s), W_{\mathsf{io}}(s)$, $Y_{\mathsf{io}}(s)$, reconstructs $Z(s) = Z_{\mathsf{io}}(s) + Z_{\mathsf{mid}}(s)$ for $Z \in \{V, W, Y\}$ and checks that:

$$\gamma_v V_{\mathsf{mid}}(s) + \gamma_w W_{\mathsf{mid}}(s) + \gamma_y Y_{\mathsf{mid}}(s) = L(s), \tag{1}$$

$$H(s) \cdot t(s) = V(s)W(s) - Y(s). \tag{2}$$

Eq. (1) ensures that the same set of wire values $\{c_k\}_{k\in\mathcal{I}_{\mathsf{mid}}}$ are used in corresponding polynomials $V_{\mathsf{mid}}(x), W_{\mathsf{mid}}(x), Y_{\mathsf{mid}}(x)$, and Eq. (2) then verifies that $V_{\mathsf{mid}}(x), W_{\mathsf{mid}}(x), Y_{\mathsf{mid}}(x)$ correspond to a valid QRP solution that satisfies the circuit C.

Complexity. In the protocol, the prover only performs plaintext-to-encoding multiplications and encoding-to-encoding additions, both supported by a *linearly homomorphic* encoding scheme. Asymptotically, it takes $O(d)$ time to calculate $\boxed{V_{\mathsf{mid}}(s)}, \boxed{W_{\mathsf{mid}}(s)}, \boxed{Y_{\mathsf{mid}}(s)}$ and $O(d \log d)$ time to derive $\boxed{H(s)}$. Hence, the runtime of the prover is $O(d \log d)$. The verifier time is constant if we consider that $|\mathcal{I}_{\mathsf{io}}|$ is small compared to d.

3.2 Towards Our Solution

We build Dinocchio, which distributes the prover computation to multiple sub-provers. Let m be the number of sub-provers. We first partition the circuit C into m smaller sub-circuits $C_1, \ldots, C_m$ in a way that as long as all sub-circuits C_i are satisfied, the original large circuit C is also satisfied. This property is formalized in Lemma 4.1. Looking ahead, this process involves adding necessary checks in each sub-circuit.

Given such a partition, a straightforward idea is to run m instances of Rinocchio in parallel and concatenate the m proofs into a single proof. However, this violates the *succinctness* definition of a distributed proof system (Definition 2.4), which states that the size of the proof should be independent of the number of sub-provers. To this end, we introduce a master prover, denoted as $\mathcal{P}_{\mathcal{M}}$, who *aggregates* the sub-proofs into a final proof π whose size is independent of m. Ensuring the succinctness of the aggregated proof and the efficiency of $\mathcal{P}_{\mathcal{M}}$ constitute the main technical challenges of our work. Similar to all prior works on distributed proofs, we do not allow communication among the sub-provers or between the sub-provers and the verifier. To reduce clutter, in the following demonstration, we omit the encodings required by the knowledge soundness argument.

Recall that our goal is to check whether $V_i(x)W_i(x) - Y_i(x) = H_i(x)t_i(x)$ for each sub-circuit C_i for $i \in [m]$ with a proof whose size is independent of m. Our main idea is to merge the checks into the following check:

$$\sum_{i\in[m]} x^{(i-1)2d} V_i(x)W_i(x) - \sum_{i\in[m]} x^{(i-1)2d} Y_i(x) = \sum_{i\in[m]} x^{(i-1)2d} H_i(x)t_i(x)\ . \tag{3}$$

Based on the generalized Schwartz-Zippel lemma over rings [27, Lemma 2], checking Eq. (3) on a random point s implies that for all i, $V_i(x)W_i(x) - Y_i(x) = H_i(x)t_i(x)$ with soundness error $\frac{2md}{|\mathbb{A}^*|}$, as long as V_i, W_i, Y_i, H_i all have degrees at most d. Now our challenge is to construct a *succinct* proof for Eq. (3) with m sub-provers.

Naive attempt via direct multiplication. Let us first show why extending Hekaton directly is not feasible in our case. For ease of exposition, we ignore the IO inputs of each sub-circuit and write $V_i(x) = V_{i,\mathsf{mid}}(x), W_i(x) = W_{i,\mathsf{mid}}(x), Y_i(x) = Y_{i,\mathsf{mid}}(x)$. To help $\mathcal{P}_{\mathcal{M}}$ evaluate Eq. (3) on the secret point s chosen by the verifier, a first attempt is to have sub-provers compute $\boxed{s^{(i-1)d}V_i(s)}$, $\boxed{s^{(i-1)d}W_i(s)}$, $\boxed{s^{(i-1)2d}Y_i(s)}$. In our setting, however, the master prover is not able to directly multiply $\boxed{s^{(i-1)d}V_i(s)}$ and $\boxed{s^{(i-1)d}W_i(s)}$ for Eq. (3), since the encoding scheme is *linear-only*.

An intuitive way to resolve this "multiplication" issue is to include $\left\{\boxed{s^{(i-1)2d}V_{i,k}(s)W_{i,k'}(s)}\right\}_{k,k'\in\mathcal{I}_{i,\mathsf{mid}}}$ in the crs, so that each sub-prover can derive $\boxed{s^{(i-1)2d}V_i(s)W_i(s)} := \sum_{k,k'}(c_{i,k}c_{i,k'})\diamond\boxed{s^{(i-1)2d}V_{i,k}(s)W_{i,k'}(s)}$. However, it is still unclear how to prove the product relation among the encodings $\left(\boxed{s^{(i-1)d}V_i(s)}, \boxed{s^{(i-1)d}W_i(s)}, \boxed{s^{(i-1)2d}V_i(s)W_i(s)}\right)$. A separate issue is consistency checks, as in Eq. (1). The sub-prover needs to show that the encoding $\boxed{s^{(i-1)2d}V_i(s)W_i(s)}$ is derived from $V_i(x)$ and $W_i(x)$ that are computed using the *same* wire values $\{c_{i,k}\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}$, i.e., $V_i(x) = \sum_k c_{i,k}V_{i,k}(x)$ and $W_i(x) = \sum_k c_{i,k}W_{i,k}(x)$. In Rinocchio, this consistency is enforced through the random linear combination check, briefly recalled as follows. During setup we provide encodings $\boxed{L_k(s)} := \boxed{\gamma_v V_k(s) + \gamma_w W_k(s) + \gamma_y Y_k(s)}$ for $k \in \mathcal{I}_{\mathsf{mid}}$, and then the prover computes $\boxed{L(s)} := \sum_k c_k \diamond \boxed{L_k(s)}$. If $\boxed{L(s)}$ is indeed a linear combination of $\boxed{L_k(s)}$ w.r.t. $\{c_k\}$, the prover is compelled to form the polynomials $V_{\mathsf{mid}}(x) = \sum_{k\in\mathcal{I}_{\mathsf{mid}}} c_k V_k(x), W_{\mathsf{mid}}(x) = \sum_{k\in\mathcal{I}_{\mathsf{mid}}} c_k W_k(x)$ using the same set $\{c_k\}$. However, this check fundamentally depends on the fact that every term manipulated by the prover is *linear* in the coefficients $\{c_k\}_{k\in\mathcal{I}_{\mathsf{mid}}}$. If instead the prover submits an encoding of the form $\boxed{V_{\mathsf{mid}}(s)W_{\mathsf{mid}}(s)}$, the coefficients of $V_{\mathsf{mid}}(x)W_{\mathsf{mid}}(x)$ lie in the *quadratic* span of $\{c_k\}$. It is unclear how such nonlinearity can be captured in Rinocchio's checks. For example, assume $V_i(x) = c\cdot(x+1)$ and $W_i(x) = c'\cdot(x+1)$, then with $V_i(x)W_i(x) = 4\cdot(x+1)^2$, it could be either $c = c' = 2$ or $c = 1, c' = 4$, while the latter violates consistency.

Our solution. To address these challenges, we let the master use an *aggregation circuit* (AC) to prove the inner product relationship. Roughly speaking, AC is a circuit that checks Eq. (3). Let $v'_i = s^{(i-1)d}(V_{i,\mathsf{io}}(s) + V_{i,\mathsf{mid}}(s))$, similarly we define w'_i, y'_i, h'_i. As illustrated in Fig. 1, AC takes as input the encodings $\left(\boxed{v'_i}, \boxed{w'_i}, \boxed{y'_i}, \boxed{h'_i}, \boxed{t_i}\right)$ from each sub-prover $\mathcal{P}_i$ and computes the following:

$$\sum_{i\in[m]} \left(\boxed{v'_i} \otimes \boxed{w'_i} - \boxed{y'_i} \otimes \boxed{1} - \boxed{h'_i} \otimes \boxed{t_i}\right). \tag{4}$$

Note that if the sub-proofs satisfy Eq. (3), then Eq. (4) should output (an encoding of) zero.

We now explain the product operation $\otimes$ between encodings. As in Rinocchio [27, Section 7.2], the encoding scheme is instantiated with Regev-style encryption (see also Section 5). Therefore, for $\boxed{v'_i} := (a_1, b_1)$ and $\boxed{w'_i} := (a_2, b_2)$, $\otimes$ represents the following computation: $\boxed{v'_i} \otimes \boxed{w'_i} := (a_1a_2, a_1b_2 + a_2b_1, b_1b_2)$. The result encodes m_1m_2 under the *extended* secret key $(1, \mathsf{sk}, \mathsf{sk}^2)$. Therefore, the

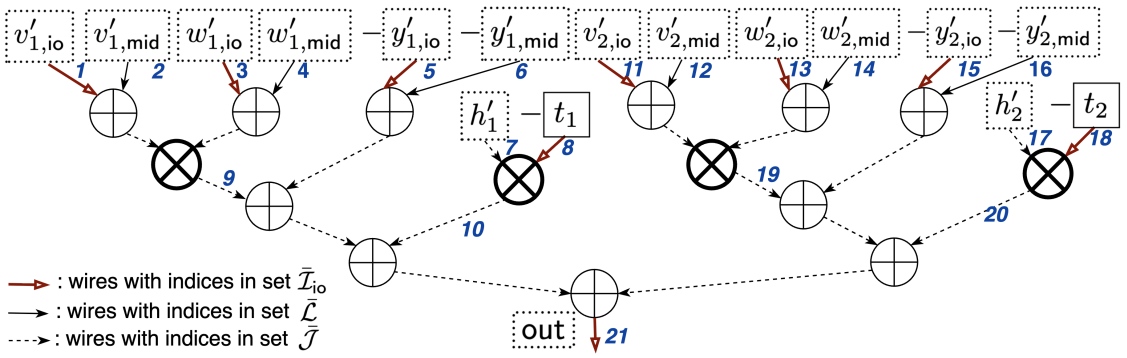

Figure 1: An illustration for the aggregation circuit with $m = 2$, with inputs defined w.r.t. our proof scheme. Notice that each multiplication gate represents an $\otimes$ operation which takes two encodings $\in \mathcal{R}^2$ as inputs and derives an *extended* encoding as the output. Notice that in QRP, the addition gates are handled for free, so we only label wires that are inputs or outputs of the multiplication gates and the final output of the circuit.

output encoding derived in Eq. (4) is an *extended* encoding with three ring elements, denoted by a double-bashed box $\boxed{\boxed{\cdot}}$.

At this point, one might ask how the master prover can prove the honest evaluation of Eq. (4), since all operands are already encodings under $\Pi_E^{(1)}$, rather than plain ring elements. To overcome this issue, we introduce a second encoding scheme $\Pi_E^{(2)}$, whose plaintext space is the same as the encoding space of $\Pi_E^{(1)}$. This is similar to the use of Rinocchio for verifying FHE computation, which is represented by a circuit C over the ciphertext space $\mathcal{R}_q$. The encoding scheme in Rinocchio directly operates on the FHE ciphertext by setting the plaintext space of the encoding scheme to be the FHE ciphertext space. This *layered* use of encodings also occurs in AC: The computation is performed over ring elements that are themselves encodings under a linear-only encoding scheme. Consequently, the master prover proves the honest computation, including the operation "$\otimes$" mentioned above, over those encodings under $\Pi_E^{(1)}$ by invoking Rinocchio instantiated with the encoding scheme $\Pi_E^{(2)}$. A formal definition is presented in Definition 4.1.

With AC specified, the prover side must prove the following two statements: 1) the inputs and outputs of AC satisfy the circuit, and 2) the input, which contains the sub-proofs $\left\{\boxed{v'_i}, \boxed{w'_i}, \boxed{y'_i}, \boxed{h'_i}, \boxed{t_i}\right\}$, is consistent with the wire assignments $\{c_{i,k}\}_{k\in\mathcal{I}_{i,\text{mid}}}$ of sub-circuit C_i. I.e., the sub-proof for each sub-circuit passes the consistency checks.

1) Proving satisfiability for AC. Recall that honestly generated sub-proofs π_i consist of $\boxed{v'_i} := \boxed{s^{(i-1)d} V_{i,\text{mid}}(s)}$, $\boxed{w'_i} := \boxed{s^{(i-1)d} W_{i,\text{mid}}(s)}$, $\boxed{y'_i} := \boxed{s^{(i-1)2d} Y_{i,\text{mid}}(s)}$, and $\boxed{h_i}$. These sub-proofs $\{\pi_i\}_{i\in[m]}$ are submitted to the master prover $\mathcal{P}_\mathcal{M}$, who treats them as *plaintext* inputs to AC. $\mathcal{P}_\mathcal{M}$ first evaluates the aggregation circuit using the encodings in $\{\pi_i\}$ and then runs Rinocchio to prove that the inputs satisfy AC.

While this proof is a standard Rinocchio proof, we emphasize that we must use an encoding scheme $\Pi_E^{(2)}$ whose plaintext space is the encoding space of the encoding scheme used for the sub-proofs, which we denote with $\Pi_E^{(1)}$. From this point on, we add the subscripts "1" and "2" in encodings from $\Pi_E^{(1)}$ and $\Pi_E^{(2)}$ respectively.

We denote by $\left(\{\bar{V}_{\bar{k}}(x), \bar{W}_{\bar{k}}(x), \bar{Y}_{\bar{k}}(x)\}_{\bar{k}\in\bar{\mathcal{I}}}, \bar{t}(x)\right)$ the QRP of AC, where $\bar{\mathcal{I}}$ is the index set of all wires in AC. Given the above inputs, $\mathcal{P}_{\mathcal{M}}$ derives wire values for AC, denoted by $\{\bar{c}_{\bar{k}}\}$ for each wire index $\bar{k} \in \bar{\mathcal{I}}_{\mathsf{mid}}$. Then, $\mathcal{P}_{\mathcal{M}}$ generates a Rinocchio proof: it computes $\boxed{\bar{V}_{\mathsf{mid}}(\bar{s})}_2 := \sum_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}} \bar{c}_{\bar{k}} \cdot \boxed{\bar{V}_{\bar{k}}(\bar{s})}_2$, and similarly $\boxed{\bar{W}_{\mathsf{mid}}(\bar{s})}_2, \boxed{\bar{Y}_{\mathsf{mid}}(\bar{s})}_2$, where $\bar{s}$ is a random evaluation point chosen by the verifier as in Rinocchio. It computes the quotient polynomial $\bar{H}(x)$ and derives $\boxed{\bar{H}(\bar{s})}_2$. The proof for the satisfiability of AC consists of the encodings $\boxed{\bar{V}_{\mathsf{mid}}(\bar{s})}_2, \boxed{\bar{W}_{\mathsf{mid}}(\bar{s})}_2, \boxed{\bar{Y}_{\mathsf{mid}}(\bar{s})}_2$ and $\boxed{\bar{H}(\bar{s})}_2$, just like a regular Rinocchio proof, and the verification is identical to Rinocchio, i.e.:

$$\left(\bar{V}_{\mathsf{io}}(\bar{s}) + \bar{V}_{\mathsf{mid}}(\bar{s})\right)\left(\bar{W}_{\mathsf{io}}(\bar{s}) + \bar{W}_{\mathsf{mid}}(\bar{s})\right) - \left(\bar{Y}_{\mathsf{io}}(\bar{s}) + \bar{Y}_{\mathsf{mid}}(s)\right) = \bar{H}(\bar{s})\bar{t}(\bar{s}), \tag{5}$$

where $\bar{V}_{\mathsf{mid}}(\bar{s}) = \sum_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}} \bar{c}_{\bar{k}}\bar{V}_{\bar{k}}(\bar{s})$ (similar to $\bar{W}_{\mathsf{mid}}(\bar{s}), \bar{Y}_{\mathsf{mid}}(\bar{s})$) and $\{\bar{c}_{\bar{k}}\}$ are encodings under $\Pi_E^{(1)}$, casted to plaintexts under $\Pi_E^{(2)}$.

2) Augmented consistency check. The above proof shows that *some* inputs satisfy the AC circuit, but it does not prove that the inputs are valid sub-proofs for sub-circuits. Key to the validity of sub-proofs is the consistency requirement: $\boxed{v'_i}_1, \boxed{w'_i}_1, \boxed{y'_i}_1$ are themselves derived w.r.t. a consistent set of wire values of the sub-circuit C_i.

Absent this check, we do not have soundness. As a toy example, for the leftmost multiplication gate in Fig. 1 assume that we have $\boxed{v'_{1,\mathsf{io}}}_1 = \boxed{w'_{1,\mathsf{io}}}_1 = \boxed{0}_1$, then a malicious $\mathcal{P}_{\mathcal{M}}$ can simply ignore the sub-prover's inputs $\left(\boxed{v'_{1,\mathsf{mid}}}_1, \boxed{w'_{1,\mathsf{mid}}}_1\right)$, replace them with $\left(\boxed{1}_1, \boxed{1}_1\right)$ as the two inputs and set the output to $\boxed{1}_1$. This assignment indeed deduces a satisfying and consistent wire assignment for AC, but might not be a valid proof for the original circuit. Thus, we augment the consistency check as follows.

We divide the wires of AC into two sets $\bar{\mathcal{L}}$ and $\bar{\mathcal{J}}$, as shown in Fig. 1. We denote by $\bar{\mathcal{L}}$ the set of indices for the input wires carrying the values $\left\{\boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1\right\}$. Let $\bar{\mathcal{J}}$ denote the wires that do not belong to $\bar{\mathcal{L}}$ and are not from public inputs. For the wires in the set $\bar{\mathcal{J}}$, we apply the consistency check directly inherited from Rinocchio to prove that $\left(\bar{V}_{\mathsf{mid}}(x), \bar{W}_{\mathsf{mid}}(x), \bar{Y}_{\mathsf{mid}}(x)\right)$ are constructed w.r.t. a consistent wire assignments $\{\bar{c}_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{J}}}$. In particular, for each wire with index $\bar{k} \in \bar{\mathcal{J}}$, the crs contains $\left\{\boxed{\gamma_v\bar{V}_{\bar{k}}(\bar{s}) + \gamma_w\bar{W}_{\bar{k}}(\bar{s}) + \gamma_y\bar{Y}_{\bar{k}}(\bar{s})}_2\right\}_{\bar{k}\in\bar{\mathcal{J}}}$ for random γ_v, γ_w, and γ_y chosen by the verifier. Then, the master prover computes the sub-term $\boxed{L_{\bar{\mathcal{J}}}(\bar{s})}_2 := \sum_{\bar{k}\in\bar{\mathcal{J}}} \bar{c}_{\bar{k}} \diamond \boxed{\gamma_v\bar{V}_{\bar{k}}(\bar{s}) + \gamma_w\bar{W}_{\bar{k}}(\bar{s}) + \gamma_y\bar{Y}_{\bar{k}}(\bar{s})}_2$.

The checks of the set $\bar{\mathcal{L}}$ are more involved. Recall that for the wires with indices in $\bar{\mathcal{L}}$ carrying values $\left\{\boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1\right\}_{i\in[m]}$, parsed from sub-proofs, we need to verify that these triples are derived from consistent wire assignments from their sub-circuits: i.e., for each $i \in [m]$, the same wire assignments $\{c_{i,k}\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}$ are used in $\boxed{v'_{i,\mathsf{mid}}}_1 = \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{V_{i,k}(s)}_1$, $\boxed{w'_{i,\mathsf{mid}}}_1 = \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{W_{i,k}(s)}_1$, and $\boxed{y'_{i,\mathsf{mid}}}_1 = \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{Y_{i,k}(s)}_1$.

According to the definition of QRP, $\bar{V}_{\bar{\mathcal{L}}}(\bar{s})$ (similarly $\bar{W}_{\bar{\mathcal{L}}}(\bar{s})$ and $\bar{Y}_{\bar{\mathcal{L}}}(\bar{s})$) is computed as $\sum_{\bar{k}\in\bar{\mathcal{L}}} c_{\bar{k}}\bar{V}_{\bar{k}}(\bar{s})$, but we can arrange the sum terms to make explicit the relationship between $c_{\bar{k}}$ and sub-proofs

$v'_{i,\mathsf{mid}}$,$w'_{i,\mathsf{mid}}$, and $y'_{i,\mathsf{mid}}$:

$$\boxed{\bar{V}_{\bar{\mathcal{L}}}(\bar{s})}_2 := \sum_{i\in[m]} \boxed{\sharp v'_{i,\mathsf{mid}} \sharp} \diamond \boxed{\bar{V}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 + \boxed{\sharp w'_{i,\mathsf{mid}} \sharp} \diamond \boxed{\bar{V}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 + \boxed{\sharp y'_{i,\mathsf{mid}} \sharp} \diamond \boxed{\bar{V}_{\mathsf{Idx}_y(i)}(\bar{s})}_2, \quad (6)$$

where the mapping function $\mathsf{Idx}_\zeta(i)$ outputs the wire index in AC that carries the value $\left\{\boxed{\zeta'_{i,\mathsf{mid}}}_1\right\}$ for $\zeta \in \{v, w, y\}$. With a formal definition deferred to Definition 4.1, we refer readers to examples in Fig. 1: we have that $\mathsf{Idx}_v(1) = 2$ (since wire 2 carries the value $\boxed{v'_{1,\mathsf{mid}}}_1$), $\mathsf{Idx}_w(2) = 14, \mathsf{Idx}_y(2) = 16$. Operation $\boxed{\sharp \cdot \sharp}$ represents the casting from an encoding to a plaintext.

The honest prover needs to guarantee the consistency among the wire values used for constructing $\left\{\boxed{\bar{V}_{\bar{\mathcal{L}}}(\bar{s})}_2, \boxed{\bar{W}_{\bar{\mathcal{L}}}(\bar{s})}_2, \boxed{\bar{Y}_{\bar{\mathcal{L}}}(\bar{s})}_2\right\}$. Based on Rinocchio, $\mathcal{P}_{\mathcal{M}}$ has to evaluate the sub-term $\boxed{L_{\bar{\mathcal{L}}}(\bar{s})}_2$ as:

$$\sum_{i\in[m]} \sum_{\zeta\in\{v,w,y\}} \left(\boxed{\sharp \zeta'_{i,\mathsf{mid}} \sharp} \diamond \boxed{\gamma_v \bar{V}_{\mathsf{Idx}_\zeta(i)}(\bar{s}) + \gamma_w \bar{W}_{\mathsf{Idx}_\zeta(i)}(\bar{s}) + \gamma_y \bar{Y}_{\mathsf{Idx}_\zeta(i)}(\bar{s})}_2 \right). \quad (7)$$

However, as we mentioned before, this check alone is not sufficient as a malicious prover can replace the sub-proofs $\left\{\boxed{\zeta'_{i,\mathsf{mid}}}_1\right\}$ with other values when deriving $\boxed{L_{\bar{\mathcal{L}}}(\bar{s})}_2$. To rule out this attack, first recall that:

$$v'_{i,\mathsf{mid}} := s^{(i-1)d} V_i(s) = s^{(i-1)d} \sum_{k\in\mathcal{I}_{\mathsf{mid}}} c_{i,k} V_{i,k}(s) = \sum_{k\in\mathcal{I}_{\mathsf{mid}}} c_{i,k} V'_{i,k}(s),$$

where $V'_{i,k}(s) = s^{(i-1)d} V_{i,k}(s)$. Similar expressions hold for $w'_{i,\mathsf{mid}}$ and $y'_{i,\mathsf{mid}}$. Hence, substituting these expressions in Eq. (7) and using the linear homomorphism on the encoding, we have that:

$$\begin{aligned}
\boxed{L_{\bar{\mathcal{L}}}(\bar{s})}_2 &= \sum_{i\in[m]} \sum_{\substack{(\zeta,Z)\in\{(v,V),\\(w,W),(y,Y)\}}} \left(\boxed{\sharp \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} Z'_{i,k}(s) \sharp} \diamond \boxed{\gamma_v \bar{V}_{\mathsf{Idx}_\zeta(i)}(\bar{s}) + \gamma_w \bar{W}_{\mathsf{Idx}_\zeta(i)}(\bar{s}) + \gamma_y \bar{Y}_{\mathsf{Idx}_\zeta(i)}(\bar{s})}_2 \right) \\
&= \sum_{i\in[m]} \sum_{\substack{(\zeta,Z)\in\{(v,V),\\(w,W),(y,Y)\}}} \left(\boxed{\sharp \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} V'_{i,k}(s) \sharp} \diamond \boxed{\gamma_\zeta \bar{Z}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 \right. \\
&\quad \left. + \boxed{\sharp \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} W'_{i,k}(s) \sharp} \diamond \boxed{\gamma_\zeta \bar{Z}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 + \boxed{\sharp \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} Y'_{i,k}(s) \sharp} \diamond \boxed{\gamma_\zeta \bar{Z}_{\mathsf{Idx}_y(i)}(\bar{s})}_2 \right). \quad (8)
\end{aligned}$$

We now define the *entangled encodings* for $Z \in \{V, W, Y\}$:

$$\boxed{\Psi^Z_{i,k}(\bar{s})}_2 := \boxed{\sharp \boxed{V'_{i,k}(s)} \sharp} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 + \boxed{\sharp \boxed{W'_{i,k}(s)} \sharp} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 + \boxed{\sharp \boxed{Y'_{i,k}(s)} \sharp} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_y(i)}(\bar{s})}_2, \quad (9)$$

where $i \in [m], k \in \mathcal{I}_{i,\mathsf{mid}}$ [4]. Then each sub-prover can compute:

$$\boxed{L_{i,\bar{\mathcal{L}}}(\bar{s})}_2 := \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\gamma_v \Psi^V_{i,k}(\bar{s}) + \gamma_w \Psi^W_{i,k}(\bar{s}) + \gamma_y \Psi^Y_{i,k}(\bar{s})}_2, \quad (10)$$

[4]Note that here only "$\bar{s}$" is the variable to the entangled polynomial $\Psi^Z_{i,k}(x)$, since the encoding $\boxed{Z'_{i,k}(s)}_1$ with hidden variable s is treated as plaintext under $\Pi^{(2)}_E$.

given the encodings $\boxed{\gamma_v \Psi^V_{i,k}(\bar{s}) + \gamma_w \Psi^W_{i,k}(\bar{s}) + \gamma_y \Psi^Y_{i,k}(\bar{s})}_2, k \in \mathcal{I}_{i,\mathsf{mid}}$ in the crs. Intuitively, based on Eq. (8), we can see that $\left\{\boxed{L_{i,\bar{\mathcal{L}}}(\bar{s})}_2\right\}$ are the components of $\boxed{L_{\bar{\mathcal{L}}}(\bar{s})}_2$ which depend on the wire assignments of sub-circuit C_i. Finally, the sub-proofs include $\boxed{L_{i,\bar{\mathcal{L}}}(\bar{s})}_2$ in addition to $\left(\boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1, \boxed{h'_i}_1\right)$, and $\mathcal{P}_{\mathcal{M}}$ computes:

$$\boxed{L(\bar{s})}_2 = \boxed{L_{\bar{\mathcal{L}}}(\bar{s})}_2 + \boxed{L_{\bar{\mathcal{J}}}(\bar{s})}_2 = \sum_{i\in[m]} \boxed{L_{i,\bar{\mathcal{L}}}(\bar{s})}_2 + \boxed{L_{\bar{\mathcal{J}}}(\bar{s})}_2, \tag{11}$$

and appends it to the final proof.

This new consistency check does not allow the prover to freely decide the value $\boxed{\zeta'_{i,\mathsf{mid}}}_1 := \sum_k c_{i,k} \diamond \boxed{Z'_{i,k}(s)}_1$, rather each individual component $\boxed{Z'_{i,k}(s)}_1$ is absorbed into the entangled encoding $\boxed{\Psi^Z_{i,k}(\bar{s})}_2$, fixed in the crs as in Eq. (9). The prover can only choose the scalars $\{c_{i,k}\}$ to be multiplied with the entangled encodings $\left\{\boxed{\Psi^Z_{i,k}(\bar{s})}_2\right\}$. Thus, the verifier will perform a similar check to the consistency check in Rinocchio (Eq. (1)) to enforce that the sub-provers use a single, consistent wire assignment across V'_i, W'_i, Y'_i.

An important technical detail for the above consistency check is that the encoding scheme must satisfy *associativity* of plaintext-to-encoding multiplication, which we formalize and construct in Section 5. Now we focus on illustrating the challenge. Comparing Eq. (6) and Eq. (11), we observe a crucial difference in the computation of the encodings $\left(\boxed{\bar{V}_{\bar{\mathcal{L}}}(\bar{s})}_2, \boxed{\bar{W}_{\bar{\mathcal{L}}}(\bar{s})}_2, \boxed{\bar{Y}_{\bar{\mathcal{L}}}(\bar{s})}_2\right)$ and the sub-term $\boxed{L_{\bar{\mathcal{L}}}(\bar{s})}_2$. In Eq. (6), $\boxed{\bar{V}_{\bar{\mathcal{L}}}(\bar{s})}_2$ is computed as:

$$\begin{aligned}
&\sum_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}} \left(\boxed{v'_{i,\mathsf{mid}}} \diamond \boxed{\bar{V}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 + \boxed{w'_{i,\mathsf{mid}}} \diamond \boxed{\bar{V}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 + \boxed{y'_{i,\mathsf{mid}}} \diamond \boxed{\bar{V}_{\mathsf{Idx}_y(i)}(\bar{s})}_2 \right) \\
&= \sum_{i\in[m]} \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} \left(\left(c_{i,k} \diamond \boxed{V'_{i,k}(s)}_1 \right) \diamond \boxed{\bar{V}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 + \left(c_{i,k} \diamond \boxed{W'_{i,k}(s)}_1 \right) \diamond \boxed{\bar{V}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 \right. \\
&\quad \left. + \left(c_{i,k} \diamond \boxed{Y'_{i,k}(s)}_1 \right) \diamond \boxed{\bar{V}_{\mathsf{Idx}_y(i)}(\bar{s})}_2 \right).
\end{aligned} \tag{12}$$

While in Eq. (11), the sub-term $\boxed{\bar{V}_{\bar{\mathcal{L}}}(\bar{s})}_2$ embedded in $\boxed{L_{\bar{\mathcal{L}}}(\bar{s})}_2$ is computed as:

$$\begin{aligned}
\sum_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\bar{V}_{i,k}(\bar{s})}_2 = \sum_{i\in[m]} \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} \left(c_{i,k} \diamond \left(\boxed{V'_{i,k}(s)} \diamond \boxed{\bar{V}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 \right) \right. \\
\left. + c_{i,k} \diamond \left(\boxed{W'_{i,k}(s)} \diamond \boxed{\bar{V}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 \right) + c_{i,k} \diamond \left(\boxed{Y'_{i,k}(s)} \diamond \boxed{\bar{V}_{\mathsf{Idx}_y(i)}(\bar{s})}_2 \right) \right).
\end{aligned} \tag{13}$$

We want Eq. (12) and Eq. (13) to encode the same value. [5]

[5] We remark that $\boxed{a}_1$ is a Π_1 encoding containing two ring elements (r_1, r_2) and we define the multiplication with

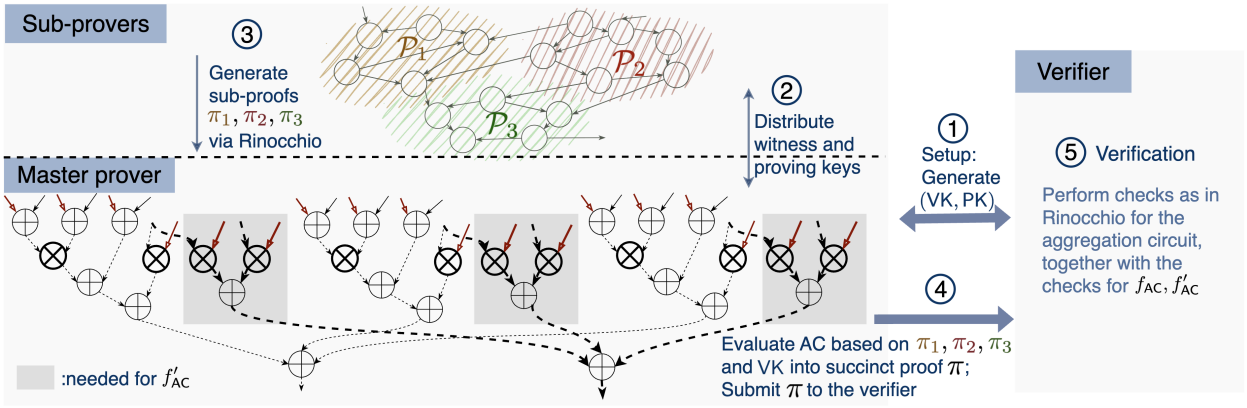

Figure 2: High-level overview of Dinocchio protocol. We assume $m = 3$ sub-provers. The inputs of the aggregation circuit are not explicitly written out. We refer readers to Fig. 1 and Fig. 3 for more details on how AC evaluates f_{AC} and f'_{AC}.

However, this is not necessarily true for any encoding satisfying Definition 2.5. The same issue appears in the computation of $\boxed{\bar{W}_{\bar{\mathcal{L}}}(\bar{s})}_2$ and $\boxed{\bar{Y}_{\bar{\mathcal{L}}}(\bar{s})}_2$. To address this barrier, we require encodings to satisfy *associativity* of plaintext-to-encoding multiplication; we formalize it and present an associative encoding based on the ring LWE assumptions in Section 5.

Degree bound checks. Recall that our ultimate goal is to use AC to evaluate Eq. (3) on a random point and prove its honest evaluation using Rinocchio. However, proving that Eq. (3) evaluates to zero on some random "s" does not directly imply that $V_i(x)W_i(x) - Y_i(x) = H_i(x)t_i(x)$ for all sub-circuits because some terms might cancel out. On the contrary, let $f_i(x) = x^{(i-1)2d}\left(V_i(x)W_i(x) - Y_i(x) - H_i(x)t_i(x)\right)$, if the polynomials $f_i(x)$ all have disjoint monomials (i.e., for each j, x^j appears in at most one of these polynomials $f_i(x)$), then proving that Eq. (3) evaluates to zero on s does imply that $f_i(x) = 0$ for all i with soundness error $\frac{2md}{|\mathbb{A}^*|}$. Consequently, the AC should be augmented to embed the *checks* that all sub-polynomials $V_i(x), W_i(x), Y_i(x), H_i(x)$ contributing to $f_i(x)$ have degrees at most d. Thus, if the prover demonstrates the satisfiability of the augmented AC, the verifier is convinced that $\forall i, V_i(x)W_i(x) - Y_i(x) = H_i(x)t_i(x)$ with overwhelming probability.

Before introducing our check, we first recall that Rinocchio achieves this via the *α-relation check*. As aforementioned, the Rinocchio prover includes the tuple $\left(\boxed{Z_i(s)}, \boxed{\alpha_{i,\zeta}Z_i(s)}\right)$ (or $\left(\boxed{H_i(s)}, \boxed{\alpha_{i,h}H_i(s)}\right)$) in the proof, constructed using $\left\{\boxed{s^j}, \boxed{\alpha_{i,\zeta}s^j}, \boxed{\alpha_{i,h}s^j}\right\}$ in the crs, where $\alpha_{i,\zeta}$ and $\alpha_{i,h}$ are sampled during setup. By invoking the d-PKE assumption (see Assumption 2.2) in the security analysis, all polynomials $V_i(x), W_i(x), Y_i(x), H_i(x)$ are guaranteed to have degree at most d.

Analogously, in our case, naively the sub-provers need to generate the encoding tuples: $\left\{\left(\boxed{Z_i(s)}_1, \boxed{\alpha_{i,\zeta}Z_i(s)}_1\right)\right\}, \left(\boxed{H_i(s)}_1, \boxed{\alpha_{i,h}H_i(s)}_1\right)$ for the α-relation checks for $Z \in \{V, W, Y\}$ and $i \in [m]$. However, notice that as shown in Eq. (12), $\boxed{\bar{Z}_{\mathsf{mid}}(\bar{s})}_2$ is derived based on $\left\{\boxed{\zeta'_{i,\mathsf{mid}}}_1\right\}$, so enforcing the standard Rinocchio-style checks on $\{\bar{Z}_{\mathsf{mid}}(x)\}$ already guarantees a consistent set of coefficients for

$\boxed{b}_2$ under Π_2 as two plaintext-to-encoding multiplications, i.e., $\boxed{a} \diamond \boxed{b}_2 = (r_1 \diamond \boxed{b}_2, r_2 \diamond \boxed{b}_2)$. In Definition 4.2, we denote this operation by $\diamond'$. For simplicity and readability, we abstract away this distinction in the overview and use $\diamond$ instead.

constructing $\{\bar{V}_{\mathsf{mid}}(x), \bar{W}_{\mathsf{mid}}(x), \bar{Y}_{\mathsf{mid}}(x)\}$. And then due to the underlying structure of the entangled encodings, in the soundness proof, the extractor derives the polynomials $\{Z_{i,\mathsf{mid}}(x)\}$ of degree at most d. See Section 7 for details.

Hence, the sub-provers are only required to generate the tuple $\left(\boxed{H_i(s)}_1, \boxed{\alpha_{i,h} H_i(s)}_1\right)$. At the same time, to preserve *succinctness*, the prover side cannot send all these individual tuples to the verifier. Thus, the challenge is to batch these checks into a single check. Our solution is quite intuitive: for each $H_i(s), i \in [m]$, we introduce a new multiplication gate in AC that takes $\boxed{H_i(s)}_1$ as the left input and $\boxed{\boxed{\alpha_{i,h}}}_1$ as the right input. Subtracting $\boxed{\alpha_{i,h} H_i(s)}_1$ from the result of the multiplication gate should give an encoding of zero.

Crucially, these checks are tightly coupled with the evaluation of the original Eq. (3), i.e., the same values $\left\{\boxed{H_i(s)}_1\right\}$ are used both in the satisfiability check and in the corresponding α-relation check. In AC, this coupling is enforced by sharing the same wires in both checks. For instance, assume that the k-th wire in AC carries the value $\boxed{H_i(s)}_1$. In the QRP representation of AC, this wire is encoded by the sub-polynomial $\bar{V}_k(x)$, indicating whether it is a left input to the multiplication gates. To ensure that the same wire serves as the input to both the multiplication gate g_1 in the satisfiability check and g_2 in its corresponding α-relation check, we have that $\bar{V}_k(g_1) = \bar{V}_k(g_2) = 1$, thereby guaranteeing that both checks operate on the same wire value $\boxed{H_i(s)}_1$. A simple illustration is demonstrated in Fig. 3. The augmented AC contains m such checks.

For the batch verification, we observe that if $\alpha_i \in \mathbb{A}^*$ is randomly sampled s.t. $\sum_i \alpha_i x_i = 0$, then $\forall i, x_i = 0$, except with probability $\frac{1}{|\mathbb{A}^*|}$. In the encodings $\boxed{x_{i,h}}_1 := \boxed{H_i(s)}_1 \otimes \boxed{\boxed{\alpha_{i,h}}}_1 - \boxed{\alpha_{i,h} H_i(s)}_1 \otimes \boxed{\boxed{1}}_1$, where a double box denotes the extended encoding derived from the operation $\otimes$, we directly have $\alpha_{i,h}$ serve as the random scalars. Thus, it suffices for the master prover to aggregate all subtraction results via an addition gate in the augmented AC:

$$\boxed{\mathsf{out}'}_1 := \sum_i \boxed{x_{i,h}}_1 = \sum_i \left(\boxed{H_i(s)}_1 \otimes \boxed{\boxed{\alpha_{i,h}}}_1 - \boxed{\alpha_{i,h} H_i(s)}_1 \otimes \boxed{\boxed{1}}_1\right). \tag{14}$$

Finally, this single encoding $\boxed{\mathsf{out}'}_1$ is part of the final proof and the verifier checks if it encodes zero using the extended decoding key $(1, \mathsf{sk}, \mathsf{sk}^2)$. If so, then with probability $1 - \frac{1}{|\mathbb{A}^*|}$, all pairs $(h', h'') := (H_i'(s), \alpha_{i,h} H_i'(s))$ satisfy that $\alpha_{i,h} h' = h''$. Based on the d-PKE assumption (see Assumption 2.2), the extracted polynomials $H_i(x)$ therefore have degree at most d [6] .

Summary. To summarize, the sub-provers help prepare the inputs of AC. The master prover then utilizes those encodings submitted from the sub-provers to compute the function outputs $\boxed{\mathsf{out}}_1$ and $\boxed{\mathsf{out}'}_1$ of Eq. (4) and Eq. (14), which are embedded in the AC. With the final proof sent from the master prover, the verifier launches three checks. It first verifies that $\boxed{\mathsf{out}}_1$ and $\boxed{\mathsf{out}'}_1$ decode to zero, so that the batched satisfiability check and the batched α-relation check (see $f_{\mathsf{AC}}, f'_{\mathsf{AC}}$ in Definition 4.1) for all sub-circuits are satisfied. Next, the verifier checks that the encoded values of $\left\{\boxed{\bar{V}_{\mathsf{mid}}(\bar{s})}_2, \boxed{\bar{W}_{\mathsf{mid}}(\bar{s})}_2, \boxed{\bar{Y}_{\mathsf{mid}}(\bar{s})}_2\right\}$ align with the linear combination result encoded in $\boxed{L(\bar{s})}_2$ constructed in Eq. (11), which guarantees that the wire value assignment of each sub-circuit, as well as the assignment for $\bar{\mathcal{J}}$ in AC are consistent. This prevents the prover from supplying inconsistent

[6] In our soundness proof, the parameter in the PKE assumption is not exactly d due to some technical subtleties. See Section 7 for details.

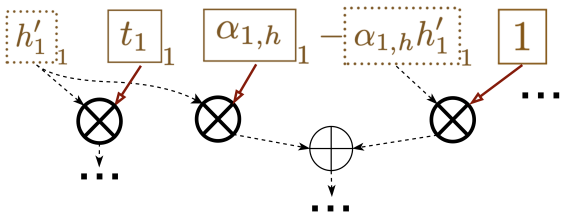

Figure 3: A simple illustration of $\alpha_{1,h}$ check for $[h'_1]_1$. Notice that here the wire carrying $[h'_1]_1$ is shared in the calculation of Eq. (4) (refer also to the AC structure in Fig. 2) and the α-check.

wire values when constructing $\bar{V}(\bar{s}), \bar{W}(\bar{s}), \bar{Y}(\bar{s})$ for the AC. Lastly, the master prover also supplies $[\bar{H}(\bar{s})]_2$ so that the satisfiability check for the AC as in Eq. (5) is verified, which convinces the verifier that the prover side evaluated the AC honestly.

Efficiency. By partitioning the original circuit into m non-interacting sub-circuits, we distribute the work to m sub-provers. Asymptotically, the runtime of each sub-prover is $O(d \log d)$, where d is the maximum degree of the sub-circuits, and the runtime of the master prover is $O(m \log m)$, where m is the total number of sub-provers.

4 Dinocchio

In this section, we formally present our distributed proof protocol for the indexed relation $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathsf{R}_{\mathsf{SAT}}$ for circuit satisfiability. Our main protocol consists of five phases: (1) a setup phase that generates the verification and proving keys, which also specify the structure of all sub-circuits and the aggregation circuit, (2) a distribution phase to generate sub-witnesses based on the partitioned sub-circuits and distribute the proving keys to the prover side, (3) the parallel computation of sub-provers, (4) the aggregation performed by the master prover, which generates a succinct proof, and (5) the verification checks computed by the verifier. An illustration is shown in Fig. 2. Before we describe each phase, we provide the definition of the aggregation circuit, which embeds the batched satisfiability check and α-relation check for all sub-circuits, and the associativity property of a secure encoding scheme needed for the consistency check across all sub-witnesses. In Section 5, we show an encoding scheme that satisfies this associativity property.

The aggregation circuit (AC) serves two tasks. First, it performs the batched satisfiability check, expressed as f_{AC}, of all sub-circuits as illustrated in Eq. (4). The input to the function f_{AC} is denoted as $\{\vec{a}_i\}_{i\in[m]}$. Looking ahead, $\vec{a}_i$ contains the encodings $\Big([v'_{i,\mathsf{io}}]_1, [v'_{i,\mathsf{mid}}]_1, [w'_{i,\mathsf{io}}]_1, [w'_{i,\mathsf{mid}}]_1, [y'_{i,\mathsf{io}}]_1, [y'_{i,\mathsf{mid}}]_1, [h'_i]_1, [t_i]_1\Big)$ taken from the sub-proofs and the proving key. Second, it embeds the batched α-relation check, expressed as f'_{AC}, for the polynomials $H_i(x)$ as shown in Eq. (14). The input to the function f'_{AC} consists of three components, $a_{i,7}, a'_i, a''_i$, where $a_{i,7} := [h'_i]_1$, $a'_i \in \mathcal{R}_q^2$ corresponds to the encoded randomness $[\alpha_{i,h}]_1$, and $a''_i := [h''_i]_1$ [7].

Definition 4.1 (Aggregation Circuit)**.** Let $\Pi_E := (\mathsf{Gen}, \mathsf{E})$ be a secure encoding scheme with encoding space $\mathcal{R}_q^2$ as defined in Definition 2.5. For $m \in \mathbb{Z}$, and $\vec{a}_i \in (\mathcal{R}_q^2)^8, a'_i \in \mathcal{R}_q^2, a''_i \in \mathcal{R}_q^2$ with $i \in [m]$ (i.e., all inputs are encodings in $\mathcal{R}_q^2$), the aggregation circuit (AC) is described by the QRP

[7] Notice that each element is an encoding which is a value in $\mathcal{R}_q^2$.

$C_{\mathsf{AC}} := \big\{\{\bar{V}_{\bar{k}}(x),\ \bar{W}_{\bar{k}}(x), \bar{Y}_{\bar{k}}(x)\}_{\bar{k}\in\bar{\mathcal{I}}}, \bar{t}(x)\big\}$, where $\bar{\mathcal{I}}$ is the index set of all wires in C_{AC}, and evaluates the functions:

$$f_{\mathsf{AC}}\left(\{\vec{a}_i\}_{i\in[m]}\right) = \sum_{i\in[m]} (a_{i,1}+a_{i,2})\otimes(a_{i,3}+a_{i,4}) - (a_{i,5}+a_{i,6})\otimes \mathsf{E}(1) - a_{i,7}\otimes a_{i,8},$$

$$f'_{\mathsf{AC}}\left(\{a_{i,7}, a'_i, a''_i\}_{i\in[m]}\right) = \sum_{i\in[m]} (a_{i,7}\otimes a'_i - a''_i \otimes \mathsf{E}(1)).$$

We denote the output of f_{AC} by $\boxed{\mathsf{out}}_1 \in \mathcal{R}_q^3$ and the output of f'_{AC} by $\boxed{\mathsf{out}'}_1 \in \mathcal{R}_q^3$, which are both extended encodings under Π_E. Moreover, we denote by $\bar{\mathcal{I}}_{\mathsf{io}}$ the set of indices for wires carrying the values $\{a_{i,1}, a_{i,3}, a_{i,5}, a_{i,8}\}_{i\in[m]} \cup \{a'_i\}_{i\in[m]}$, $\bar{\mathcal{L}}$ the set of indices for wires carrying the values of $\{a_{i,2}, a_{i,4}, a_{i,6}\}_{i\in[m]}$, and $\bar{\mathcal{J}} = \bar{\mathcal{I}}\backslash(\bar{\mathcal{I}}_{\mathsf{io}}\cup\bar{\mathcal{L}})$ (see Fig. 1 for an illustration). We define the index-mapping function $\mathsf{Idx}_v : [m] \to \bar{\mathcal{L}}$, which, with input $i \in [m]$, outputs the wire index in AC that carries the value $a_{i,2}$. Similarly, $\mathsf{Idx}_w : [m] \to \bar{\mathcal{L}}$ outputs the wire index in AC that carries the value $a_{i,4}$ and $\mathsf{Idx}_y : [m] \to \bar{\mathcal{L}}$ outputs the wire index in AC that carries the value $a_{i,6}$.

Definition 4.2 (Associative Secure Encoding)**.** A secure encoding scheme $(\mathsf{Gen}, \mathsf{E})$, initialized with two parameter sets, one with plaintext modulus t and encoding modulus q denoted as $\Pi_E^{(1)}$ and another with plaintext modulus q and encoding modulus Q denoted as $\Pi_E^{(2)}$, is *associative* w.r.t. the plaintext-to-encoding multiplication, if for all $a \in \mathcal{R}_t$, for all $\mu_1 \in \mathbb{Z}_t$ and $\mu_2 \in \mathbb{Z}_q$ the following holds: let $\mathsf{ct}_1 := \Pi_E^{(1)}.\mathsf{E}(\mu_1) \in \mathcal{R}_q^2$ and $\mathsf{ct}_2 = \Pi_E^{(2)}.\mathsf{E}(\mu_2) \in \mathcal{R}_Q^2$, and $\left(\boxed{x_1}, \boxed{x_2}\right) := \{\!|a \diamond \mathsf{ct}_1|\!\} \diamond' \mathsf{ct}_2$, $\left(\boxed{y_1}, \boxed{y_2}\right) := a \diamond \left(\{\!|\mathsf{ct}_1|\!\} \diamond' \mathsf{ct}_2\right)$, we have that:

$$\Pr\left[x_1 = y_1 \wedge x_2 = y_2\right] = 1 - \mathsf{negl}(\lambda),$$

i.e., both procedures output two encodings whose underlying plaintexts are the same. As a reminder $\diamond : \mathcal{R}_t \times \mathcal{R}_q^2 \to \mathcal{R}_q^2$ denotes the plaintext-to-encoding multiplication and $\diamond' : \mathcal{R}_q^2 \times \mathcal{R}_Q^2 \to (\mathcal{R}_Q^2)^2$ denotes the "double" plaintext-to-encoding multiplications between two plaintexts and one encoding, i.e., $(a, b) \diamond' \boxed{c} = (a \diamond \boxed{c}, b \diamond \boxed{c})$.

With the formal definition of the aggregation circuit and the associativity property of the underlying encoding scheme in place, we now present our main theorem for Dinocchio.

Theorem 4.1. Let $\mathcal{R}_t := \mathbb{Z}_t[X]/(X^D+1), \mathcal{R}_q := \mathbb{Z}_q[X]/(X^D+1), \mathcal{R}_Q := \mathbb{Z}_Q[X]/(X^{D'}+1)$ be commutative cyclotomic rings, where the ring dimensions D, D' are power-of-two. Denote the exceptional sets that are also unit as $\mathbb{A}_t^* \subset \mathcal{R}_t, \mathbb{A}_q^* \subset \mathcal{R}_q$. Let m be the number of sub-provers, d be an upper bound of the size of sub-circuits, $\mathsf{w} := \sum_{i\in[m]} |\mathcal{I}_{i,\mathsf{mid}}| + |\bar{\mathcal{J}}|$, and $\bar{d}$ be the size of the aggregation circuit, with AC and $\bar{\mathcal{J}}$ as in Definition 4.1. Let $\Pi_E^{(1)}$ and $\Pi_E^{(2)}$ be two instantiations of an associative encoding scheme, where $\Pi_E^{(1)}$ has plaintext modulus $\mathcal{R}_t$ and ciphertext modulus $\mathcal{R}_q$, and $\Pi_E^{(2)}$ has plaintext modulus $\mathcal{R}_q$ and ciphertext modulus $\mathcal{R}_Q$. Assume that the generalized augmented $((3m-1)d+1)$-PKE and the generalized $((2m-1)2d+1)$-PDH hold for $\Pi_E^{(1)}$, and the generalized augmented $(3\bar{d}+2)$-PKE and the generalized $(\mathsf{w}+3\bar{d}+3)$-PDH hold for $\Pi_E^{(2)}$, then the Dinocchio protocol (specified in **Procedure 1-5**) is a distributed SNARK (Definition 2.4) with soundness error $\frac{1}{|\mathbb{A}_t^*|} + \frac{1}{|\mathbb{A}_q^*|}$.

The correctness and the knowledge soundness proofs are deferred to Section 6 and Section 7.

Remark 4.2. To add zero-knowledge to Dinocchio, it suffices for the master prover to adapt the technique for adding zero-knowledge in Rinocchio [27, Figure 2], using the target polynomials $\bar{t}(x)$ of AC to mask out the underlying wire assignments. The encodings $\boxed{\bar{H}(\bar{s})}_2$ (as well as its α-relation pair $\boxed{\alpha_h \bar{H}(\bar{s})}_2$), $\boxed{L(\bar{s})}_2$ in the final proof are then augmented with appropriate masking terms to account for the additional randomness introduced by the zero-knowledge property.

Specifically, the verifier appends $\left\{\boxed{\gamma_\zeta \bar{t}(\bar{s})}_2, \boxed{\beta\gamma_\zeta \bar{t}(\bar{s})}_2\right\}_{\zeta\in\{v,w,y\}}$ to the crs. The master prover samples $\delta_v, \delta_w, \delta_y \in \mathcal{R}$ and computes: $\boxed{\bar{t}'_\zeta(\bar{s})}_2 := \delta_\zeta \diamond \boxed{\gamma_\zeta \bar{t}(\bar{s})}_2$ for $\zeta \in \{w, v, y\}$. Then, instead of $\boxed{\bar{Z}_{\mathsf{mid}}(\bar{s})}_2$, the final proof includes $\boxed{\bar{V}'_{\mathsf{mid}}(\bar{s})}_2 := \boxed{\bar{V}_{\mathsf{mid}}(\bar{s})}_2 + \boxed{\bar{t}'_v(\bar{s})}_2$, $\boxed{\bar{W}'_{\mathsf{mid}}(\bar{s})}_2 := \boxed{\bar{W}_{\mathsf{mid}}(\bar{s})}_2 + \boxed{\bar{t}'_w(\bar{s})}_2$, and $\boxed{\bar{Y}'_{\mathsf{mid}}(\bar{s})}_2 := \boxed{\bar{Y}_{\mathsf{mid}}(\bar{s})}_2 + \boxed{\bar{t}'_y(\bar{s})}_2$. To compensate for the masked randomness in the satisfiability check, the master prover computes:

$$\bar{H}'(x) = \bar{H}(x) + \delta_v \bar{W}(x) + \delta_w \bar{V}(x) + \delta_v \delta_w \bar{t}(x) - \delta_y,$$

where $\bar{Z}(x) := \bar{Z}_{\mathsf{io}}(x) + \bar{Z}_{\mathsf{mid}}(x)$ for $Z \in \{V, W, Y\}$, and then derive $\boxed{\bar{H}'(\bar{s})}_2$ based on $\left\{\boxed{\bar{s}^j}_2\right\}$ and randomness $\gamma_v, \gamma_w, \gamma_y$. Now, during verification, it still holds that $(\bar{V}_{\mathsf{io}}(\bar{s}) + \bar{V}'_{\mathsf{mid}}(\bar{s})) \cdot (\bar{W}_{\mathsf{io}}(\bar{s}) + \bar{W}'_{\mathsf{mid}}(\bar{s})) - (\bar{Y}_{\mathsf{io}}(\bar{s}) + \bar{Y}_{\mathsf{mid}}(\bar{s})) = \bar{H}'(\bar{s})\bar{t}(\bar{s})$.

Similarly, for the consistency check, the master prover computes $\boxed{L'(\bar{s})}_2 := \boxed{L(\bar{s})}_2 + \delta_v \diamond \boxed{\beta\gamma_v \bar{t}(\bar{s})}_2 + \delta_w \diamond \boxed{\beta\gamma_w \bar{t}(\bar{s})}_2 + \delta_y \diamond \boxed{\beta\gamma_y \bar{t}(\bar{s})}_2$ and includes $\boxed{L'(\bar{s})}_2$ to the final proof instead of $\boxed{L(\bar{s})}_2$.

4.1 Partition Functions

Before formally defining our distributed SNARK, we describe how we partition a circuit with its instance and witness into m sub-circuits with the corresponding sub-instances and sub-witnesses. We view an arithmetic circuit as a directed acyclic graph with edges representing the wires and vertices representing the gates.

We provide a simplified version of the partition lemma of Hekaton [56, Lemma 4.1]. At a high level, we split a large circuit into smaller sub-circuits that satisfy the following properties: 1) Given a valid witness to the original large circuit, there is an efficient algorithm that derives the wire assignments for all sub-circuits; 2) Given the wire assignments of all sub-circuits, there is an efficient algorithm that produces a valid assignment for the original circuit with overwhelming probability. Our lemma differs from the one in Hekaton because we do not use a *commitment* to guarantee the consistency among *shared* wires, which is tailored to a commit-carrying SNARK. Instead, we introduce the function f_{map}, which takes as inputs the sub-circuit index and the wire index in that sub-circuit, and returns the wire index in the original circuit. Finally, our knowledge soundness says that if all sub-circuits are satisfied and if all wires that correspond to the same wire in the original circuit carry consistent values, then the original circuit is also satisfied.

Lemma 4.1 (Circuit Partition)**.** For the indexed relation $(\mathbb{i} = (\mathsf{pp}, C), \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$, for some public parameter pp and a circuit C, there exists an efficient transformation $f_{\mathsf{part}} = (f_{\mathbb{i}}, f_{\mathbb{x}}, f_{\mathbb{w}}, f_{\mathsf{map}})$:

$$\begin{aligned}\left(\{\mathbb{i}_i := (\mathsf{pp}, C_i)\}_{i\in[m]}, \mathbb{i}_{\mathsf{AC}}\right) &\leftarrow f_{\mathbb{i}}(\mathbb{i}, m),\\ \{\mathbb{x}_i\}_{i\in[m]} &\leftarrow f_{\mathbb{x}}(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, \mathbb{x}),\\ \{\mathbb{w}_i\}_{i\in[m]} &\leftarrow f_{\mathbb{w}}(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, \mathbb{x}, \mathbb{w}),\end{aligned}$$

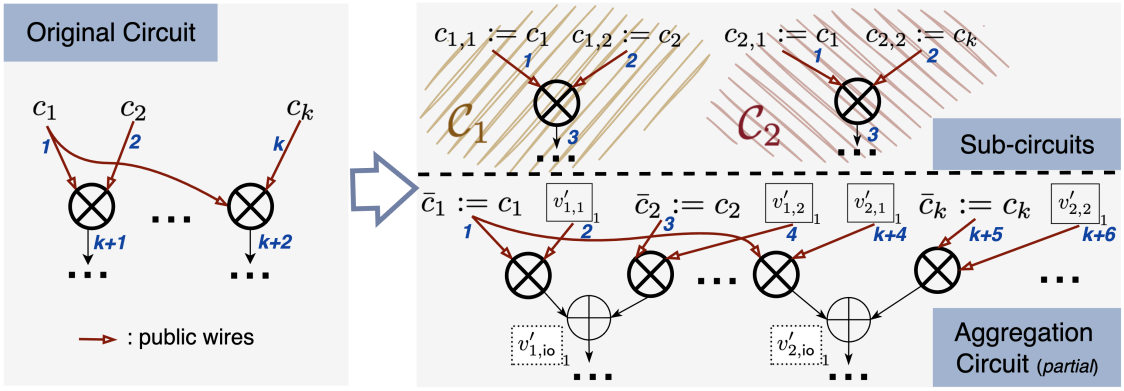

Figure 4: A simple illustration of handling shared public inputs. On the left, the public input c_1 is used in two different multiplication gates, which are then assigned to two sub-circuits C_1, C_2, as depicted in the top-right. As in Section 4.1, the value c_1 is *duplicated* in both sub-circuits. To enforce consistency, the aggregation circuit uses the same value $\bar{c}_1 := c_1$ to reconstruct $\boxed{v'_{1,\mathsf{io}}}_1$ and $\boxed{v'_{2,\mathsf{io}}}_1$. Note that the (partial) depiction of AC here contains only the derivation of $\boxed{v'_{i,\mathsf{io}}}$, which are then used as inputs to f_{AC} (see Definition 4.1 and Fig. 1).

and $k' \leftarrow f_{\mathsf{map}}\left(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, i, k\right)$ maps the wire index $k \in \mathcal{I}_i$ in sub-circuit C_i to a wire index $k' \in \mathcal{I}$ in the original circuit C, such that the following properties hold:

- *Completeness*: If $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$, then for all $i \in [m]$, $(\mathbb{i}_i, \mathbb{x}_i, \mathbb{w}_i) \in \mathrm{R}_{\mathsf{SAT}}$.
- *Knowledge soundness*: Let $\left(\{\mathbb{i}_i\}_{i\in[m]}, \mathbb{i}_{\mathsf{AC}}\right) \leftarrow f_{\mathbb{i}}(\mathbb{i}, m)$, for every PPT algorithm $\mathcal{A}$, $\{(\mathbb{x}_i, \mathbb{w}_i)\}_{i\in[m]} \leftarrow \mathcal{A}(\{\mathbb{i}_i\}_{i\in[m]}, \mathbb{i}_{\mathsf{AC}}, m)$, if for all $i, i' \in [m], k \in \mathcal{I}_i, k' \in \mathcal{I}_{i'}$ s.t. $f_{\mathsf{map}}(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, i, k) = f_{\mathsf{map}}(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, i', k')$, we have $c_k = c'_k$, i.e., the k-th wire in C_i carries the same value as the k'-th wire in $C_{i'}$ if they are stemmed from the same wire in the original circuit C, then there exists a PPT extractor $\mathcal{E}$, $(\mathbb{x}, \mathbb{w}) \leftarrow \mathcal{E}(\mathbb{i}, \{(\mathbb{i}_i, \mathbb{x}_i, \mathbb{w}_i)\}_{i\in[m]})$, such that:

$$\Pr\left[\begin{array}{c} \forall i, (\mathbb{i}_i, \mathbb{x}_i, \mathbb{w}_i) \in \mathrm{R}_{\mathsf{SAT}} \\ \wedge\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathrm{R}_{\mathsf{SAT}} \end{array}\right] = \mathsf{negl}(\lambda).$$

We first show a concrete realization of $f_{\mathsf{part}} = (f_{\mathbb{i}}, f_{\mathbb{x}}, f_{\mathbb{w}}, f_{\mathsf{map}})$ defined in Lemma 4.1 for a data-parallel circuit, i.e., a circuit where there are no shared wires among sub-circuits. At the end of the section, we remark on how to generalize the partition function to any arbitrary circuit.

This partition function exploits the fact that a data-parallel circuit can be divided into multiple *disjoint* components, without sharing wires other than the common public inputs. Therefore, we simply group those components as separate sub-circuits. Then, as long as each sub-circuit is satisfied, and the shared common public inputs are consistent among the sub-circuits (which are enforced by the f_{map} function), we have a valid witness set for the original circuit.

Proof. We now prove Lemma 4.1 for a data-parallel circuit by showing that under our partition function f_{part}, given in Fig. 5 to Fig. 7, the completeness and the knowledge soundness hold.

The completeness is clear to see, since if $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$, then we have each C_i satisfied directly.

Figure 5: Formal algorithm of $f_{\mathbb{i}}$ for the partition function f_{part}.

$\left(\{\mathbb{i}_i := (\mathsf{pp}, C_i)\}_{i\in[m]}, \mathbb{i}_{\mathsf{AC}} := (\mathsf{pp}, C_{\mathsf{AC}})\right) \leftarrow f_{\mathbb{i}}(\mathbb{i}, m)$:

1. Parse $\mathbb{i} := (\mathsf{pp}, C := (\{V_\kappa(x), W_\kappa(x), Y_\kappa(x)\}_{\kappa\in\mathcal{I}_{\mathsf{io}}}, \{V_k(x), W_k(x), Y_k(x)\}_{k\in\mathcal{I}_{\mathsf{mid}}}))$.
2. For the data-parallel circuit C, divide it into m sub-circuits with disjoint wire sets $(\mathcal{W}_{i,\mathsf{io}}, \mathcal{W}_{i,\mathsf{mid}})_{i\in[m]}$, where $\mathcal{W}_{i,\mathsf{io}}$ represents the indices of the public inputs or outputs and $\mathcal{W}_{i,\mathsf{mid}}$ includes the indices of all other wires. Notice that for shared public inputs, we simply "duplicate" them among different sub-circuits. [a] Initialize the QRP of each sub-circuit C_i as $\{Z_{i,\kappa}(x)\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}, \{Z_{i,k}(x)\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}$, for $Z \in \{V, W, Y\}$, with the wire indices re-labelled, i.e., $\mathcal{I}_{i,\mathsf{io}} := [|\mathcal{W}_{i,\mathsf{io}}|]$ and $\mathcal{I}_{i,\mathsf{mid}} := \{|\mathcal{I}_{i,\mathsf{io}}| + 1, \ldots, |\mathcal{I}_{i,\mathsf{io}}| + |\mathcal{W}_{i,\mathsf{mid}}|\}$.
3. For the aggregation circuit, as an initial setup, set $\{\bar{c}_{\bar{\kappa}}\}_{\bar{\kappa}\in\bar{\mathcal{I}}_{\mathsf{io}}} := \{c_\kappa\}_{\kappa\in\mathcal{I}_{\mathsf{io}}} \cup \left\{\boxed{\zeta'_{i,\kappa}}_1\right\}_{i\in[m],\kappa\in\mathcal{I}_{i,\mathsf{io}}}$. For each sub-circuit C_i, for $\kappa \in \mathcal{I}_{i,\mathsf{io}}$, add a multiplication gate taking $\left(\bar{c}_{\bar{\kappa}}, \boxed{\zeta'_{i,\kappa}}_1\right)$ as inputs if $\bar{c}_{\bar{\kappa}} = c_{i,\kappa}$, for $\zeta \in \{v, w, y\}$, where $v'_{i,\kappa} := s^{(i-1)d} V_{i,\kappa}(s)$; similarly add gates for w' and y'. Add an addition gate to sum over $\kappa \in \mathcal{I}_{i,\mathsf{io}}$ for all $i \in [m]$, and denote the output as $\dddot{\zeta}'_{i,\mathsf{io}}$.
   Since the same public input $\bar{c}_{\bar{\kappa}}$ might be used in different multiplication gates with indices $\{g_j, \ldots, g_{j'}\}$, we force them to share the value $\bar{c}_{\bar{\kappa}}$ by setting $\bar{V}_{\bar{\kappa}}(g_j) = \cdots = \bar{V}_{\bar{\kappa}}(g_{j'}) = 1$, i.e., $\bar{c}_{\bar{\kappa}}$ serves as the left input to all such multiplication gates. A simple illustration of this augmentation to AC is shown in Fig. 4.
   Augment the aggregation circuit to evaluate functions $f_{\mathsf{AC}}, f'_{\mathsf{AC}}$ as in Definition 4.1, where the inputs $(\{\vec{a}_i, a'_i, a''_i\})$ are all derived during proof generation, except that $a_{1,i} = \dddot{v}'_{i,\mathsf{io}}$, $a_{3,i} = \dddot{w}'_{i,\mathsf{io}}$ and $a_{5,i} = \dddot{y}'_{i,\mathsf{io}}$. Let the wire index set $\bar{\mathcal{L}}, \bar{\mathcal{J}}$ be as defined in Definition 4.1, where $\bar{\mathcal{I}}_{\mathsf{mid}} := \bar{\mathcal{L}} \cup \bar{\mathcal{J}}$.
4. For $i \in [m]$, output $\mathbb{i}_i := \left(\mathsf{pp}, C_i := \left(\{V_{i,\kappa}(x), W_{i,\kappa}(x), Y_{i,\kappa}(x)\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}, \{V_{i,k}(x), W_{i,k}(x), Y_{i,k}(x)\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}\right)\right)$.
   Output $\mathbb{i}_{\mathsf{AC}} := \left(\mathsf{pp}, C_{\mathsf{AC}} := \left(\left\{\bar{V}_{\bar{\kappa}}(x), \bar{W}_{\bar{\kappa}}(x), \bar{Y}_{\bar{\kappa}}(x)\right\}_{\bar{\kappa}\in\bar{\mathcal{I}}_{\mathsf{io}}}, \left\{\bar{V}_{\bar{k}}(x), \bar{W}_{\bar{k}}(x), \bar{Y}_{\bar{k}}(x)\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}\right)\right)$.

[a] For instance, if $\mathcal{I}_{\mathsf{io}} = \{1, 2, 3\}$ and sub-circuits C_1, C_2 both use the second wire, then $2 \in \mathcal{W}_{1,\mathsf{io}}$ and $2 \in \mathcal{W}_{2,\mathsf{io}}$. See Fig. 4 for a simple illustration of how one shared public input is encoded across different sub-circuits.

$\{\mathbb{x}_i\}_{i\in[m]} \leftarrow f_{\mathbb{x}}\left(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, \mathbb{x}\right)$:

1. Parse $\mathbb{i}_i := \left(\mathsf{pp}, C_i := \left(\{V_{i,\kappa}(x), W_{i,\kappa}(x), Y_{i,\kappa}(x)\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}, \{V_{i,k}(x), W_{i,k}(x), Y_{i,k}(x)\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}\right)\right)$.
   Parse $\mathbb{x} := \{c_\kappa\}_{\kappa\in\mathcal{I}_{\mathsf{io}}}$ to be the wire values of public inputs and outputs of the original circuit C.
2. For each wire with index $\kappa \in \mathcal{I}_{i,\mathsf{io}}$, let $\kappa' \leftarrow f_{\mathsf{map}}\left(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, i, \kappa\right)$ be the wire index in C, and set $c_{i,\kappa} = c_{\kappa'}$.
3. Output $\mathbb{x}_i := \{c_{i,\kappa}\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}$.

Figure 6: Formal algorithm of $f_{\mathbb{x}}$ for the partition function f_{part}.

For the knowledge soundness, we first derive the complete wire assignment $\{c_{i,k}\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}$ based on $\mathbb{w}_i$ for each sub-circuit C_i. Let $\mathcal{I}'_{i,\mathsf{io}}$ denote the index set of wires that carry public inputs also shared with other sub-circuits, and $\mathcal{I}''_{i,\mathsf{io}} := \mathcal{I}_{i,\mathsf{io}} \setminus \mathcal{I}'_{i,\mathsf{io}}$ represents the index set for wires carrying values used only in sub-circuit C_i. We note that there are no overlaps among the $\mathcal{I}_{i,\mathsf{mid}}$ and $\mathcal{I}''_{i,\mathsf{io}}$. Thus, we extract the values of wires with indices in $\mathcal{I}_{\mathsf{mid}} := \bigcup_i \mathcal{I}_{i,\mathsf{mid}}$ and $\mathcal{I}''_{\mathsf{io}} := \bigcup_i \mathcal{I}''_{i,\mathsf{io}}$ as follows: 1) for $k \in \mathcal{I}_{\mathsf{mid}}$, set $c_k := c_{i,k'}$ where $k \leftarrow f_{\mathsf{map}}(i, k')$; and 2) for $\kappa'' \in \mathcal{I}''_{\mathsf{io}}$, set $c_{\kappa''} := c_{i,\kappa}$

$\{\mathbb{w}_i\}_{i\in[m]} \leftarrow f_{\mathbb{w}}\left(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, \mathbb{x}, \mathbb{w}\right)$:

1. Parse $\mathbb{i}_i := \left(\mathsf{pp}, C_i := \left(\{V_{i,\kappa}(x), W_{i,\kappa}(x), Y_{i,\kappa}(x)\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}, \{V_{i,k}(x), W_{i,k}(x), Y_{i,k}(x)\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}\right)\right)$.
   Parse $\mathbb{w} := \{c_k\}_{k\in\mathcal{I}_{\mathsf{mid}^*}}$, where $\mathcal{I}_{\mathsf{mid}^*} \subseteq \mathcal{I}_{\mathsf{mid}}$ denotes the index set of wires carrying witness in the original circuit C, i.e., the wires which carry values in $\mathbb{w}$ which are used to derive the wire assignments of the circuit.
2. Initialize $\mathcal{I}_{i,\mathsf{mid}^*} := \emptyset$. For all wires $c_{i,k}$ with indices $k \in \mathcal{I}_{i,\mathsf{mid}}$, if $k' \leftarrow f_{\mathsf{map}}\left(\mathbb{i}, \{\mathbb{i}_i\}_{i\in[m]}, i, k\right) \in \mathcal{I}_{\mathsf{mid}^*}$, set $c_{i,k} := c_{k'}$, and set $\mathcal{I}_{i,\mathsf{mid}^*} \leftarrow \mathcal{I}_{i,\mathsf{mid}^*} \cup \{k\}$.
3. Output $\mathbb{w}_i := \{c_{i,k}\}_{k\in\mathcal{I}_{i,\mathsf{mid}^*}}$.

Figure 7: Formal algorithm of $f_{\mathbb{w}}$ for the partition function f_{part}.

where $\kappa'' \leftarrow f_{\mathsf{map}}(i,\kappa)$. For the shared public inputs $\kappa' \in \mathcal{I}'_{\mathsf{io}} := \bigcup_i \mathcal{I}'_{i,\mathsf{io}}$, under the constraint that $c_{i,\kappa} = c_{i^*,\kappa^*}$ when $f_{\mathsf{map}}(i,\kappa) = f_{\mathsf{map}}(i^*,\kappa^*) = \kappa'$, we can consistently set $c_{\kappa'} := c_{i,\kappa}$. Hence, $\{c_\kappa\}_{\kappa\in\mathcal{I}_{\mathsf{io}}} := \{c_{\kappa'}\}_{\kappa\in\mathcal{I}'_{\mathsf{io}}} \cup \{c_{\kappa''}\}_{\kappa''\in\mathcal{I}''_{\mathsf{io}}}$. Together, $\{c_\kappa\}_{\kappa\in\mathcal{I}_{\mathsf{io}}} \cup \{c_k\}_{k\in\mathcal{I}_{\mathsf{mid}}}$ forms the valid wire assignment of the original circuit C, from which $\mathbb{w}$ can be directly extracted. □

Remark 4.3. For an arbitrary circuit, one can partition the circuit by following the procedure of Hekaton [56, figure 4]. The only caveat is that Hekaton assumes a commit-carrying SNARK (i.e., Mirage [35]) as the base protocol, and utilizes the commitment to facilitate the consistency proof of shared wires among multiple sub-circuits. Even though Rinocchio is not naturally commit-carrying, the consistency proof of the shared wires is an NP language, which can be represented as an arithmetic circuit. Therefore, we augment each sub-circuit to also express the verification of this consistency check. A naive solution is to hash the shared wires into a Merkle root and require each sub-circuit to verify the corresponding Merkle paths for the wire values. A rough estimation suggests that if approximately 128 wires are shared, augmenting each sub-circuit with Merkle paths incurs an additional cost of about 2^{18} multiplication gates. We leave the design of a more efficient mechanism for handling shared wires to future work.

4.2 Setup Phase

The setup phase is described in **Procedure 1** in Fig. 8. The setup phase takes as inputs the circuit description and some public parameters. In our case, the public parameters include the QRP (Definition 2.2) representation of the original circuit and the number of sub-provers m. First, the setup algorithm invokes $f_{\mathbb{i}}(\mathbb{i}, m)$ given in Section 4.1 to distribute the original circuit description in $\mathbb{i}$ into m sub-circuits $\{C_i\}_{i\in[m]}$ described in $\{\mathbb{i}_i\}_{i\in[m]}$ and computed by the QRPs $\{\{V_{i,k}(x), W_{i,k}(x), Y_{i,k}(x)\}_{k\in\mathcal{I}_i}, t_i(x)\}_{i\in[m]}$ where $\mathcal{I}_i$ is the set of wires of the sub-circuit C_i. Besides the sub-circuits, the setup procedure also generates an aggregation circuit (AC) deterministically based on m, formally defined in Definition 4.1. In our protocol, the master prover uses AC to evaluate the aggregated proof based on the sub-proofs.

We now describe how to produce the proving and verification keys, which depend on the m sub-circuits and the AC circuit. Assume that we have a secure associative encoding scheme, instantiated as $\Pi_E^{(1)} := (\mathsf{Gen}^{(1)}, \mathsf{E}^{(1)}), \Pi_E^{(2)} := (\mathsf{Gen}^2, \mathsf{E}^{(2)})$ as in Definition 4.2, with two different parameter sets. The setup first generates the key pairs $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Gen}^{(1)}(1^\lambda)$ and $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Gen}^{(2)}(1^\lambda)$. Denote by t and q the plaintext moduli of $\Pi_E^{(1)}$ and $\Pi_E^{(2)}$ respectively. Then, as in Rinocchio,

Procedure 1: $(\mathsf{VK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{i}, m)$:

1. Parse $\mathbb{i}$ as $(\mathsf{pp} := (\mathcal{R}, f_{\mathsf{part}}), C := (\{V_k(x), W_k(x), Y_k(x)\}_{k\in\mathcal{I}}, t(x)))$, where $\mathcal{I} := \mathcal{I}_{\mathsf{io}} \cup \mathcal{I}_{\mathsf{mid}}$ is the set of all wires in C as in Definition 2.2, and f_{part} is as in Lemma 4.1. The tuple $(\{V_k(x), W_k(x), Y_k(x)\}, t(x))$ represents the structure of circuit C (see Definition 2.2).

2. $\left(\left\{\mathbb{i}_i := \left(\mathsf{pp}, C_i = \left(\{V_{i,k}, W_{i,k}, Y_{i,k}\}_{k\in\mathcal{I}_i}, t_i(x)\right)\right)\right\}_{i\in[m]}, \mathbb{i}_{\mathsf{AC}} := \left(\mathsf{pp}, C_{\mathsf{AC}} = \left(\{\bar{V}_{\bar{k}}, \bar{W}_{\bar{k}}, \bar{Y}_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{I}}}, \bar{t}(x)\right)\right)\right) \leftarrow f_{\mathbb{i}}(\mathbb{i}, m)$. Let d be the maximum number of multiplication gates in the sub-circuits and let $\bar{d}$ be the number of multiplication gates in AC.

   Let $\bar{\mathcal{I}}_{\mathsf{mid}} := \bar{\mathcal{L}} \cup \bar{\mathcal{J}}$, where $\bar{\mathcal{L}}$ includes all wire indices that are supposed to carry values $\left\{\boxed{Z'_{i,\mathsf{mid}}(s)}_1\right\}$ as in Definition 4.1, and $\bar{\mathcal{J}}$ includes all other non-public wire indices in AC. To simplify our notation, throughout the construction, we use Z and ζ to iterate over $\{V, W, Y\}$ and $\{v, w, y\}$ respectively.

3. Choose t, q such that $|C_i| \mid t-1$ and $|C_{\mathsf{AC}}| \mid q-1$. [a] Initialize the associative encoding scheme $\Pi_E^{(1)} := (\mathsf{Gen}^{(1)}, \mathsf{E}^{(1)})$ with key pair $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Gen}^{(1)}(1^\lambda)$, ciphertext modulus q, and plaintext modulus t, and $\Pi_E^{(2)} := (\mathsf{Gen}^{(2)}, \mathsf{E}^{(2)})$ with key pair $(\mathsf{pk}_2, \mathsf{sk}_2) \leftarrow \mathsf{Gen}^{(2)}(1^\lambda)$, ciphertext modulus Q, and plaintext modulus q.

4. Sample the randomness: $s \xleftarrow{\$} \mathbb{A}_t^*, \bar{s} \xleftarrow{\$} \mathbb{A}_q^*, \gamma_v, \gamma_w \xleftarrow{\$} \mathcal{R}_q^*$, $\gamma_y = \gamma_v \cdot \gamma_w$, $\alpha, \alpha_v, \alpha_w, \alpha_y \xleftarrow{\$} \mathcal{R}_q^*$, $\forall i \in [m], \alpha_{i,h} \xleftarrow{\$} \mathcal{R}_t^*$, $\beta \xleftarrow{\$} \mathcal{R}_q \backslash \{0\}$, $\forall i \in [m], \gamma_{i,v}, \gamma_{i,w} \xleftarrow{\$} \mathcal{R}_t^*$, and $\gamma_{i,y} = \gamma_{i,v} \cdot \gamma_{i,w}$.

5. Construct $\mathsf{crs} :=$

$$\left(\left\{\left\{\boxed{\gamma_{i,y}s^j}_1, \boxed{\alpha_{i,h}\gamma_{i,y}s^j}_1\right\}_{j\in\{(i-1)2d,\ldots,(i-1)2d+d\}}, \boxed{\alpha_{i,h}}_1\right\}_{i\in[m]}, \tag{15}$$

$$\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,\kappa}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,\kappa}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,\kappa}(s)}_1\right\}_{i\in[m],\kappa\in\mathcal{I}_{i,\mathsf{io}}}, \tag{16}$$

$$\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,k}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,k}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,k}(s)}_1\right\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}, \tag{17}$$

$$\left\{\left\{\boxed{\gamma_\zeta \bar{Z}_{\bar{k}}(\bar{s})}_2, \boxed{\alpha_\zeta\gamma_\zeta \bar{Z}_{\bar{k}}(\bar{s})}_2\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}\right\}_{Z\in\{V,W,Y\}}, \tag{18}$$

$$\left\{\boxed{\beta(\gamma_v\bar{V}_{\bar{k}}(\bar{s}) + \gamma_w\bar{W}_{\bar{k}}(\bar{s}) + \gamma_y\bar{Y}_{\bar{k}}(\bar{s}))}_2\right\}_{\bar{k}\in\bar{\mathcal{J}}}, \left\{\boxed{\beta\left(\gamma_v\Psi_{i,k}^V(\bar{s}) + \gamma_w\Psi_{i,k}^W(\bar{s}) + \gamma_y\Psi_{i,k}^Y(\bar{s})\right)}_2\right\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}, \tag{19}$$

$$\left\{\boxed{t_i(s)}_1\right\}_{i\in[m]}, \left\{\boxed{\bar{s}^j}_2, \boxed{\alpha\bar{s}^j}_2\right\}_{j\in\{0,\ldots,\bar{d}\}}\Bigg), \text{ where } \boxed{\Psi_{i,k}^Z(\bar{s})}_2 \text{ is as defined in Eq. (9).} \tag{20}$$

6. Let $\{\bar{c}'_{\bar{\kappa}}\} := \left\{\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,\kappa}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,\kappa}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,\kappa}(s)}_1\right\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}\right\}_{i\in[m]} \cup \left\{\boxed{\alpha_{i,h}}_1\right\}_{i\in[m]}$. Compute $\bar{Z}'_{\mathsf{io}}(\bar{s}) := \sum_{\bar{\kappa}} \bar{c}'_{\bar{\kappa}} \cdot \bar{Z}_{\bar{\kappa}}(\bar{s})$, for $Z \in \{V, W, Y\}$.

7. Set $\mathsf{VK} = \left(\mathsf{sk}_1, \mathsf{sk}_2, s, \bar{s}, \alpha, \alpha_v, \alpha_w, \alpha_y, \gamma_v, \gamma_w, \gamma_y, \beta, \bar{V}'_{\mathsf{io}}(\bar{s}), \bar{W}'_{\mathsf{io}}(\bar{s}), \bar{Y}'_{\mathsf{io}}(\bar{s})\right)$, $\mathsf{PK} = \left(\mathsf{crs}, \mathsf{pp}, \mathsf{pk}_1, \mathsf{pk}_2, \{\mathbb{i}_i\}_{i\in[m]}, \mathbb{i}_{\mathsf{AC}}\right)$. Output $(\mathsf{VK}, \mathsf{PK})$.

[a] This requirement ensures the existence of a $|C_i|$-th root of unity $\{\xi^j\}$ in $\mathcal{R}_t$. We can then compute $H_i(x), \bar{H}(x)$ more efficiently. See Section 8 for details.

Figure 8: The formal algorithm of Setup for Dinocchio.

Procedure 2: $\underline{(\overline{\mathsf{PK}}, \{(\mathsf{PK}_i, \mathbb{x}_i, \mathbb{w}_i)\}_{i\in[m]}) \leftarrow \mathsf{Distribute}(\mathsf{PK}, \mathbb{x}, \mathbb{w}, \mathsf{st})}$:

1. Parse $\mathsf{PK} \rightarrow (\mathsf{crs}, \mathsf{pp}, \mathsf{pk}_1, \mathsf{pk}_2, \{\mathbb{i}_i\}_{i\in[m]}, \mathbb{i}_{\mathsf{AC}})$.
2. Compute $\{\mathbb{x}_i\}_{i\in[m]} \leftarrow f_{\mathbb{x}}\left(\{\mathbb{i}_i\}_{i\in[m]}, \mathbb{x}\right)$ and $\{\mathbb{w}_i\}_{i\in[m]} \leftarrow f_{\mathbb{w}}\left(\{\mathbb{i}_i\}_{i\in[m]}, \mathbb{x}, \mathbb{w}\right)$.
3. Split the public proving keys into:

$$\begin{aligned}\overline{\mathsf{PK}} = \Bigg(&\mathbb{i}_{\mathsf{AC}}, \left\{\boxed{\bar{s}^j}_2, \boxed{\alpha\bar{s}^j}_2\right\}_{j\in\{0,\ldots,\bar{d}\}}, \left\{\boxed{t_i(s)}_1\right\}_{i\in[m]}, \left\{\boxed{\alpha_{i,h}}_1\right\}_{i\in[m]} \\ &\left\{\boxed{\beta(\gamma_v\bar{V}_{\bar{k}}(\bar{s}) + \gamma_w\bar{W}_{\bar{k}}(\bar{s}) + \gamma_y\bar{Y}_{\bar{k}}(\bar{s}))}_2\right\}_{\bar{k}\in\bar{\mathcal{J}}}, \left\{\left\{\boxed{\gamma_\zeta\bar{Z}_{\bar{k}}(\bar{s})}_2, \boxed{\alpha_\zeta\gamma_\zeta\bar{Z}_{\bar{k}}(\bar{s})}_2\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}\right\}_{Z\in\{V,W,Y\}}, \\ &\left\{\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,\kappa}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,\kappa}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,\kappa}(s)}_1\right\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}\right\}_{i\in[m]}\Bigg) \end{aligned} \tag{21}$$

$$\begin{aligned}\{\mathsf{PK}_i\}_{i\in[m]} = \Bigg\{\Bigg(&\mathbb{i}_i, \left\{\boxed{\gamma_{i,y}s^j}_1, \boxed{\alpha_{i,h}\gamma_{i,y}s^j}_1\right\}_{j\in\{(i-1)2d,\ldots,(i-1)2d+d\}}, \\ &\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,k}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,k}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,k}(s)}_1\right\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}, \\ &\left\{\boxed{\beta\left(\gamma_v\Psi^V_{i,k}(\bar{s}) + \gamma_w\Psi^W_{i,k}(\bar{s}) + \gamma_y\Psi^Y_{i,k}(\bar{s})\right)}_2\right\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}\Bigg\}_{i\in[m]}. \end{aligned} \tag{22}$$

4. Output $\left(\overline{\mathsf{PK}}, \{(\mathsf{PK}_i, \mathbb{x}_i, \mathbb{w}_i)\}_{i\in[m]}\right)$.

Figure 9: The formal algorithm of Distribute for Dinocchio.

we sample the random evaluation points $s \xleftarrow{\$} \mathbb{A}^*_t, \bar{s} \xleftarrow{\$} \mathbb{A}^*_q$, and two sets of randomness: 1) $\alpha_v, \alpha_w, \alpha_y, \gamma_v, \gamma_w, \gamma_y \xleftarrow{\$} \mathcal{R}^*, \beta \xleftarrow{\$} \mathcal{R}^* \setminus \{0\}$, which are used to facilitate the consistency checks for each sub-circuit as well as the aggregation circuit; and 2) $\{\alpha_{i,h}, \gamma_{i,v}, \gamma_{i,w}, \gamma_{i,y}\}_{i\in[m]}$, which are used to enforce the α-relation check for polynomials $H_i(x)$.

Lastly, the setup constructs the crs, which provides all encodings to facilitate the proof generation. To be more specific, it generates encodings in Eq. (16) and Eq. (17). Those encodings are used by the sub-provers to construct $\left\{\boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1\right\}$, later treated as the inputs to AC (see Fig. 1). The encodings in Eq. (15) are provided to sub-provers in order to derive $\left\{\boxed{h'_i}_1\right\}$ and $\left\{\boxed{\alpha_{i,h}h'_i}_1\right\}$.

Similar to Rinocchio, it generates the encodings in Eq. (18) and Eq. (19) used for proving the consistency of wire values in the set $\bar{\mathcal{I}}_{\mathsf{mid}} := \bar{\mathcal{L}} \cup \bar{\mathcal{J}}$. Notice that for $\bar{\mathcal{L}}$, the setup generates the encodings of the latter part in Eq. (19) based on the entangled encodings $\boxed{\Psi^Z_{i,k}(\bar{s})}_2$ introduced in Eq. (9), for $Z \in \{V, W, Y\}$, $i \in [m]$, and $k \in \mathcal{I}_{i,\mathsf{mid}}$. These entangled encodings ensure that all inputs in $\bar{\mathcal{L}}$ are derived consistently from the witness of the corresponding sub-circuit. We further discuss their role in Section 4.5. A technicality that we omit in Section 3.2 is that now we require the randomness $\{\gamma_{i,\zeta}\}$ to ensure that the encodings generated in Eq. (16) and Eq. (17) are

Procedure 3: $\pi_i \leftarrow \mathsf{SubProve}(\mathsf{PK}_i, \mathbb{x}_i, \mathbb{w}_i)$:

1. Based on $(\mathbb{x}_i, \mathbb{w}_i)$, evaluate C_i, denote the wire assignment as $\{c_{i,k}\}_{k\in\mathcal{I}_i} = \{c_{i,\kappa}\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}} \cup \{c_{i,k}\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}$.
2. Based on the encodings $\left\{\left(\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,k}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,k}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,k}(s)}_1\right)\right\}_{k\in\mathcal{I}_{i,\mathsf{mid}}}$ parsed from PK_i, compute $\boxed{v'_{i,\mathsf{mid}}}_1 := \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\gamma_{i,v}s^{(i-1)d}V_{i,k}(s)}_1$, $\boxed{w'_{i,\mathsf{mid}}}_1 := \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,k}(s)}_1$, $\boxed{y'_{i,\mathsf{mid}}}_1 := \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,k}(s)}_1$.
3. Derive $H_i(x) := \frac{V_i(x)W_i(x)-Y_i(x)}{t_i(x)}$ based on C_i and $\{c_{i,k}\}_{k\in\mathcal{I}_i}$. Compute $\boxed{h'_i}_1 := \boxed{\gamma_{i,y}s^{(i-1)2d}H_i(s)}_1$ via $\left\{\boxed{\gamma_{i,y}s^j}_1\right\}_{j\in\{(i-1)2d,\ldots,(i-1)2d+d\}}$ provided in PK_i, and similarly compute $\boxed{h''_i}_1$ via $\left\{\boxed{\alpha_{i,h}\gamma_{i,y}s^j}_1\right\}$.
4. Compute $\boxed{l_{i,\bar{\mathcal{L}}}}_2 = \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\beta\left(\gamma_v\Psi^V_{i,k}(\bar{s}) + \gamma_w\Psi^W_{i,k}(\bar{s}) + \gamma_y\Psi^Y_{i,k}(\bar{s})\right)}_2$.
5. Output $\pi_i := \left(\boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1, \boxed{h'_i}_1, \boxed{h''_i}_1, \boxed{l_{i,\bar{\mathcal{L}}}}_2\right)$.

Figure 10: The formal algorithm of SubProve for Dinocchio.

independently distributed. Therefore, in Eq. (9), the entangled encodings $\boxed{\Psi^Z_{i,k}(\bar{s})}_2$ are defined in terms of $V'_{i,k}(s) := \gamma_{i,v}s^{(i-1)d}V_{i,k}(s)$, where the additional randomness $\gamma_{i,v}$ is incorporated (similarly for $W'_{i,k}(s)$ and $Y'_{i,k}(s)$).

To facilitate the proof of the satisfiability check of AC, the setup provides the encodings of Eq. (20). The $\mathcal{P}_\mathcal{M}$ uses these encodings to construct an encoding of $\bar{H}(\bar{s})$.

Finally, the setup phase preprocesses some computation in Step 6 to enable succinct verification. We defer a formal discussion to Section 4.6.

The proving key PK contains the crs, the encoding keys $(\mathsf{pk}_1, \mathsf{pk}_2)$, the description of the sub-circuits and the public parameters, while the verification key VK holds the secret keys $(\mathsf{sk}_1, \mathsf{sk}_2)$, the secret randomness and the pre-processed $\{\bar{Z}'_{\mathsf{io}}(\bar{s})\}$.

4.3 Circuit Distribution

The circuit distribution is described in **Procedure 2** in Fig. 9. Given the public key PK that contains the description of all sub-circuits and AC in $\{\mathbb{i}_i\}_{i\in[m]}$ and $\mathbb{i}_{\mathsf{AC}}$, and the instance-witness tuple $(\mathbb{x}, \mathbb{w})$ of the original circuit, the distribution phase invokes $f_\mathbb{x}$ and $f_\mathbb{w}$ to get $\{\mathbb{x}_i\}_{i\in[m]}, \{\mathbb{w}_i\}_{i\in[m]}$ as defined in Lemma 4.1. The public key PK is split into $\overline{\mathsf{PK}}$ as in Eq. (21) of size $O(m)$, and a list of $\{\mathsf{PK}_i\}_{i\in[m]}$ as in Eq. (22), each of size $O(|C|/m)$. Looking ahead, the master prover uses $\overline{\mathsf{PK}}$ to derive the aggregation proof for AC, and similarly, each sub-prover only processes PK_i when generating the sub-proof instead of reading the whole PK, which is as large as the original circuit.

4.4 Sub-Prover Computation

The sub-prover computation is described in **Procedure 3** in Fig. 10. Each sub-prover $\mathcal{P}_i$, given PK_i, which includes the sub-circuit description $\{\{V_{i,k}, W_{i,k}, Y_{i,k}, H_{i,k}\}_{k\in\mathcal{I}_i}, t_i\}$, and the sub-witness $\mathbb{w}_i$, generates the wire assignment $\{c_{i,k}\}_{k\in\mathcal{I}_i}$. Then, it computes the encodings $\boxed{v'_{i,\mathsf{mid}}}_1 :=$

Procedure 4: $\pi \leftarrow \mathsf{AggProve}(\overline{\mathsf{PK}}, \{\pi_i\}_{i\in[m]})$:

1. Parse $\pi_i \rightarrow \left(\boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1, \boxed{h'_i}_1, \boxed{h''_i}_1, \boxed{l_{i,\bar{\mathcal{L}}}}_2\right)$, for all $i \in [m]$.
2. Based on $\left\{\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,\kappa}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,\kappa}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,\kappa}(s)}_1\right\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}, \boxed{t_i(s)}_1, \boxed{\alpha_{i,h}}_1\right\}_{i\in[m]}$ in the $\overline{\mathsf{PK}}$, and together with $\left\{\boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1, \boxed{h'_i}_1, \boxed{h''_i}_1\right\}_{i\in[m]}$ from $\{\pi_i\}_{i\in[m]}$, evaluate the AC which outputs $\boxed{\mathsf{out}}_1$ that encodes the result of f_{AC} and $\boxed{\mathsf{out}'}_1$ that encodes the result of f'_{AC} as in Definition 4.1.
   Denote by $\bar{\mathcal{I}}_{\mathsf{io}}$ the set of indices for wires carrying the values $\left\{\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,\kappa}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,\kappa}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,\kappa}(s)}_1\right\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}, \boxed{t_i(s)}_1, \boxed{\alpha_{i,h}}_1\right\}_{i\in[m]}$, $\mathbb{x}$, $\boxed{\mathsf{out}}_1$ and $\boxed{\mathsf{out}'}_1$; we use $\{\bar{c}_{\bar{\kappa}}\}_{\bar{\kappa}\in\bar{\mathcal{I}}_{\mathsf{io}}}$ to iterate through the corresponding wire values. Similarly, denote by $\bar{\mathcal{I}}_{\mathsf{mid}} := \bar{\mathcal{I}}\backslash\bar{\mathcal{I}}_{\mathsf{io}}$ the set of indices for all other wires; we use $\{\bar{c}_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}$ to iterate through the corresponding wire values.
3. Derive $\bar{H}(x) := \frac{\bar{V}(x)\bar{W}(x)-\bar{Y}(x)}{t(x)}$ using the set $\{\bar{c}_{\bar{\kappa}}\}_{\bar{\kappa}\in\bar{\mathcal{I}}_{\mathsf{io}}} \cup \{\bar{c}_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}$ from Step 2; then compute $\boxed{\bar{h}}_2 := \boxed{\bar{H}(\bar{s})}_2$ and $\boxed{\bar{h}'}_2 := \boxed{\alpha\bar{H}(\bar{s})}_2$ using $\left\{\boxed{\bar{s}^j}_2\right\}_{j\in\{0,\ldots,\bar{d}\}}$ and $\left\{\boxed{\alpha\bar{s}^j}_2\right\}_{j\in\{0,\ldots,\bar{d}\}}$.
4. Compute $\boxed{\bar{\zeta}'_{\mathsf{mid}}}_2 = \sum_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}} \bar{c}_{\bar{k}} \diamond \boxed{\gamma_\zeta \bar{Z}_{\bar{k}}(\bar{s})}_2$ and $\boxed{\bar{\zeta}''_{\mathsf{mid}}}_2 = \sum_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}} \bar{c}_{\bar{k}} \diamond \boxed{\alpha_\zeta\gamma_\zeta \bar{Z}_{\bar{k}}(\bar{s})}_2$, where $\{\bar{c}_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}$ are derived from Step 2.
5. Compute $\boxed{l_{\bar{\mathcal{L}}}}_2 = \sum_{i\in[m]} \boxed{l_{i,\bar{\mathcal{L}}}}_2$, and $\boxed{l_{\bar{\mathcal{J}}}}_2 = \sum_{\bar{k}\in\bar{\mathcal{J}}} \bar{c}_{\bar{k}} \diamond \boxed{\beta(\gamma_v\bar{V}_{\bar{k}}(\bar{s}) + \gamma_w\bar{W}_{\bar{k}}(\bar{s}) + \gamma_y\bar{Y}_{\bar{k}}(\bar{s}))}_2$. Derive the final linear combination for $\bar{\mathcal{I}}_{\mathsf{mid}}$ as $\boxed{l}_2 = \boxed{l_{\bar{\mathcal{L}}}}_2 + \boxed{l_{\bar{\mathcal{J}}}}_2$.
6. Output $\pi := \left(\boxed{\mathsf{out}}_1, \boxed{\mathsf{out}'}_1, \boxed{\bar{v}_{\mathsf{mid}}}_2, \boxed{\bar{w}'_{\mathsf{mid}}}_2, \boxed{\bar{y}'_{\mathsf{mid}}}_2, \boxed{\bar{v}''_{\mathsf{mid}}}_2, \boxed{\bar{w}''_{\mathsf{mid}}}_2, \boxed{\bar{y}''_{\mathsf{mid}}}_2, \boxed{l}_2, \boxed{\bar{h}}_2, \boxed{\bar{h}'}_2\right)$.

Figure 11: The formal algorithm of AggProve for Dinocchio.

$\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,\mathsf{mid}}(s)}_1$. Similarly, the sub-prover computes $\boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1$ as well as $\boxed{h'_i}_1 := \boxed{\gamma_{i,y}s^{(i-1)2d}H_i(s)}_1$. To help the batched α-relation check which enforces that the polynomials $H_i(x)$ have bounded degree, the sub-provers also derive $\boxed{h''_i}_1 := \boxed{\alpha_{i,h}h'_i}_1$. These steps are similar to the proof generation in Rinocchio [27] and are described in Steps 2 and 3 of **Procedure 3**.

To aid with the proof aggregation, and specifically with proving the consistency of the sub-witness across the encodings $\left\{\boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1\right\}$ computed in the previous steps, $\mathcal{P}_i$ prepares $\boxed{l_{i,\bar{\mathcal{L}}}}_2$ as in Step 4. A high-level explanation for the role of these encodings can be found in the paragraph **Augmented consistency check** in Section 3.2.

In total, 6 encodings are included in each sub-proof, sent to the master prover for aggregation.

Procedure 5: $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{VK}, \mathbb{x}, \pi)$:

1. Parse $\pi \rightarrow \left(\boxed{\mathsf{out}}_1, \boxed{\mathsf{out}'}_1, \boxed{\bar{v}'_{\mathsf{mid}}}_2, \boxed{\bar{w}'_{\mathsf{mid}}}_2, \boxed{\bar{y}'_{\mathsf{mid}}}_2, \boxed{\bar{v}''_{\mathsf{mid}}}_2, \boxed{\bar{w}''_{\mathsf{mid}}}_2, \boxed{\bar{y}''_{\mathsf{mid}}}_2, \boxed{\bar{l}}_2, \boxed{\bar{h}}_2, \boxed{\bar{h}'}_2\right)$, return 0 if parse fails (i.e., the image verification for some of the encodings fails).
2. Define the quadratic program $Q_a(x_1, x_2, \alpha) = x_1 - ax_2$, then using the secret key sk_2:
   (a) for tuples $\left\{\left(\boxed{\bar{\zeta}'_{\mathsf{mid}}}_2, \boxed{\bar{\zeta}''_{\mathsf{mid}}}_2\right)\right\}_{\zeta \in \{v,w,y\}}$, verify that $Q_{\alpha_\zeta}\left(\bar{\zeta}''_{\mathsf{mid}}, \bar{\zeta}'_{\mathsf{mid}}\right) = 0$;
   (b) for tuple $\left(\boxed{\bar{h}}_2, \boxed{\bar{h}'}_2\right)$, verify that $Q_{\alpha_h}\left(\bar{h}'', \bar{h}'\right) = 0$.
3. Define the quadratic program $Q(x_1) = x_1$, then using the extended secret key $(1, \mathsf{sk}_1, \mathsf{sk}_1^2)$ verify that for $\boxed{\mathsf{out}}_1$ and $\boxed{\mathsf{out}'}_1$ it holds that
$$Q(\mathsf{out}) = 0, \tag{23}$$
$$Q(\mathsf{out}') = 0, \tag{24}$$
4. Define the quadratic program $Q_b(x_1, x_2, x_3, x_4) = x_4 - b(x_1 + x_2 + x_3) = 0$, then using the secret key sk': for the encodings $\left(\boxed{\bar{v}'_{\mathsf{mid}}}_2, \boxed{\bar{w}'_{\mathsf{mid}}}_2, \boxed{\bar{y}'_{\mathsf{mid}}}_2, \boxed{\bar{l}}_2\right)$, verify that $Q_\beta\left(\bar{v}'_{\mathsf{mid}}, \bar{w}'_{\mathsf{mid}}, \bar{y}'_{\mathsf{mid}}, l\right) = 0$.
5. Let $\bar{\mathcal{I}}'_{\mathsf{io}}$ denote the index set of wires carrying values in $\{\mathbb{x}\} \cup \left\{\boxed{\mathsf{out}}_1, \boxed{\mathsf{out}'}_1\right\}$, derive $\bar{Z}''_{\mathsf{io}}(\bar{s}) = \sum_{\bar{\kappa} \in \bar{\mathcal{I}}'_{\mathsf{io}}} \bar{c}_{\bar{\kappa}} \bar{Z}_{\bar{\kappa}}(\bar{s})$ for $Z \in \{V, W, Y\}$, and $\bar{\zeta}_{\mathsf{io}} = \bar{Z}'_{\mathsf{io}}(\bar{s}) + \bar{Z}''_{\mathsf{io}}(\bar{s})$ for $\zeta \in \{v, w, y\}$, where $\bar{Z}'_{\mathsf{io}}(\bar{s})$ are parsed from VK. Compute $\bar{t}(\bar{s})$ based on the public structure of AC. With the encodings $\left(\boxed{\bar{v}'_{\mathsf{mid}}}_2, \boxed{\bar{w}'_{\mathsf{mid}}}_2, \boxed{\bar{y}'_{\mathsf{mid}}}_2, \boxed{\bar{h}}_2\right)$ and the secret key sk_2, verify the quadratic equation:
$$\left(\bar{v}_{\mathsf{io}} + \gamma_v^{-1} \cdot \bar{v}'_{\mathsf{mid}}\right) \cdot \left(\bar{w}_{\mathsf{io}} + \gamma_w^{-1} \cdot \bar{w}'_{\mathsf{mid}}\right) - \left(\bar{y}_{\mathsf{io}} + \gamma_y^{-1} \cdot \bar{y}'_{\mathsf{mid}}\right) - \bar{h} \cdot \bar{t}(\bar{s}) = 0. \tag{25}$$
6. If all checks above pass, output 1, and 0 otherwise.

Figure 12: The formal algorithm of Verify for Dinocchio.

4.5 Master Prover Aggregation

The computation of the master prover is described in **Procedure 4** in Fig. 11. In Step 2 of **Procedure 4**, the master prover derives the following evaluations in AC (Definition 4.1):

$$f_{\mathsf{AC}}\left(\{\vec{a}_i\}_{i\in[m]}\right) = \sum_{i\in[m]} \left(\left(\boxed{v'_{i,\mathsf{io}}}_1 + \boxed{v'_{i,\mathsf{mid}}}_1\right) \otimes \left(\boxed{w'_{i,\mathsf{io}}}_1 + \boxed{w'_{i,\mathsf{mid}}}_1\right) - \left(\boxed{y'_{i,\mathsf{io}}}_1 + \boxed{y'_{i,\mathsf{mid}}}_1\right) \otimes \mathsf{E}^{(1)}(1) - \boxed{h'_i}_1 \otimes \boxed{t_i}_1\right) = \boxed{\mathsf{out}}_1, \tag{26}$$

$$f'_{\mathsf{AC}}\left(\left\{a_{i,7}, a'_i, a''_i\right\}_{i\in[m]}\right) = \sum_{i\in[m]} \left(\boxed{h'_i}_1 \otimes \boxed{\alpha_{i,h}}_1 - \boxed{h''_i}_1 \otimes \mathsf{E}^{(1)}(1)\right) = \boxed{\mathsf{out}'}_1, \tag{27}$$

where $\vec{a}_i := \left\{\boxed{v'_{i,\mathsf{io}}}_1, \boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{io}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{io}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1, \boxed{h'_i}_1, \boxed{t_i}_1\right\}$, $a'_i := \boxed{\alpha_{i,h}}_1$, and $a''_i := \boxed{h''_i}_1$. The encodings $\boxed{\mathsf{out}}_1$ and $\boxed{\mathsf{out}'}_1$ are sent to the verifier.

In Step 3, the master prover $\mathcal{P}_{\mathcal{M}}$ uses the set of wire values $\{\bar{c}_{\bar{\kappa}}\}_{\bar{\kappa}\in\bar{\mathcal{I}}_{\mathsf{io}}} \cup \{\bar{c}_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}$ constructed in Step 2 to compute $\left(\boxed{\bar{h}}_2, \boxed{\bar{h}'}_2\right)$. In Step 4, $\mathcal{P}_{\mathcal{M}}$ derives the encodings $\boxed{\bar{\zeta}'_{\mathsf{mid}}}_2$ and $\boxed{\bar{\zeta}''_{\mathsf{mid}}}_2$ for

$\zeta \in \{v, w, y\}$ via $\{\bar{c}_{\bar{k}}\}_{\bar{k} \in \bar{\mathcal{I}}_{\mathsf{mid}}}$. In Step 5, it also derives $\boxed{\bar{l}}_2 := \boxed{\bar{l}_{\bar{\mathcal{L}}}}_2 + \boxed{\bar{l}_{\bar{\mathcal{J}}}}_2$ to facilitate the consistency check of wire values $\{c_{i,k}\}_{i \in [m], k \in \mathcal{I}_{i,\mathsf{mid}}}$ and $\{\bar{c}_{\bar{k}}\}_{\bar{k} \in \bar{\mathcal{J}}}$ across the polynomials $\{\bar{V}_{\bar{\mathcal{L}}}(x), \bar{W}_{\bar{\mathcal{L}}}(x), \bar{Y}_{\bar{\mathcal{L}}}(x)\}$ and $\{\bar{V}_{\bar{\mathcal{J}}}(x), \bar{W}_{\bar{\mathcal{J}}}(x), \bar{Y}_{\bar{\mathcal{J}}}(x)\}$ respectively. As discussed in Section 3.2, $\boxed{\bar{l}_{\bar{\mathcal{L}}}}_2$ is constructed using the encodings in Eq. (19) derived from the entangled encodings $\boxed{\Psi^Z_{i,k}(\bar{s})}_2$. This check relies on the associativity of plaintext-to-encoding multiplication of the encoding scheme (Definition 4.2), so that the following holds:

$$
\begin{aligned}
\boxed{\bar{Z}_{\bar{\mathcal{L}}}(\bar{s})}_2 &= \sum_{i \in [m]} \Bigg(\Big\{ \textstyle\sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{v'_{i,k}} \Big\} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 + \Big\{ \textstyle\sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{w'_{i,k}} \Big\} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 \\
&\quad + \Big\{ \textstyle\sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{y'_{i,k}} \Big\} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_y(i)}(\bar{s})}_2 \Bigg) \\
&= \sum_{i \in [m]} \Bigg(\sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \Big(\{ \boxed{v'_{i,k}} \} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 \Big) + \sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \Big(\{ \boxed{w'_{i,k}} \} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 \Big) \\
&\quad + \sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \Big(\{ \boxed{y'_{i,k}} \} \diamond \boxed{\bar{Z}_{\mathsf{Idx}_y(i)}(\bar{s})}_2 \Big) \Bigg) \\
&= \sum_{i \in [m]} \sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\Psi^Z_{i,k}(\bar{s})}_2 .
\end{aligned}
$$

In total, the succinct proof includes 11 encodings, which is roughly of the same size as the original proof size of Rinocchio, with two extra terms, $\boxed{\mathsf{out}}_1$ and $\boxed{\mathsf{out}'}_1$, required to verify the evaluation results of $f_{\mathsf{AC}}, f'_{\mathsf{AC}}$.

4.6 Verification

The verification checks are described in **Procedure 5** in Fig. 12. The verification checks consist of quadratic root detections as in Definition 2.5, where the verifier uses the secret key sk and encodings $\boxed{a_1}, \ldots, \boxed{a_k}$ to check that $Q(a_1, \ldots, a_k) = 0$ for a quadratic function Q. In Step 1, the verifier parses the proof into 11 encodings. Then, in Step 2, the verifier checks that $\bar{\zeta}''_{\mathsf{mid}} = \alpha_\zeta \bar{\zeta}_{\mathsf{mid}}$ for $\zeta \in \{v, w, y\}$, and $\bar{h}'' = \alpha_h \bar{h}'$; these checks are similar to Rinocchio.

Crucially, the verifier also performs the following three steps, which are different than Rinocchio. In Step 3, the verifier checks that $\boxed{\mathsf{out}}_1$ and $\boxed{\mathsf{out}'}_1$ are both encodings of zero under the extended secret key $(1, \mathsf{sk}_1, \mathsf{sk}_1^2)$. This step ensures two things: On one hand, we have that $f_{\mathsf{AC}} = \sum_i s^{(i-1)2d}(V_i(s)W_i(s) - Y_i(s) - H_i(s)t_i(s)) = 0$. On the other hand, $f'_{\mathsf{AC}} = \sum_i (h' \cdot \alpha_{i,h} - h'') = 0$. Overall, this step convinces the verifier that the satisfiability check passes for all sub-circuits. In Step 4, the verifier verifies that $l = \beta(\bar{v}'_{\mathsf{mid}} + \bar{w}'_{\mathsf{mid}} + \bar{y}'_{\mathsf{mid}})$ and thus ensures the consistency of the wire values $\{c_{i,k}\}_{i \in [m], k \in \mathcal{I}_{i,\mathsf{mid}}}$ and $\{\bar{c}_{\bar{k}}\}_{\bar{k} \in \bar{\mathcal{J}}}$ used to compute $\{\bar{\zeta}'_{\mathsf{mid}}\}$ for the set $\bar{\mathcal{I}}_{\mathsf{mid}}$. The consistency among $\{c_{i,k}\}$ then implies the consistency of $\{\bar{c}_{\bar{k}}\}_{\bar{k} \in \bar{\mathcal{L}}} := \left\{ \sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\zeta'_{i,k}(s)}_1 \right\}_{i \in [m], \zeta \in \{v,w,y\}}$.

Finally, the verifier performs the satisfiability check as shown in Eq. (25) by first computing $\bar{Z}_{\mathsf{io}}(\bar{s}) = \bar{Z}'_{\mathsf{io}}(\bar{s}) + \bar{Z}''_{\mathsf{io}}(\bar{s}), Z \in \{V, W, Y\}$. Specifically, the term $\bar{Z}'_{\mathsf{io}}(\bar{s})$ is derived based on the public in-

puts $\left\{\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,\kappa}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,\kappa}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,\kappa}(s)}_1\right\}_{\kappa\in\mathcal{I}_{i,\mathsf{io}}}\right\}_{i\in[m]} \cup \left\{\boxed{\alpha_{i,h}}_1\right\}_{i\in[m]}$.

As shown in Step 6 in **Procedure 1**, the term $\bar{Z}'_{\mathsf{io}}(\bar{s})$, which requires $O(m)$ computational time, is pre-computed during the setup phase. Meanwhile, $\bar{Z}''_{\mathsf{io}}(\bar{s})$ only depends on $\{\mathbb{x}\}$ and is derived in Step 5 during the verification phase. Therefore, the runtime of the verification phase is just $O(|\mathbb{x}|)$, independent of m.

5 Secure Encoding Scheme

In this section, we describe the encoding scheme used in Dinocchio and prove that this scheme is a secure associative encoding.

RLWE-based encryption scheme. Our encoding scheme is based on the LPR RLWE-based encryption scheme [50], which consists of the following algorithms.

- $\mathsf{pp} = (D, t, q, \sigma, \mathcal{D}, h) \leftarrow \mathsf{GenParam}(1^\lambda)$: Let $\mathcal{R} := \mathbb{Z}[X]/(X^D + 1)$ and $\mathcal{D}$ be a distribution that samples a random ring element with ternary coefficients and a fixed Hamming weight h, such that $\mathsf{RLWE}_{n,q,\mathcal{D},\sigma}$ holds. Set the ciphertext modulus q, the plaintext modulus t, and the standard deviation σ for the noise generation.

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$: Draw a secret key $s \leftarrow \mathcal{D}$. Sample $\alpha \xleftarrow{\$} \mathcal{R}_q$ and noise $x \leftarrow \chi_\sigma \in \mathcal{R}$, and set $\mathsf{pk} := (\alpha, \beta) = (\alpha, \alpha s + x) \in \mathcal{R}_q^2$, $\mathsf{sk} := s$.

- $\mathsf{ct} := (a, b) \leftarrow \mathsf{E}(\mathsf{pp}, \mathsf{pk}, m)$: To encrypt the message $m \in \mathcal{R}_t$, draw $e \leftarrow \mathcal{D} \in \mathcal{R}_q$ and $x', x'' \leftarrow \chi_\sigma$. Construct the ciphertext $(a, b) = (\alpha e + x', \beta e + \Delta m + x'') \in \mathcal{R}_q^2$, where $\Delta := \frac{q}{t}$ is the scaling factor.

- $m \leftarrow \mathsf{D}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct} = (a, b))$: Decrypt the ciphertext to the message $m' = \left\lfloor \frac{t}{q} \cdot (b - a \cdot \mathsf{sk}) \right\rceil$.

RLWE-based encoding scheme. The encoding scheme $\Pi_\mathsf{E} = (\Pi_\mathsf{E}.\mathsf{Gen}, \Pi_\mathsf{E}.\mathsf{E})$ is defined based on the RLWE-based encryption scheme $(\mathsf{GenParam}, \mathsf{KeyGen}, \mathsf{E})$ as follows:

- $(\mathsf{ek}, \mathsf{sk}) \xleftarrow{\$} \Pi_\mathsf{E}.\mathsf{Gen}(1^\lambda)$: invoke $\mathsf{pp} := (D, t, q, \sigma, \mathcal{D}, h) \xleftarrow{\$} \mathsf{GenParam}(1^\lambda)$ of the encryption scheme, and generate the key pair: $(\mathsf{ek}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pp})$;

- $\Pi_\mathsf{E}.\mathsf{E} = \mathsf{E}$.

The addition operation between two encodings follows the standard semantic definition of ring addition, i.e., $\mathsf{ct}_1 + \mathsf{ct}_2 = (c_{0,0}, c_{0,1}) + (c_{1,0}, c_{1,1}) = (c_{0,0} + c_{1,0}, c_{0,1} + c_{1,1})$. We additionally define the "$\diamond$" operation, which is the plaintext-to-encoding multiplication $\mathsf{ct}' \leftarrow a \diamond \mathsf{ct}$ for the encoding scheme as follows: For $a \in \mathcal{R}_t$ and $\mathsf{ct} := (c_0, c_1) \in \mathcal{R}_q^2$, we first lift a into $\mathcal{R}_q$ as $a' \in \mathcal{R}_q$. We denote this lift operation as $[\cdot]_q$. I.e., for $a = \sum_{i\in[n]} a_i \cdot X^i \in \mathcal{R}_t$, let $[a]_q = \sum_{i\in[n]} a'_i \cdot X^i \in \mathcal{R}_q$ s.t. $a_i = a'_i$. Thus, $a \diamond \mathsf{ct} := ([a]_q \cdot c_0, [a]_q \cdot c_1)$ where $\cdot$ is the ring multiplication.

We first prove that the above scheme is an encoding scheme.

Lemma 5.1. Under the $\mathsf{RLWE}_{n,q,\mathcal{D},\chi}$ assumption (Definition 2.1), the d-PKE assumption (Assumption 2.2) and the d'-PDH assumption (Assumption 2.1) the above encoding scheme Π_E parameterized

by $(n, t, q, \sigma, \mathcal{D}, h)$ is a secure encoding scheme (Definition 2.5) which is ℓ-linearly homomorphic with $n = \mathsf{poly}(\lambda)$ and $1 - \mathsf{erf}\left(\frac{q/2t-1}{\ell t\sqrt{2n(2h+1)}\sigma}\right) \leq 2^{-\lambda}$, where λ is the security parameter.

Proof. We first show that the encoding scheme satisfies the following properties:

- *ℓ-linearly homomorphic*: We show that if ℓ satisfies:

$$1 - \mathsf{erf}\left(\frac{q/2t-1}{\ell t\sqrt{2n(2h+1)}\sigma}\right) \leq 2^{-\lambda},$$

  then $\sum_{i\in[\ell]} a_i \diamond \mathsf{ct}_i = \Pi_{\mathsf{E}}.\mathsf{E}(\sum_{i\in[\ell]} a_i m_i)$ where $a_i \in \mathcal{R}_t$ and $\mathsf{ct}_i \in \mathcal{R}_q^2$.

  Notice that the encodings can be viewed as encryptions. In particular, if $\mathsf{ct} := (c_0, c_1)$ is an encoding of m, then $c_0 = \alpha e + x'$ and $c_1 = \beta + \Delta m + x''$, and the decryption noise is $e' := x \cdot e + x'' - s \cdot x'$, where $s, e \leftarrow \mathcal{D}, x, x', x'' \leftarrow \chi_\sigma$. Since the Hamming weight of e and s is h, this noise term e' is the sum of $(2h+1)$ independently sampled noises from discrete Gaussian with parameter σ, and thus the standard derivation of e' is simply $\sqrt{(2h+1)} \cdot \sigma$. Moreover, in each plaintext-to-encoding multiplication, the decryption noise is blown up by a factor bounded by $t\sqrt{n}$, where t is the plaintext modulus and n is the underlying ring dimension. Hence, when we have the summation of ℓ such multiplications, the final decryption noise is drawn from the Gaussian distribution with standard deviation $\ell t\sqrt{n(2h+1)}\sigma$.

  To have an overwhelming correct decryption probability w.r.t. the security parameter λ, the decryption noise e' for $\sum_{i\in[\ell]} a_i \diamond \mathsf{ct}_i$ must be smaller than $\frac{\Delta}{2}$, where $\Delta := q/t$ is the scaling factor. I.e., $\Pr[\Delta/2 > |e'|] \geq 1 - 2^{-\lambda}$. Since the tail of a discrete Gaussian is upper-bounded by that of the corresponding continuous Gaussian, up to a constant shift, it suffices to require $1 - \mathsf{erf}\left(\frac{q/2t-1}{\ell t\sqrt{2n(2h+1)}\sigma}\right) \leq 2^{-\lambda}$ based on the two-sided tail bound.

- *Quadratic root detection*: Given the secret key sk, quadratic root detection is implemented by running the decryption algorithm for each encoding c_i to recover the message a_i, and checking if $Q(a_1, \ldots, a_k) = 0$.

- *Image verification*: We view the encoding $\mathsf{ct} := (c_0, c_1)$ as an encryption which has decryption noise $\left|\frac{t}{q} \cdot (c_1 - c_0 \cdot \mathsf{sk}) - \left\lfloor \frac{t}{q} \cdot (c_1 - c_0 \cdot \mathsf{sk}) \right\rceil\right| \in \mathcal{R}_q$. If the decryption noise has infinity norm less than $\left\lfloor \frac{q}{2t} \right\rceil$, then ct is a valid encoding. As shown by the ℓ-linear homomorphic property, all valid encodings have the noise at most $q/2t$ for each coefficient. Since an encoding has $n = \mathsf{poly}(\lambda)$ such coefficients, the probability that a malformed encoding passes the image verification is negligible w.r.t. λ.

Moreover, as assumed in Rinocchio [27], and in other prior works [9], our encoding scheme satisfies both the q-PDH and the q-PKE assumptions defined in Assumption 2.1 and Assumption 2.2. So, our encoding scheme is a secure encoding. □

Below we present a lemma showing that this encoding is associative as in Definition 4.2 under appropriate parameter choices. At a high level, both evaluations perform the same ring multiplications, but differ in how ring elements under a smaller modulus are lifted to a larger modulus. This lifting introduces additional noise during decryption, which rounds to zero as long as the ciphertext modulus of $\Pi_E^{(2)}$ is sufficiently large to tolerate this noise.

Lemma 5.2. Assume that the encoding scheme $(\Pi_{\mathsf{E}}.\mathsf{Gen}, \Pi_{\mathsf{E}}.\mathsf{E})$ with plaintext-to-encoding operation "$\diamond$" as defined above is initialized with two different parameter sets, one with plaintext modulus t and encoding modulus q, denoted as $\Pi_{\mathsf{E}}^{(1)}$, and the other with plaintext modulus q and encoding modulus Q, denoted as $\Pi_{\mathsf{E}}^{(2)}$. If $t \mid q, q \mid Q$ and $Q \gg tq^2\sqrt{2\lambda}\sigma$, where σ is the standard deviation of error distribution and λ is the security parameter, then Π_{E} is an associative encoding (Definition 4.2).

Proof. Let $\circ : \mathcal{R}^2 \times \mathcal{R}^2 \to \mathcal{R}^4$ denote the tensor product between tuples of ring elements under the same modulus (i.e., if $(a_0, a_1), (b_0, b_1) \in \mathcal{R}_t^2$, $(a_0, a_1) \circ (b_0, b_1) \to (a_0 \cdot b_0, a_0 \cdot b_1, a_1 \cdot b_0, a_1 \cdot b_1)$). Similarly, we have that $\diamond' : \mathcal{R}^2 \times \mathcal{R}^2 \to (\mathcal{R}^2)^2$ denotes the tensor product between elements under different moduli (i.e., if $a \in \mathcal{R}_t^2, b \in \mathcal{R}_q^2$, $a \diamond' b := [a]_q \circ b$.

Observe that for $a \in \mathcal{R}_t$, $\mathsf{ct}_1 \in \mathcal{R}_q^2$, and $\mathsf{ct}_2 \in \mathcal{R}_Q^2$:

$$\begin{aligned}
\left\lfloor a \diamond \mathsf{ct}_1 \right\rceil \diamond' \mathsf{ct}_2 &= \left\lfloor ([a]_q \cdot \mathsf{ct}_1) \mod q \right\rceil \diamond' \mathsf{ct}_2 \mod Q \\
&= [[a]_q \cdot \mathsf{ct}_1 - qK]_Q \circ \mathsf{ct}_2 \mod Q \\
&= \left([[a]_q \cdot \mathsf{ct}_1]_Q \circ \mathsf{ct}_2 - [qK]_Q \circ \mathsf{ct}_2 \right) \mod Q \\
&= \left(\boxed{x_1}, \boxed{x_2} \right) \in (\mathcal{R}_Q^2)^2
\end{aligned}$$

where $K \in \mathcal{R}_q^2$, and

$$\begin{aligned}
a \diamond \left(\left\lfloor \mathsf{ct}_1 \right\rceil \diamond' \mathsf{ct}_2 \right) &= a \diamond (([\mathsf{ct}_1]_Q \circ \mathsf{ct}_1) \mod Q) \mod Q \\
&= [a]_Q \cdot ([\mathsf{ct}_1]_Q \circ \mathsf{ct}_2 - QK') \mod Q \\
&= ([a]_Q \cdot [\mathsf{ct}_1]_Q \circ \mathsf{ct}_2 - [a]_Q \cdot QK') \mod Q \\
&= [a]_Q \cdot [\mathsf{ct}_1]_Q \circ \mathsf{ct}_2 \mod Q \\
&= \left(\boxed{y_1}, \boxed{y_2} \right) \in (\mathcal{R}_Q^2)^2
\end{aligned}$$

where $K' \in (\mathcal{R}_Q^2)^2$.

Now, it remains to show that the encoded messages in $\left(\boxed{x_1}, \boxed{x_2} \right)$ and $\left(\boxed{y_1}, \boxed{y_2} \right)$ are the same. Let $\mathsf{ct}_1 := (c_{1,0}, c_{1,1})$ and $\mathsf{ct}_2 := (c_{2,0}, c_{2,1})$, then expanding $\boxed{x_1}$ and $\boxed{y_1}$ we have that:

$$\boxed{x_1} = \left([[a]_q \cdot c_{1,0}]_Q \cdot c_{2,0} - [qK]_Q \cdot c_{2,0} \bmod Q, [[a]_q \cdot c_{1,0}]_Q \cdot c_{2,1} - [qK]_Q \cdot c_{2,1} \bmod Q \right) \tag{28}$$

$$\boxed{y_1} = \left([a]_Q \cdot [c_{1,0}]_Q \cdot c_{2,0} \bmod Q, [a]_Q \cdot [c_{1,0}]_Q \cdot c_{2,1} \bmod Q \right). \tag{29}$$

Since our encoding scheme is additively homomorphic, $\boxed{x_1}$ can be written as a sum of two encodings:

$$\begin{aligned}
\boxed{x_1'} &= \left([[a]_q \cdot c_{1,0}]_Q \cdot c_{2,0} \bmod Q, [[a]_q \cdot c_{1,0}]_Q \cdot c_{2,1} \bmod Q \right) \\
\boxed{x_1''} &= \left(-[qK]_Q \cdot c_{2,0} \bmod Q, -[qK]_Q \cdot c_{2,1} \bmod Q \right).
\end{aligned}$$

We observe that $[a]_q \cdot \mathsf{ct}_1 < Q$ since $Q > q^2$ and $[a]_Q \cdot [c_{1,0}]_Q < Q$ since $Q > t \cdot q$. This implies that there is no wrap-around modulo Q, and hence $[[a]_q \cdot c_{1,0}]_Q = [a]_Q \cdot [c_{1,0}]_Q \bmod Q$. So, $\boxed{x_1'} = \boxed{y_1}$.

Now, it suffices to show that $\boxed{x_1''}$ is an encoding of zero. Since our encoding scheme is based on an encryption scheme, we equivalently show that the ciphertext $\boxed{x_1''}$ decrypts to zero. In particular,

recall that $\mathsf{ct}_2 := (c_{2,0}, c_{2,1}) = (c_{2,0}, c_{2,0}\mathsf{sk}' + e' + \Delta'\mu_2)$, where $\Delta' = \frac{Q}{q}$ is the scaling factor of $\Pi_{\mathsf{E}}^{(2)}$ and sk' is the underlying secret key. To compute the decryption of $\boxed{x_1''}$, we first compute $\mathsf{sk}' \cdot (-[qK]_Q c_{2,0}) - (-[qK]_Q c_{2,1}) \mod Q$, and then multiply it by $\frac{q}{Q}$ and round to scale the result down to the plaintext space of $\Pi_{\mathsf{E}}^{(2)}$. Therefore,

$$\begin{aligned}\mathsf{sk}' \cdot (-[qK]_Q c_{2,0}) - (-[qK]_Q c_{2,1}) &= -\mathsf{sk}' \cdot [qK]_Q c_{2,0} + [qK]_Q(c_{2,0}\mathsf{sk}' + e' + \Delta'\mu_2)) \\ &= [qK]_Q(e' + \Delta'\mu_2) \bmod Q.\end{aligned}$$

Since $K \leq t < q$ and $Q \gg q^2$, it holds that $[qK]_Q = [q]_Q[K]_Q$. Hence, using the fact that $q \mid Q$, we have that $[qK]_Q\Delta'\mu_2 = [q]_Q[K]_Q\frac{Q}{q}\mu_2 = [K]_Q Q\mu_2 = 0 \bmod Q$, and thus $[qK]_Q(e' + \Delta'\mu_2) = [qK]_Q e' \bmod Q$. Finally, rounding $\frac{q}{Q}[qK]_Q e'$ equals to zero since $Q \gg tq^2\sqrt{2\lambda}\sigma$ and e' has norm bounded by $\sqrt{2\lambda}\sigma$ with overwhelming probability, where σ is the standard derivation of the Gaussian distribution from which each error element is sampled.

Similarly, we argue that $\boxed{x_2}$ and $\boxed{y_2}$ encode the same message.

□

6 Completeness

Under the completeness of circuit partition (Lemma 4.1), as long as $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$, each sub-prover $\mathcal{P}_i$, given $(\mathbb{i}_i, \mathbb{x}_i, \mathbb{w}_i) \in \mathrm{R}_{\mathsf{SAT}}$, can derive a valid wire assignment $\{c_{i,k}\}$ of sub-circuit C_i s.t., $V_i(s) \cdot W_i(s) - Y_i(s) - H_i(s) \cdot t_i(s) = 0$.

Thus, for an honest prover, we observe that the sub-proof π_i computed based on $\{c_{i,k}\}$ contains they following elements:

- $\boxed{v'_{i,\mathsf{mid}}}_1 := \sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\gamma_{i,v} s^{(i-1)d} V_{i,k}(s)}_1$. The encodings $\boxed{w'_{i,\mathsf{mid}}}_1$ and $\boxed{y'_{i,\mathsf{mid}}}_1$ are computed similarly.
- $\boxed{h'}_1 := \boxed{\gamma_{i,y} s^{(i-1)2d} \frac{V_i(s)W_i(s) - Y_i(s)}{t_i(s)}}_1$, and correspondingly $\boxed{h''}_1 := \boxed{\alpha_{i,h} h'}_1$.
- $\boxed{l_{i,\bar{\mathcal{L}}}}_2 := \sum_{k \in \mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\beta(\gamma_v \Psi_{i,k}^V(\bar{s}) + \gamma_w \Psi_{i,k}^W(\bar{s}) + \gamma_y \Psi_{i,k}^Y(\bar{s}))}_2$, where $\Psi_{i,k}^Z(x) := \boxed{\gamma_{i,v} s^{(i-1)d} V_{i,k}(s)} \cdot \bar{Z}_{\mathsf{Idx}_v(i)}(x) + \boxed{\gamma_{i,w} s^{(i-1)d} W_{i,k}(s)} \cdot \bar{Z}_{\mathsf{Idx}_w(i)}(x) + \boxed{\gamma_{i,y} s^{(i-1)2d} Y_{i,k}(s)} \cdot \bar{Z}_{\mathsf{Idx}_y(i)}(x)$.

Thus, by executing $\mathsf{AggProve}$ depicted in **Procedure 4**, the final proof contains the following:

- $\boxed{\bar{\zeta}'_{\mathsf{mid}}}_2 = \sum_{i \in [m]} \Big(\boxed{v'_{i,\mathsf{mid}}} \diamond \boxed{\gamma_\zeta \bar{Z}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 + \boxed{w'_{i,\mathsf{mid}}} \diamond \boxed{\gamma_\zeta \bar{Z}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 + \boxed{y'_{i,\mathsf{mid}}} \diamond \boxed{\gamma_\zeta \bar{Z}_{\mathsf{Idx}_y(i)}(\bar{s})}_2 \Big) + \sum_{\bar{k} \in \bar{\mathcal{J}}} \bar{c}_{\bar{k}} \diamond \boxed{\gamma_\zeta \bar{Z}_{\bar{k}}(\bar{s})}$, for $\zeta \in \{v, w, y\}, Z \in \{V, W, Y\}$;
- $\boxed{\bar{\zeta}''_{\bar{\mathcal{L}}}}_2 = \sum_{i \in [m]} \Big(\boxed{v'_{i,\mathsf{mid}}} \diamond \boxed{\alpha_\zeta \gamma_\zeta \bar{Z}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 + \boxed{w'_{i,\mathsf{mid}}} \diamond \boxed{\alpha_\zeta \gamma_\zeta \bar{Z}_{\mathsf{Idx}_w(i)}(\bar{s})}_2 + \boxed{y'_{i,\mathsf{mid}}} \diamond \boxed{\alpha_\zeta \gamma_\zeta \bar{Z}_{\mathsf{Idx}_v(i)}(\bar{s})}_2 \Big) + \sum_{\bar{k} \in \bar{\mathcal{J}}} \bar{c}_{\bar{k}} \diamond \boxed{\alpha_\zeta \gamma_\zeta \bar{Z}_{\bar{k}}(\bar{s})}$;

- $\boxed{l}_2 := \sum_{i\in[m]} \left(\sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \diamond \boxed{\beta\left(\gamma_v \Psi^V_{i,k}(\bar{s}) + \gamma_w \Psi^W_{i,k}(\bar{s}) + \gamma_y \Psi^Y_{i,k}(\bar{s})\right)}_2 \right)$
  $+ \sum_{\bar{k}\in\bar{\mathcal{J}}} \bar{c}_{\bar{k}} \diamond \boxed{\beta'\left(\gamma'_v \bar{V}_{\bar{k}}(\bar{s}) + \gamma'_w \bar{W}_{\bar{k}}(\bar{s}) + \gamma'_y \bar{Y}_{\bar{k}}(\bar{s})\right)}_2$;
- $\boxed{\bar{h}'}_2 := \boxed{\gamma_h \cdot \dfrac{\bar{V}(\bar{s})\bar{W}(\bar{s}) - \bar{Y}(\bar{s})}{\bar{t}(\bar{s})}}_2$;
- $\boxed{\bar{h}''}_2 := \boxed{\alpha\gamma_h \cdot \dfrac{\bar{V}(\bar{s})\bar{W}(\bar{s}) - \bar{Y}(\bar{s})}{\bar{t}(\bar{s})}}_2$;
- $\boxed{\mathsf{out}}_1 := f_{\mathsf{AC}}\left(\{\vec{a}_i\}_{i\in[m]}\right)$, where f_{AC} is defined in Eq. (26) and $\vec{a}_i := \Big(\boxed{v'_{i,\mathsf{io}}}_1, \boxed{v'_{i,\mathsf{mid}}}_1, \boxed{w'_{i,\mathsf{io}}}_1, \boxed{w'_{i,\mathsf{mid}}}_1, \boxed{y'_{i,\mathsf{io}}}_1, \boxed{y'_{i,\mathsf{mid}}}_1, \boxed{h'_i}_1, \boxed{t_i}_1\Big)$.
- $\boxed{\mathsf{out}'}_1 := f'_{\mathsf{AC}}\Big(\{a_{i,7}, a'_i, a''_i\}_{i\in[m]}\Big)$, where f'_{AC} is defined in Eq. (27) and $a_{i,7} := \boxed{h'_i}_1$, $a'_i := \boxed{\alpha_{i,h}}_1$, $a''_i := \boxed{h''_i}_1 = \boxed{\alpha_{i,h} h'_i}_1$.

During verification, $\left\{\left(\boxed{\bar{\zeta}'_{\mathsf{mid}}}_2, \boxed{\bar{\zeta}''_{\mathsf{mid}}}_2\right)\right\}, \left(\boxed{\bar{h}'}_2, \boxed{\bar{h}''}_2\right)$ all satisfy the scaling relation w.r.t. $\{\alpha_\zeta\}$, α_h specified in Step 2 of **Procedure 5**, for $\zeta \in \{v, w, y\}$.

It is also clear to see that as in Step 5 of **Procedure 4**, we have:

$$
\begin{aligned}
l_{\bar{\mathcal{L}}} := \sum_{i\in[m]} \Big(\beta\Big(\gamma_v \Big(&\boxed{v'_{i,\mathsf{mid}}}_1 \cdot \bar{V}_{\mathsf{Idx}_v}(\bar{s}) + \boxed{w'_{i,\mathsf{mid}}}_1 \cdot \bar{V}_{\mathsf{Idx}_w}(\bar{s}) + \boxed{y'_{i,\mathsf{mid}}}_1 \cdot \bar{V}_{\mathsf{Idx}_y}(\bar{s})\Big) \\
&+\gamma_w \Big(\boxed{v'_{i,\mathsf{mid}}}_1 \cdot \bar{W}_{\mathsf{Idx}_v}(\bar{s}) + \boxed{w'_{i,\mathsf{mid}}}_1 \cdot \bar{W}_{\mathsf{Idx}_w}(\bar{s}) + \boxed{y'_{i,\mathsf{mid}}}_1 \cdot \bar{W}_{\mathsf{Idx}_y}(\bar{s})\Big) \\
&+\gamma_y \Big(\boxed{v'_{i,\mathsf{mid}}}_1 \cdot \bar{Y}_{\mathsf{Idx}_v}(\bar{s}) + \boxed{w'_{i,\mathsf{mid}}}_1 \cdot \bar{Y}_{\mathsf{Idx}_w}(\bar{s}) + \boxed{y'_{i,\mathsf{mid}}}_1 \cdot \bar{Y}_{\mathsf{Idx}_y}(\bar{s})\Big)\Big)\Big) \\
= \beta \cdot \Bigg(&\sum_{i\in[m]} \boxed{v'_{i,\mathsf{mid}}}_1 \cdot \left(\gamma_v \bar{V}_{\mathsf{Idx}_v}(\bar{s}) + \gamma_w \bar{W}_{\mathsf{Idx}_v}(\bar{s}) + \gamma_y \bar{Y}_{\mathsf{Idx}_v}(\bar{s})\right) \\
&+ \sum_{i\in[m]} \boxed{w'_{i,\mathsf{mid}}}_1 \cdot \left(\gamma_v \bar{V}_{\mathsf{Idx}_w}(\bar{s}) + \gamma_w \bar{W}_{\mathsf{Idx}_w}(\bar{s}) + \gamma_y \bar{Y}_{\mathsf{Idx}_w}(\bar{s})\right) \\
&+ \sum_{i\in[m]} \boxed{y'_{i,\mathsf{mid}}}_1 \cdot \left(\gamma_v \bar{V}_{\mathsf{Idx}_y}(\bar{s}) + \gamma_w \bar{W}_{\mathsf{Idx}_y}(\bar{s}) + \gamma_y \bar{Y}_{\mathsf{Idx}_y}(\bar{s})\right)\Bigg) \\
= \beta \cdot &\sum_{\bar{k}\in\bar{\mathcal{L}}} \left(\gamma_v \cdot \bar{c}_{\bar{k}} \bar{V}_{\bar{k}}(\bar{s}) + \gamma_w \cdot \bar{c}_{\bar{k}} \bar{W}_{\bar{k}}(\bar{s}) + \gamma_y \cdot \bar{c}_{\bar{k}} \bar{Y}_{\bar{k}}(\bar{s})\right) \\
= \beta \cdot &\left(\gamma_v \bar{V}_{\bar{\mathcal{L}}}(\bar{s}) + \gamma_w \bar{W}_{\bar{\mathcal{L}}}(\bar{s}) + \gamma_y \bar{Y}_{\bar{\mathcal{L}}}(\bar{s})\right),
\end{aligned}
$$

where $\bar{c}_{\bar{k}} = \boxed{\zeta'_{i,\mathsf{mid}}}_1$, if $\mathsf{Idx}_\zeta(i) = \bar{k}$. Thus, in Step 4 of **Procedure 5**, we have:

$$
\begin{aligned}
l &= \beta \cdot \left(\gamma_v (\bar{V}_{\bar{\mathcal{L}}}(\bar{s}) + \bar{V}_{\bar{\mathcal{J}}}(s)) + \gamma_w (\bar{W}_{\bar{\mathcal{L}}}(\bar{s}) + \bar{W}_{\bar{\mathcal{J}}}(\bar{s})) + \gamma_y (\bar{Y}_{\bar{\mathcal{L}}}(\bar{s}) + \bar{Y}_{\bar{\mathcal{J}}}(\bar{s}))\right) \\
&= \beta \cdot \left(\gamma_v \bar{V}_{\mathsf{mid}}(\bar{s}) + \gamma_w \bar{W}_{\mathsf{mid}}(\bar{s}) + \gamma_y \bar{Y}_{\mathsf{mid}}(\bar{s})\right).
\end{aligned}
$$

For $\boxed{\mathsf{out}}_1$, observe that the underlying encoded plaintext value equals to:

$$\sum_{i\in[m]} s^{(i-1)2d} \cdot ((\gamma_{i,v}\,(V_{i,\mathsf{io}}(s) + V_{i,\mathsf{mid}}(s))) \cdot (\gamma_{i,w}\,(W_{i,\mathsf{io}}(s) + W_{i,\mathsf{mid}}(s))) - (\gamma_{i,y}\,(Y_{i,\mathsf{io}}(s) + Y_{i,\mathsf{mid}}(s)))$$
$$- \gamma_{i,y} H_i(s) \cdot t_i(s))$$
$$= \sum_{i\in[m]} s^{(i-1)2d} \gamma_{i,y}\,(V_i(s) \cdot W_i(s) - Y_i(s) - H_i(s) \cdot t_i(s)) = 0,$$

since $\gamma_{i,y} = \gamma_{i,v} \cdot \gamma_{i,w}$ for all $i \in [m]$.

For $\boxed{\mathsf{out}'}_1$, the underlying encoded plaintext value equals to:

$$\sum_{i\in[m]} \left(s^{(i-1)2d} \cdot (\gamma_{i,y} H_i(s) \cdot \alpha_{i,h} - \alpha_{i,h}\gamma_{i,y} H_i(s)) \right) = 0.$$

In short, we have that $\boxed{\mathsf{out}}_1$ and $\boxed{\mathsf{out}'}_1$ both decode to zero, and thus f_{AC} and f'_{AC} evaluate to zero, which fulfill Eq. (23) and Eq. (24) in Step 3 of **Procedure 5**.

The satisfiability check of AC in Eq. (25) also passes in a straightforward way with an honest master prover. Therefore, we conclude that given $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$ it holds that $\mathsf{Verify}(\mathsf{VK}, \mathbb{x}, \pi) = 1$, and this completes the completeness proof.

7 Knowledge Soundness

In this section, we formally prove the knowledge soundness of Theorem 4.1. We first present the batched satisfiability lemma, which enables a succinct proof in our aggregation.

Lemma 7.1 (Batched Satisfiability)**.** Let $F(x) = \sum_{i\in[m]} x^{(i-1)2d} \cdot f_i(x)$ and $f_i(x) = V_i(x)W_i(x) - Y_i(x) - H_i(x)t_i(x)$. For $s \xleftarrow{\$} \mathbb{A}^*$, if $\forall i \in [m], f_i(s) = 0$, then we have that $F(s) = 0$. If $F(s) = 0$ and $f_i(x)$ has degree at most $2d$ for all $i \in [m]$, then $\Pr[\forall i, f_i(s) = 0] \geq 1 - \frac{2md}{|\mathbb{A}^*|}$.

Proof. It is clear to see that if $\forall i \in [m], f_i(s) = 0$, then $F(s) = 0$. The other direction is straightforward by applying the generalized Schwartz-Zippel lemma [27, Lemma 2]. □

Based on Lemma 7.1, we now prove the knowledge soundness (see Theorem 4.1) of our protocol given in Section 4.

Assume that the $k^{(1)}$-PDH and $k'^{(1)}$-PKE assumptions hold for the encoding scheme $\Pi_E^{(1)}$, and that the $k^{(2)}$-PDH and $k'^{(2)}$-PKE assumptions hold for $\Pi_E^{(2)}$, for parameters $k^{(1)}, k'^{(1)}, k^{(2)}, k'^{(2)}$ to be fixed later. Let $(\mathsf{VK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbb{i}, m)$. Then, for any PPT adversary $\mathcal{A}$ and any output $(\mathbb{x}, \pi) \leftarrow \mathcal{A}(\mathsf{PK})$, we construct an extractor $\mathcal{E}$ such that $\mathbb{w} \leftarrow \mathcal{E}(\mathsf{PK}, \mathbb{x}, \pi)$, and the following holds:

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{VK}, \mathbb{x}, \pi) = 1 \\ \wedge\, \mathrm{R}_{\mathsf{SAT}}(\mathbb{i}, \mathbb{x}, \mathbb{w}) = 0 \end{array}\right] = \mathsf{negl}(\lambda).$$

Overview. We will show that the extractor $\mathcal{E}$ can either extract a valid witness or break the PDH assumption using $\mathcal{A}$. The extractor $\mathcal{E}$ is given $k^{(1)}$-PDH and $k^{(2)}$-PDH instances, and plays the role of the designated verifier; it provides the PK to $\mathcal{A}$ based on the given PDH instances and receives a proof, from which $\mathcal{E}$ can either extract a valid witness $\mathbb{w}$ or break the given PDH instances. We split our soundness proof into four parts:

1. We show how $\mathcal{E}$ constructs PK from the PDH instances so that PK is computationally indistinguishable from the output of Setup. Specifically, the $k^{(1)}$-PDH instance is $\left(\mathsf{pk}_1, \left\{\boxed{s^j}_1\right\}_{j\in[2k^{(1)}]\setminus\{k^{(1)}+1\}}\right)$ and the $k^{(2)}$-PDH instance is $\left(\mathsf{pk}_2, \left\{\boxed{\bar{s}^j}_2\right\}_{j\in[2k^{(2)}]\setminus\{k^{(2)}+1\}}\right)$ for $\Pi_E^{(1)}$ and $\Pi_E^{(2)}$ respectively, where s and $\bar{s}$ are independently sampled from $\mathbb{A}_t^*$ and $\mathbb{A}_q^*$, respectively. During this step, $\mathcal{E}$ meticulously picks the randomness (e.g., $\gamma_v, \alpha_v, \beta$), so that it can efficiently derive all required encodings in crs and the rest of PK.

2. If $\mathsf{Verify}(\mathsf{VK}, \mathbb{x}, \pi) = 1$ (i.e., all checks passed as specified in Section 4.6), then we show how $\mathcal{E}$ can extract polynomials $\bar{V}_{\mathsf{mid}}(x), \bar{W}_{\mathsf{mid}}(x), \bar{Y}_{\mathsf{mid}}(x)$ from the proof π based on the $k'^{(1)}$-PKE and $k'^{(2)}$-PKE assumptions (with $k'^{(1)}, k'^{(2)}$ to be fixed later).

3. We then argue that if certain properties (to be explained in later paragraphs) hold, not only are the extracted polynomials constructed from the same set of consistent coefficients, but this coefficient set also serves as our consistent wire value assignment for the original circuit. I.e., the constructed polynomials:

$$\bar{V}(x) = \bar{V}_{\mathsf{io}}(x) + \bar{V}_{\bar{\mathcal{L}}}(x) + \bar{V}_{\bar{\mathcal{J}}}(x),$$
$$\bar{W}(x) = \bar{W}_{\mathsf{io}}(x) + \bar{W}_{\bar{\mathcal{L}}}(x) + \bar{W}_{\bar{\mathcal{J}}}(x),$$
$$\bar{Y}(x) = \bar{Y}_{\mathsf{io}}(x) + \bar{Y}_{\bar{\mathcal{L}}}(x) + \bar{Y}_{\bar{\mathcal{J}}}(x),$$

   satisfy that $\bar{V}(x)\bar{W}(x) - \bar{Y}(x) = \bar{H}(x)\bar{t}(x)$ and $\sum_{i\in[m]} s^{(i-1)2d}(V_i(x)W_i(x) - Y_i(x) - H_i(x)t_i(x)) = 0$, i.e., $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathrm{R}_{\mathsf{SAT}}$.

4. In contrast, if one of the properties *does not* hold, based on Lemma 2.1, then we utilize the extracted polynomials from Step 2 above to break either the $k^{(1)}$-PDH assumption for $\Pi_E^{(1)}$ *or* the $k^{(2)}$-PDH assumption for $\Pi_E^{(2)}$.

Construction of PK. Recall that m is the number of sub-provers, d is the maximum number of multiplication gates in each sub-circuit, $\bar{d}$ is the number of multiplication gates in AC. With $k^{(1)} = (2m-1)2d+3$ and $k^{(2)} = \mathsf{w}+3\bar{d}+3$ as in Theorem 4.1, where $\mathsf{w} := \sum_{i\in[m]} |\mathcal{I}_{i,\mathsf{mid}}| + |\bar{\mathcal{J}}|$, given the $((2m-1)2d+1)$-PDH instance $\left(\mathsf{pk}_1, \left\{\boxed{s^j}_1\right\}_{j\in\{0,\ldots,(2m-1)2d+3\}}\right)$ under $\Pi_E^{(1)}$, and $(\mathsf{w}+3\bar{d}+3)$-PDH instance $\left(\mathsf{pk}_2, \left\{\boxed{\bar{s}^j}_2\right\}_{j\in\{0,\ldots,\mathsf{w}+3\bar{d}+3\}}\right)$ under $\Pi_E^{(2)}$, the extractor sets $\mathsf{PK} = (\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{crs})$, where crs is computed as follows:

- $\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,k}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,k}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,k}(s)}_1\right\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}$: The extractor $\mathcal{E}$ samples $r_{i,v}, r_{i,w} \xleftarrow{\$} \mathcal{R}_t^*$, where t is the plaintext modulus of $\Pi_E^{(1)}$, and sets $r_{i,y} = r_{i,v}\cdot r_{i,w}$. Let $\gamma_{i,v} = r_{i,v}\cdot s^{(2m-1)d+1}$, $\gamma_{i,w} = r_{i,w}$, and thus $\gamma_{i,y} = \gamma_{i,v}\cdot\gamma_{i,w} = r_{i,y}\cdot s^{(2m-1)d+1}$ [8]. For

[8] We need the offset by $(2m-1)d$ to guarantee that when invoking the $k^{(1)}$-PDH assumption w.r.t. the tuple $\left(\boxed{h_i'}_1, \boxed{h_i''}_1\right)$, all other encodings in the crs (denoted as auxiliary input z in Assumption 2.2) are efficiently generated by Z. Since the α-relation checks for $V_i(x), W_i(x), Y_i(x)$ are replaced by a single check on $\bar{Z}_{\mathsf{mid}}(x)$, we do not need offsets for V_i, W_i, Y_i separately as in Rinocchio. However, because $H_i(s)$ is multiplied by $\gamma_{i,y}$, we need the same offset on $V_i(s)W_i(s)$ and $Y_i(s)$, so that the batched satisfiability check in f_{AC} still evaluates to zero.

$Z \in \{V, W, Y\}$ and $\zeta \in \{v, w, y\}$, the extractor $\mathcal{E}$ computes $Z_{i,k}(x) = \sum_{j=0}^{d} \zeta_{i,k,j} x^j$ using the public structure of the sub-circuits, where $\zeta_{i,k,j}$ denotes the j-th coefficient of the polynomial $Z_{i,k}(x)$, and sets these encodings to be equal to:

$$\left\{ \sum_{j\in\{0,\ldots,d\}} (\gamma_{i,v} \cdot v_{i,k,j}) \diamond \boxed{s^{(i-1)d+j}}_1 := \sum_{j\in\{0,\ldots,d\}} (r_{i,v} \cdot v_{i,k,j}) \diamond \boxed{s^{(2m+i-2)d+j+1}}_1 \right\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}},$$

$$\left\{ \sum_{j\in\{0,\ldots,d\}} (\gamma_{i,w} \cdot w_{i,k,j}) \diamond \boxed{s^{(i-1)d+j}}_1 := \sum_{j\in\{0,\ldots,d\}} (r_{i,w} \cdot w_{i,k,j}) \diamond \boxed{s^{(i-1)d+j}}_1 \right\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}},$$

$$\left\{ \sum_{j\in\{0,\ldots,d\}} (\gamma_{i,y} \cdot y_{i,k,j}) \diamond \boxed{s^{(i-1)2d+j}}_1 := \sum_{j\in\{0,\ldots,d\}} (r_{i,y} \cdot y_{i,k,j}) \diamond \boxed{s^{(2m+i-2)d+j+1}}_1 \right\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}.$$

With $k^{(1)} \geq (3m-1)d+1$, the extractor $\mathcal{E}$ has access to all encodings $\left\{\boxed{s^{j'}}_1\right\}$ as part of the $k^{(1)}$-PDH instance. Notice that we use solid boxes to represent everything that is included in the final PK, though they might be derived based on initial encodings provided in a $k^{(1)}$-PDH (or $k^{(2)}$-PDH) instance.

- $\left\{\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,\kappa}(s)}_1, \boxed{\gamma_{i,w}s^{(i-1)d}W_{i,\kappa}(s)}_1, \boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,\kappa}(s)}_1\right\}_{i\in[m],\kappa\in\mathcal{I}_{i,\mathsf{io}}}$: By following exactly the same procedure as above, the extractor $\mathcal{E}$ sets these encodings to be equal to:

$$\left\{ \sum_{j\in\{0,\ldots,d\}} (r_{i,v} \cdot v_{i,\kappa,j}) \diamond \boxed{s^{(2m+i-2)d+j+1}}_1 \right\}_{i\in[m],\kappa\in\mathcal{I}_{i,\mathsf{io}}},$$

$$\left\{ \sum_{j\in\{0,\ldots,d\}} (r_{i,w} \cdot w_{i,\kappa,j}) \diamond \boxed{s^{(i-1)d+j}}_1 \right\}_{i\in[m],\kappa\in\mathcal{I}_{i,\mathsf{io}}},$$

$$\left\{ \sum_{j\in\{0,\ldots,d\}} (r_{i,y} \cdot y_{i,\kappa,j}) \diamond \boxed{s^{(2m+i-2)d+j+1}}_1 \right\}_{i\in[m],\kappa\in\mathcal{I}_{i,\mathsf{io}}}.$$

- $\left\{\boxed{\gamma_{i,y}s^{j}}_1\right\}_{j\in\{(i-1)2d,(i-1)2d+d\}}$: With the $r_{i,y}$ sampled in previous steps, $\forall i \in [m]$, the extractor $\mathcal{E}$ sets:

$$\left\{ \boxed{\gamma_{i,y}s^{j}}_1 := r_{i,y} \diamond \boxed{s^{(2m-1)d+j+1}}_1 \right\}_{j\in\{(i-1)2d,\ldots,(i-1)2d+d\}}.$$

With $k^{(1)} \geq (2m-1)2d+1$, the extractor $\mathcal{E}$ has access to all encoding $\left\{\boxed{s^{j'}}_1\right\}$ as part of the $k^{(1)}$-PDH instance.

- $\left\{\boxed{\alpha_{i,h}\gamma_{i,y}s^j}_1\right\}_{j\in\{(i-1)2d,(i-1)2d+d\}}$: Similarly to above, the extractor $\mathcal{E}$ samples $\alpha_{i,h} \xleftarrow{\$} \mathcal{R}_t^*$ and $\forall i \in [m]$ sets:

$$\left\{\boxed{\alpha_{i,h}\gamma_{i,y}s^j}_1 := (\alpha_{i,h}\cdot r_{i,y}) \diamond \boxed{s^{(2m-1)d+j+1}}_1\right\}_{j\in\{(i-1)2d,\ldots,(i-1)2d+d\}}.$$

- $\left\{\boxed{\alpha_{i,h}}_1\right\}$: The extractor $\mathcal{E}$ uses the encoding key to encrypt the randomness $\{\alpha_{i,h}\}$.
- $\left\{\boxed{t_i(s)}_1\right\}_{i\in[m]}$: The extractor $\mathcal{E}$ computes $t_i(x) = \sum_{j=0}^{d}\tau_{i,j}x^j$ based on the public structure of the circuit, and sets:

$$\left\{\boxed{t_i(s)}_1\right\}_{i\in[m]} := \left\{\sum_{j\in\{0,\ldots,d\}}\tau_{i,j}\diamond\boxed{s^j}_1\right\}_{i\in[m]}.$$

  The extractor $\mathcal{E}$ has access to $\left\{\boxed{s^j}_1\right\}$ for $j \in [d]$ as part of the $k^{(1)}$-PDH instance, since $k^{(1)} \geq d$.

- $\left\{\boxed{\bar{s}^j}_2, \boxed{\alpha\bar{s}^j}_2\right\}_{j\in[\bar{d}]}$: The extractor $\mathcal{E}$ samples $\alpha \xleftarrow{\$} \mathcal{R}_q^*$, where q is the plaintext modulus of $\Pi_E^{(2)}$, and $\boxed{\alpha\bar{s}^j}_2 := \alpha\diamond\boxed{\bar{s}^j}_2$, where $\left\{\boxed{\bar{s}^j}_2\right\}$ is directly given in the $k^{(2)}$-PDH instance, with $k^{(2)} \geq \bar{d}$.

- $\left\{\boxed{\gamma_\zeta\bar{Z}_{\bar{k}}(\bar{s})}_2\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}$: The extractor $\mathcal{E}$ samples $r_v, r_w \xleftarrow{\$} \mathcal{R}_q^*$, and sets $r_y := r_v \cdot r_w$. Let $\gamma_v = r_v, \gamma_w = r_w\cdot\bar{s}^{\bar{d}+1}$, and $\gamma_y = r_y\cdot\bar{s}^{2\bar{d}+1}$. For $Z \in \{V, W, Y\}$ and $\zeta \in \{v, w, y\}$, the extractor $\mathcal{E}$ computes $\bar{Z}_{\bar{k}}(x) = \sum_{j=0}^{\bar{d}}\bar{\zeta}_{\bar{k},j}x^j, \bar{k} \in \bar{\mathcal{I}}_{\mathsf{mid}}$ using the public structure of AC and sets these encodings to be equal to:

$$\left\{\sum_{j\in\{0,\ldots,\bar{d}\}}(\gamma_\zeta\cdot\bar{\zeta}_{\bar{k},j})\diamond\boxed{\bar{s}^j}_2 := \sum_{j\in\{0,\ldots,\bar{d}\}}(r_\zeta\cdot\bar{\zeta}_{\bar{k},j})\diamond\boxed{\bar{s}^{\kappa_\zeta+j}}_2\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}},$$

  where $\kappa_v = 0, \kappa_w = \bar{d}+1, \kappa_y = 2\bar{d}+2$. The extractor $\mathcal{E}$ has access to $\left\{\boxed{\bar{s}^{\kappa_\zeta+j}}_2\right\}, j \in \{0,\ldots,\bar{d}\}$ as part of the $k^{(2)}$-PDH instance, since $k^{(2)} \geq 3\bar{d}+2$.

- $\left\{\boxed{\alpha_\zeta\gamma_\zeta\bar{Z}_{\bar{k}}(\bar{s})}_2\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}$: Similarly to above, for $(Z,\zeta) \in \{(V,v),(W,w),(Y,y)\}$, the extractor $\mathcal{E}$ samples $\alpha_\zeta \xleftarrow{\$} \mathcal{R}_q^*$, and for $\kappa_v = 0, \kappa_w = \bar{d}+1, \kappa_y = 2\bar{d}+2$, sets the encodings to be equal to:

$$\left\{\sum_{j\in\{0,\ldots,\bar{d}\}}(\alpha_\zeta\cdot r_\zeta\cdot\bar{\zeta}_{\bar{k},j})\diamond\boxed{\bar{s}^{\kappa_\zeta+j}}_2\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}.$$

- $\left\{\boxed{\beta\left(\gamma_v\Psi^V_{i,k}(\bar{s})+\gamma_w\Psi^W_{i,k}(\bar{s})+\gamma_y\Psi^Y_{i,k}(\bar{s})\right)}_2\right\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}, \left\{\boxed{\beta\left(\gamma_v\bar{V}_{\bar{k}}(\bar{s})+\gamma_w\bar{W}_{\bar{k}}(\bar{s})+\gamma_y\bar{Y}_{\bar{k}}(\bar{s})\right)}_2\right\}_{\bar{k}\in\bar{\mathcal{J}}}$:
The extractor $\mathcal{E}$ samples a polynomial $b(x)\in\mathbb{A}_q^*[x]_{\leq 3\bar{d}+3}$ uniformly at random, where $\mathbb{A}_q\subset\mathcal{R}_q$ is the exception set of the plaintext space of $\Pi_E^{(2)}$, subject to the constraint that the polynomials:

$$b(x)\cdot(r_v\Psi^V_{i,k}(x)+r_w x^{\bar{d}+1}\Psi^W_{i,k}(x)+r_y x^{2(\bar{d}+1)}\Psi^Y_{i,k}(x)) \text{ and}$$

$$b(x)\cdot(r_v\bar{V}_{\bar{k}}(x)+r_w x^{\bar{d}+1}\bar{W}_{\bar{k}}(x)+r_y x^{2(\bar{d}+1)}\bar{Y}_{\bar{k}}(x))$$

have a zero coefficient for the monomial $x^{3\bar{d}+3}$ for all $i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}},\bar{k}\in\bar{\mathcal{J}}$.

Set $\beta=\bar{s}^{k^{(2)}-(3\bar{d}+3)}b(\bar{s})$, where $k^{(2)}=\mathsf{w}+3\bar{d}+3$ as set in the beginning. Notice that in our proof, the polynomial $x^{k^{(2)}-(3\bar{d}+3)}b(x)$ will serve the role of $a(x)$ in Lemma 2.1 when proving the consistency for the witness across $\bar{V}_{\mathsf{mid}}(x),\bar{W}_{\mathsf{mid}}(x),\bar{Y}_{\mathsf{mid}}(x)$.

Recall that we have the entangled encodings as defined in Eq. (9):

$$\begin{aligned}\boxed{\Psi^Z_{i,k}(\bar{s})}_2 &= \sharp\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,k}(s)}\sharp\diamond\boxed{\bar{Z}_{\mathsf{Idx}_v(i)}(\bar{s})}_2+\sharp\boxed{\gamma_{i,w}s^{(i-1)d}W_{i,k}(s)}\sharp\diamond\boxed{\bar{Z}_{\mathsf{Idx}_w(i)}(\bar{s})}_2\\&\quad+\sharp\boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,k}(s)}\sharp\diamond\boxed{\bar{Z}_{\mathsf{Idx}_y(i)}(\bar{s})}_2,\end{aligned}$$

where $\bar{Z}_{\mathsf{Idx}_v(i)}(x):=\sum_{j\in\bar{d}}\bar{\zeta}_{i,v,j}x^j$ and $\bar{Z}_{\mathsf{Idx}_w(i)}(x)$ and $\bar{Z}_{\mathsf{Idx}_y(i)}(x)$ are defined similarly.

Therefore, $\mathcal{E}$ derives the encodings $\boxed{\beta\left(\gamma_v\Psi^V_{i,k}(\bar{s})+\gamma_w\Psi^W_{i,k}(\bar{s})+\gamma_y\Psi^Y_{i,k}(\bar{s})\right)}_2$ to be equal to:

$$\begin{aligned}&\left\{\sum_{j\in\bar{d}}\sum_{\zeta\in\{v,w,y\}}\beta\cdot\gamma_\zeta\cdot\left(\left(\bar{\zeta}_{i,v,j}\cdot\sharp\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,k}(s)}\sharp\right)\diamond\boxed{\bar{s}^j}_2+\left(\bar{\zeta}_{i,w,j}\cdot\sharp\boxed{\gamma_{i,w}s^{(i-1)d}W_{i,k}(s)}\sharp\right)\diamond\boxed{\bar{s}^j}_2\right.\right.\\&\left.\left.+\left(\bar{\zeta}_{i,y,j}\cdot\sharp\boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,k}(s)}\sharp\right)\diamond\boxed{\bar{s}^j}_2\right)\right\}\\=&\left\{\sum_{j\in\bar{d}}\sum_{\zeta\in\{v,w,y\}}\beta\cdot r_\zeta\cdot\left(\left(\bar{\zeta}_{i,v,j}\cdot\sharp\boxed{\gamma_{i,v}s^{(i-1)d}V_{i,k}(s)}\sharp\right)\diamond\boxed{\bar{s}^{\kappa_\zeta+j}}_2\right.\right.\\&\left.\left.+\left(\bar{\zeta}_{i,w,j}\cdot\sharp\boxed{\gamma_{i,w}s^{(i-1)d}W_{i,k}(s)}\sharp\right)\diamond\boxed{\bar{s}^{\kappa_\zeta+j}}_2+\left(\bar{\zeta}_{i,y,j}\cdot\sharp\boxed{\gamma_{i,y}s^{(i-1)2d}Y_{i,k}(s)}\sharp\right)\diamond\boxed{\bar{s}^{\kappa_\zeta+j}}_2\right)\right\},\end{aligned}$$

where $\kappa_v=0,\kappa_w=\bar{d}+1,\kappa_y=2\bar{d}+2$. Similarly, $\bar{Z}_{\bar{k}}(x)=\sum_{\bar{k}\in\bar{d}}\bar{\zeta}_{i,\bar{k},j'}x^{j'}$, and $\mathcal{E}$ computes the encoding $\boxed{\beta\left(\gamma_v\bar{V}_{\bar{k}}(\bar{s})+\gamma_w\bar{W}_{\bar{k}}(\bar{s})+\gamma_y\bar{Y}_{\bar{k}}(\bar{s})\right)}_2$ to be equal to:

$$\left\{\sum_{j'\in\bar{d}}\sum_{\zeta\in\{v,w,y\}}(\beta\cdot\gamma_\zeta\cdot\bar{\zeta}_{i,\bar{k},j'})\diamond\boxed{\bar{s}^{j'}}_2=\sum_{j'\in\bar{d}}\sum_{\zeta\in\{v,w,y\}}(\beta\cdot r_\zeta\cdot\bar{\zeta}_{i,\bar{k},j'})\diamond\boxed{\bar{s}^{\kappa_\zeta+j'}}_2\right\},$$

The extractor $\mathcal{E}$ has access to all required $\left\{\boxed{\bar{s}^j}_2\right\}$ as part of the $k^{(2)}$-PDH instance since $k^{(2)}\geq 3\bar{d}+2$.

Therefore, we conclude that all encodings in crs can be successfully constructed by $\mathcal{E}$. By following the same argument as in [27, Section 5.2], we state that crs generated by $\mathcal{E}$ has a distribution that is indistinguishable from the one in Dinocchio.

Then, $\mathcal{E}$ sends the crs to $\mathcal{A}$, which returns a proof π.

Polynomial extraction. We now utilize the proof π and the $k'^{(2)}$-PKE assumption to extract polynomials $\bar{V}_{\mathsf{mid}}(x), \bar{W}_{\mathsf{mid}}(x), \bar{Y}_{\mathsf{mid}}(x), \bar{H}(x)$, all of degree at most $\bar{d}$, where $k'^{(2)} = 3\bar{d}+2$ as specified in later paragraphs. To extract these polynomials, we first argue that the crs given to the adversary aligns with the view of the adversary in $k'^{(2)}$-PKE assumption, and then we invoke the $k'^{(2)}$-PKE extractor to recover the polynomial coefficients.

We first parse the proof π as in Step 1 in Fig. 12.

- For the tuple $\left(\boxed{\bar{h}}_2, \boxed{\bar{h}'}_2\right)$, we extract the underlying polynomial $\bar{H}(x)$ of degree $\bar{d}$ in the following way:
  - Given the instance pp of the $\bar{d}$-PKE assumption with respect to the encoding scheme $\mathsf{E}^{(2)}$ such that:
    $$\mathsf{pp} := \left\{\mathsf{E}^{(2)}(\bar{s}^j), \mathsf{E}^{(2)}(\alpha\bar{s}^j)\right\}_{j\in\{0,\ldots,\bar{d}\}},$$
    a PPT algorithm Z can construct $z = \mathsf{crs}\backslash\mathsf{pp}$ by utilizing the encodings $\left\{\boxed{\bar{s}^j}_2\right\}_{j\in\{0,\ldots,\bar{d}\}}$ and sampling the randomness, including $s \in \mathbb{A}_t^*$, $\{\alpha_{i,h}, \gamma_{i,v}, \gamma_{i,w}, \gamma_{i,y}\}_{i\in[m]} \in \mathcal{R}_t^*$, and $\alpha, \alpha_v, \alpha_w, \alpha_y, \gamma_v, \gamma_w, \gamma_y, \beta \in \mathcal{R}_q^*$, uniformly at random (i.e., independently of $\bar{s}$). Observe that this z generated by Z is from the same distribution as the encodings of PK as described in the previous step.

    We treat the prover side as the "non-uniform PPT algorithm $\mathcal{A}$" in the $\bar{d}$-PKE assumption, with input $\mathsf{crs} := (\mathsf{pp}, z)$.
  - Given $\mathcal{A}$'s output tuple $\left(\boxed{\bar{h}}_2, \boxed{\bar{h}'}_2\right)$, $\mathcal{E}$ extracts the polynomial $\bar{H}(x)$ of degree at most $\bar{d}$ based on the $\bar{d}$-PKE assumption of $\Pi_E^{(2)}$, similarly to the security proof in Rinocchio [27].
- For the tuples $\left(\boxed{\bar{\zeta}'_{\mathsf{mid}}}_2, \boxed{\bar{\zeta}''_{\mathsf{mid}}}_2\right)$, for $\zeta \in \{v, w, y\}$. As before, the crs can be parsed as:
  $$\left(\left\{\boxed{r_\zeta \bar{s}^{\kappa_\zeta} \bar{Z}_{\bar{k}}(s)}_2\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}, \left\{\boxed{\alpha_\zeta r_\zeta \bar{s}^{\kappa_\zeta} \bar{Z}_{\bar{k}}(s)}_2\right\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}}, z\right),$$
  where $\kappa_v = 0, \kappa_w = \bar{d}+1, \kappa_y = 2\bar{d}+2$. Similarly to above, z can be efficiently generated using the encodings $\left\{\mathsf{E}_{r_v}^{(2)}(\bar{s}^j),\ \mathsf{E}_{r_v}^{(2)}(\alpha_v\bar{s}^j)\right\}_{j\in\{0,\ldots,\bar{d}\}}$ and randomness sampled by $\mathcal{E}$. Thus, based on the $(\kappa_\zeta + \bar{d})$-PKE assumption, we can extract the polynomial $\bar{Z}_{\mathsf{mid}}(x)$ of degree at most $\bar{d}$ for $(Z, \zeta) \in \{(V, v), (W, w), (Y, y)\}$. See also Remark 2.1.

Consistency of wire value assignment. Up to this point, $\mathcal{E}$ has extracted the polynomials:

$$\bar{V}_{\mathsf{mid}}(x) = \sum_{j=0}^{\bar{d}} \bar{v}_j x^j, \qquad \bar{W}_{\mathsf{mid}}(x) = \sum_{j=0}^{\bar{d}} \bar{w}_j x^j, \qquad \bar{Y}_{\mathsf{mid}}(x) = \sum_{j=0}^{\bar{d}} \bar{y}_j x^j.$$

We now argue that if the following three properties hold, then we can extract a witness set $\{c_{i,k}\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}$ s.t. for each sub-circuit, we have $Z_{i,\mathsf{mid}}(x) = \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k}\cdot Z_{i,k}(x), Z_i(x) = Z_{i,\mathsf{io}}(x) + Z_{i,\mathsf{mid}}(x)$, $Z\in\{V,W,Y\}$, and:

$$\sum_{i\in[m]} x^{(i-1)2d}\cdot(V_i(x)W_i(x) - Y_i(x) - H_i(x)t_i(x)) = 0.$$

On the contrary, if any one of the properties does *not* hold, we can break either $k^{(1)}$-PDH of $\Pi_E^{(1)}$ or $k^{(2)}$-PDH of $\Pi_E^{(2)}$.

Roughly speaking, *Property 1* guarantees that the extracted polynomials $\{\bar{Z}_{\mathsf{mid}}(x)\}$ have the same set of coefficients w.r.t. their corresponding sub-polynomials. *Property 2* then claims that this consistent set of coefficients serves as our consistent wire-value assignment that satisfies the AC. In the end, *Property 3* ensures that the consistent coefficients also satisfy all sub-circuits. We formally list the three properties as follows.

Property 1: The polynomial:

$$U(x) = r_v\bar{V}_{\mathsf{mid}}(x) + r_w x^{\bar{d}+1}\bar{W}_{\mathsf{mid}}(x) + r_y x^{2(\bar{d}+1)}\bar{Y}_{\mathsf{mid}}(x) \tag{30}$$

belongs to the set $\mathcal{U}$ generated by the $\mathcal{R}_q$-linear combinations of the polynomials:

$$\begin{aligned}&\left\{U_{i,k}(x) = r_v\Psi_{i,k}^V(x) + r_w x^{\bar{d}+1}\Psi_{i,k}^W(x) + r_y x^{2\bar{d}+2}\Psi_{i,k}^Y(x)\right\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}\\ \cup&\left\{U_{\bar{k}}(x) = r_v\bar{V}_{\bar{k}}(x) + r_w x^{\bar{d}+1}\bar{W}_{\bar{k}}(x) + r_y x^{2\bar{d}+2}\bar{Y}_{\bar{k}}(x)\right\}_{\bar{k}\in\bar{\mathcal{J}}}\end{aligned} \tag{31}$$

Property 2: $\bar{V}(x)\cdot\bar{W}(x) - \bar{Y}(x) = \bar{H}(x)\cdot\bar{t}(x)$.

Property 3: $\sum_{i\in[m]} x^{(i-1)2d}(V_i(x)W_i(x) - Y_i(x)) = \sum_{i\in[m]} x^{(i-1)2d}H_i(x)t_i(x)$.

Let $\bar{V}(x) = \bar{V}_{\mathsf{io}}(x) + \bar{V}_{\mathsf{mid}}(x), \bar{W}(x) = \bar{W}_{\mathsf{io}}(x) + \bar{W}_{\mathsf{mid}}(x), \bar{Y}(x) = \bar{Y}_{\mathsf{io}}(x) + \bar{Y}_{\mathsf{mid}}(x)$, where $\bar{V}_{\mathsf{mid}}(x), \bar{W}_{\mathsf{mid}}(x), \bar{Y}_{\mathsf{mid}}(x)$ are the extracted polynomials.

If *Property 1* does hold, then $U(x)\in\mathcal{U}$, where $U(x)$ is as defined in Eq. (30). Assume that the *coefficients* of linear combination w.r.t. $\mathcal{U}$ are $\{c'_{i,k}\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}\cup\{\bar{c}'_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{J}}}$ so that:

$$\begin{aligned}U(x) :=& \sum_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}} c'_{i,k}U_{i,k}(x) + \sum_{\bar{k}\in\bar{\mathcal{J}}}\bar{c}'_{\bar{k}}U_{\bar{k}}(x)\\ =& \sum_{i,k} c'_{i,k}\cdot\left(r_v\Psi_{i,k}^V(x) + r_w x^{\bar{d}+1}\Psi_{i,k}^W(x) + r_y x^{2(\bar{d}+1)}\Psi_{i,k}^Y(x)\right)\\ &+\sum_{\bar{k}}\bar{c}'_{\bar{k}}\cdot\left(r_v\bar{V}_{\bar{k}}(x) + r_w x^{\bar{d}+1}\bar{W}_{\bar{k}}(x) + r_y x^{2(\bar{d}+1)}\bar{Y}_{\bar{k}}(x)\right)\\ =& r_v\cdot\left(\sum_{i,k} c'_{i,k}\Psi_{i,k}^V(x) + \sum_{\bar{k}\in\bar{\mathcal{J}}}\bar{c}'_{\bar{k}}\bar{V}_{\bar{k}}(x)\right) + r_w x^{\bar{d}+1}\cdot\left(\sum_{i,k} c'_{i,k}\Psi_{i,k}^W(x) + \sum_{\bar{k}\in\bar{\mathcal{J}}}\bar{c}'_{\bar{k}}\bar{W}_{\bar{k}}(x)\right)\\ &+ r_y x^{2(\bar{d}+1)}\cdot\left(\sum_{i,k} c'_{i,k}\Psi_{i,k}^Y(x) + \sum_{\bar{k}\in\bar{\mathcal{J}}}\bar{c}'_{\bar{k}}\bar{Y}_{\bar{k}}(x)\right)\\ =& r_v\cdot\bar{V}'_{\mathsf{mid}}(x) + r_w x^{\bar{d}+1}\cdot\bar{W}'_{\mathsf{mid}}(x) + r_y x^{2(\bar{d}+1)}\cdot\bar{Y}'_{\mathsf{mid}}(x)\ ,\end{aligned} \tag{32}$$

where $\bar{Z}'_{\mathsf{mid}}(x)$ represents $\sum_{i,k} c'_{i,k}\Psi^{Z}_{i,k}(s) + \sum_{\bar{k}\in\bar{\mathcal{J}}} \bar{c}'_{\bar{k}}\bar{Z}_{\bar{k}}(x)$.

Comparing Eq. (30) and Eq. (32), since the extracted polynomials $\left\{\bar{Z}_{\mathsf{mid}}(x)\right\}$ all have degree at most $\bar{d}$, we claim that $\bar{Z}_{\mathsf{mid}}(x) = \bar{Z}'_{\mathsf{mid}}(x)$, for $Z \in \{V, W, Y\}$. I.e., $\{c'_{i,k}\} \cup \{\bar{c}'_{\bar{k}}\}$ is the set of consistent coefficients for constructing $\bar{V}_{\mathsf{mid}}(x)$, $\bar{W}_{\mathsf{mid}}(x)$, and $\bar{Y}_{\mathsf{mid}}(x)$.

If *Property 2* holds, then $\bar{V}(x)\bar{W}(x) - \bar{Y}(x)$ divides the target polynomial $\bar{t}(x)$ of AC, where $\bar{Z}_{\mathsf{io}}(x)$ is derived by the verifier based on $\mathbb{x}$ and VK. We conclude that these consistent *coefficients* $\{\bar{c}'_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{I}}_{\mathsf{mid}}} := \{\bar{c}'_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{L}}} \cup \{\bar{c}'_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{J}}}$ form a valid *wire assignment* to all private wires of the aggregation circuit. Thus, there is a correspondence between the coefficients $\{c'_{i,k}\} \cup \{\bar{c}'_{\bar{k}}\}$ extracted here and the wire assignment $\{c_{i,k}\} \cup \{\bar{c}_{\bar{k}}\}$ used in the protocol presented in Section 4, s.t., $c_{i,k} := c'_{i,k}$ and $\bar{c}_{\bar{k}} := \bar{c}'_{\bar{k}}$. With $\{c_{i,k}\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}$, we then construct the polynomials $Z_{i,\mathsf{mid}}(x) = \sum_{k\in\mathcal{I}_{i,\mathsf{mid}}} c_{i,k} \cdot Z_{i,k}(x)$, for $Z \in \{V, W, Y\}$, all of degree at most d. Let $Z_i(x) = Z_{i,\mathsf{io}}(x) + Z_{i,\mathsf{mid}}(x)$.

Notice that the extracted wire values $\{\bar{c}_{\bar{k}}\}_{\bar{k}\in\bar{\mathcal{J}}}$ contain values corresponding to the tuple $\left(\boxed{h'_i}_1, \boxed{h''_i}_1\right)$ for each $i \in [m]$. Given that $\boxed{\mathsf{out}'}_1$ encodes zero as checked in Eq. (24) in Fig. 12, these tuples are all valid $(\alpha_{i,h})$-relation pairs except with probability $\frac{1}{|\mathbb{A}^*_t|}$ (where t is the plaintext modulus of $\Pi^{(1)}_E$). To extract the polynomials $H_i(x)$ of degree at most d for all $i \in [m]$, we invoke the $k'^{(1)}$-PKE assumption with the challenge instance $\sigma := \left\{\mathsf{E}^{(1)}_{r_{i,h}}(s^j), \mathsf{E}^{(1)}_{r_{i,h}}(\alpha_{i,h}s^j)\right\}_{j\in\{0,\ldots,(2m-1+i)d+1\}}$, where $k'^{(1)} = ((2m-1+i)d+1)$. Similarly to before, the auxiliary input $z := \mathsf{crs} \setminus \sigma$ can be generated by Z given pp and $r_{i,h}$. Therefore, based on Remark 2.1, $\mathcal{E}$ recovers $H_i(x)$ of degree at most d for all $i \in [m]$.

Finally, if *Property 3* also holds in addition to *Properties 1 and 2*, i.e.,

$$\sum_{i\in[m]} x^{(i-1)2d}\left(V_i(x)W_i(x) - Y_i(x)\right) = \sum_{i\in[m]} x^{(i-1)2d} H_i(x)t_i(x),$$

where $V_i(x), W_i(x), Y_i(x), H_i(x)$ are all of degree at most d, based on Lemma 7.1 we conclude that $\forall i \in [m]$, $V_i(x)W_i(x) - Y_i(x) = H_i(x)t_i(x)$. This means that the coefficients $\{c_{i,k}\}_{i\in[m],k\in\mathcal{I}_{i,\mathsf{mid}}}$ used to construct $\{V_{i,\mathsf{mid}}, W_{i,\mathsf{mid}}, Y_{i,\mathsf{mid}}\}$ are a valid (and consistent) wire-value assignment for C_i.

Reduction to $k^{(1)}$-PDH and $k^{(2)}$-PDH assumptions. As the final step of our soundness proof, we show that if one of the properties does not hold, we can craft a valid tuple of the form $\left(a, \boxed{a \cdot s^{k^{(1)}+1}}_1\right)$ or $\left(a', \boxed{a' \cdot \bar{s}^{k^{(2)}+1}}_2\right)$ to break either the $k^{(1)}$-PDH under $\Pi^{(1)}_E$ or the $k^{(2)}$-PDH assumption under $\Pi^{(2)}_E$.

At a high level, if *Property 1* does not hold, then based on Lemma 2.1, the extractor can construct a polynomial containing the monomial term $x^{k^{(2)}+1}$, whose evaluation on $\bar{s}$ is given by the adversary, i.e., our prover side. Following a similar argument as in Rinocchio, this allows us to extract a valid tuple breaking $k^{(2)}$-PDH assumption under $\Pi^{(2)}_E$. If *Property 2* does not hold, then the extractor obtains a non-zero polynomial that has $\bar{s}$ as a root. This implies that $\bar{s}^{\bar{d}^*}$ can be expressed as a linear combination of lower-degree powers of $\bar{s}$, where $\bar{d}^*$ denotes the highest degree monomial with a non-zero coefficient in that polynomial. By multiplying both sides of this linear combination equation by $\bar{s}^{k^{(2)}+1-\bar{d}^*}$, the extractor derives an expression of $\bar{s}^{k^{(2)}+1}$ as a linear combination of the monomials $\bar{s}^i$ with $i < k^{(2)}+1$, which suffices to break the $k^{(2)}$-PDH assumption under $\Pi^{(2)}_E$. Hence, we again break the $k^{(2)}$-PDH assumption under $\Pi^{(2)}_E$. The case where *Property 3* does not hold is analogous to *Property 2* but w.r.t. $\Pi^{(1)}_E$.

1. *Property 1:* ($(\mathsf{w}+3\bar{d}+3)$-PDH under $\Pi_E^{(2)}$)

   Recall that as shown in the **Construction of** PK, we sample the polynomial $b(x) \in \mathbb{A}_q^*[x]_{\leq 3\bar{d}+3}$, satisfying the constraint that the polynomials

   $$b(x)(r_v\Psi_{i,k}^V(x) + r_w x^{\bar{d}+1}\Psi_{i,k}^W(x) + r_y x^{2\bar{d}+2}\Psi_{i,k}^Y(x)),$$

   $$b(x)(r_v\bar{V}_{\bar{k}}(x) + r_w x^{\bar{d}+1}\bar{W}_{\bar{k}}(x) + r_y x^{2\bar{d}+2}\bar{Y}_{\bar{k}}(x))$$

   have a zero coefficient for the term $x^{3\bar{d}+3}$ for $i \in [m], k \in \mathcal{I}_{i,\mathsf{mid}}, \bar{k} \in \bar{\mathcal{J}}$.

   Let $a(x) = x^{k^{(2)}-(3\bar{d}+3)}b(x)$ and $\mathsf{w} := |\mathcal{U}|$ where $\mathcal{U}$ is the set defined in Eq. (31). Observe that since $k^{(2)} = \mathsf{w}+3\bar{d}+3$, $a(x)$ has degree greater than $\mathsf{w}-1$. Thus, if *Property 1* does not hold, i.e., $U(x)$ is not in the linear span of $\mathcal{U}$, based on Lemma 2.1, the polynomial

   $$G(x) = a(x)\cdot U(x) = x^{k^{(2)}-(3\bar{d}+3)}b(x)\cdot U(x)$$

   has a *non-zero* coefficient on the term $x^{k^{(2)}+1}$ with probability $1-\frac{1}{|\mathbb{A}_q^*|}$, where q is the plaintext modulus of $\Pi_E^{(2)}$. Moreover, notice that all coefficients of $G(x)$ can be directly computed by $\mathcal{E}$ since $b(x)$ is chosen by $\mathcal{E}$ and $U(x)$ is constructed based on $\bar{V}_{\mathsf{mid}}(x), \bar{W}_{\mathsf{mid}}(x), \bar{Y}_{\mathsf{mid}}(x)$. Denote by $g_{i\in\{0,\ldots,k^{(2)}+3\bar{d}+3\}}$ the coefficients of $G(x)$ (the degree of $G(x)$ is bounded by $k^{(2)}+3\bar{d}+3$). From the definition of $\{\gamma_v,\gamma_w,\gamma_y\}$ and $U(x)$, it follows that $G(\bar{s}) = \beta(\gamma_v\bar{V}_{\mathsf{mid}}(\bar{s}) + \gamma_w\bar{W}_{\mathsf{mid}}(\bar{s}) + \gamma_y\bar{Y}_{\mathsf{mid}}(\bar{s}))$. Since the linear combination checks in Step 4 in Fig. 12 is satisfied, we then have $\boxed{l}_2 := \boxed{G(\bar{s})}_2$. Hence, $\mathcal{E}$ can extract the tuple $\left(g_{k^{(2)}+1}, \boxed{g_{k^{(2)}+1}\cdot\bar{s}^{k^{(2)}+1}}_2\right)$ by subtracting all terms constructed via $\left\{\boxed{\bar{s}^j}_2\right\}_{j\in\{0,\ldots,k^{(2)}+3\bar{d}+3\}\setminus\{k^{(2)}+1\}}$, which are included in the challenge instance: $\left\{\boxed{\bar{s}^j}_2\right\}_{j\in\{0,\ldots,2k^{(2)}\}\setminus\{k^{(2)}+1\}}$, from $\boxed{l}_2$ to break the $(\mathsf{w}+3\bar{d}+3)$-PDH assumption, where $k^{(2)} = \mathsf{w}+3\bar{d}+3$.

2. *Property 2*: ($(\mathsf{w}+3\bar{d}+3)$-PDH under $\Pi_E^{(2)}$)

   If *Property 2* breaks, we have the non-zero polynomial:

   $$P(x) = \bar{V}(x)\bar{W}(x) - \bar{Y}(x) - \bar{H}(x)\bar{t}(x) = a'\cdot x^{\bar{d}^*} + P'(x)$$

   with $\bar{d}^* \leq 2\bar{d}$ and $\bar{s}$ as a root since Eq. (25) passes, where $a' \in \mathcal{R}_q$ is the coefficient of the monomial term $x^{\bar{d}^*}$ with the highest degree. Notice that both a' and all coefficients of $P'(x)$ can be efficiently computed since $\mathcal{E}$ knows $\bar{V}(x), \bar{W}(x), \bar{Y}(x), \bar{H}(x), \bar{t}(x)$. Let $\{p'_j\}_{j\in\{0,\ldots,2\bar{d}-1\}}$ denote the coefficients of $P'(x)$, which has degree at most $\bar{d}^*-1$. The extractor $\mathcal{E}$ derives:

   $$\boxed{a'\cdot\bar{s}^{k^{(2)}+1}}_2 := \boxed{-\bar{s}^{k^{(2)}+1-\bar{d}^*}P'(\bar{s})}_2 = \sum_{j\in\{0,\ldots,\bar{d}^*-1\}}(-p'_j)\diamond\boxed{\bar{s}^{k^{(2)}+1-\bar{d}^*+j}}_2,$$

   based on $\left\{\boxed{\bar{s}^j}\right\}_{j\in\{k^{(2)}+1-\bar{d}^*,\ldots,k^{(2)}\}}$, which are included in the challenge instance: $\left\{\boxed{\bar{s}^j}_2\right\}_{j\in\{0,\ldots,2k^{(2)}\}\setminus\{k^{(2)}+1\}}$ of the $k^{(2)}$-PDH assumption. Finally, the tuple $\left(a', \boxed{a'\cdot\bar{s}^{k^{(2)}+1}}_2\right)$ breaks the $\mathsf{w}+3\bar{d}+3$-PDH assumption under $\Pi_E^{(2)}$, where $k^{(2)} = \mathsf{w}+3\bar{d}+3$.

3. *Property 3*: $(((2m-1)2d+1)$-PDH under $\Pi_E^{(1)})$

   Similarly to above, if *Property 3* does not hold, we have the non-zero polynomial:

$$F(x) = \sum_{i\in[m]} x^{(i-1)2d}\left(V_i(x)W_i(x) - Y_i(x) - H_i(x)t_i(x)\right) = a'' \cdot x^{d'} + F'(x)$$

   with $d' \leq 2md$ and s as a root since $\boxed{\mathsf{out}}_1$ encodes zero as verified in Step 3 of Fig. 12, where $a'' \in \mathcal{R}_t$ is the coefficient of the monomial term $x^{d'}$ with the highest degree. The extractor $\mathcal{E}$ derives:

$$\boxed{a'' \cdot s^{k^{(1)}+1}}_1 := \boxed{-s^{k^{(1)}+1-d'}F'(s)}_1 = \sum_{j\in\{0,\ldots,d'-1\}} (-f'_j) \diamond \boxed{s^{k^{(1)}+1-d'+j}}_1 ,$$

   based on $\left\{\boxed{s^j}\right\}_{j\in\{k^{(1)}+1-d',\ldots,k^{(1)}\}}$, which are included in the challenge instance: $\left\{\boxed{s^j}_1\right\}_{j\in\{0,\ldots,2k^{(1)}\}\setminus\{k^{(1)}+1\}}$ of $k^{(1)}$-PDH assumption. Finally, the tuple $\left(a'', \boxed{a'' \cdot s^{k^{(1)}+1}}_1\right)$ breaks the $((2m-1)2d+1)$-PDH assumption under $\Pi_E^{(1)}$, where $k^{(1)} = (2m-1)2d+1$.

This completes the soundness proof of our protocol.

8 Evaluation

We demonstrate the practicality of our protocol by evaluating the distributed proof generation time for encrypted matrix multiplication (MM). We focus on this computation because matrix multiplication constitutes a fundamental linear-algebraic primitive that underlies both linear transformations in modern machine learning models [68] and core subroutines in homomorphic encryption schemes [66]. While our protocol applies to arbitrary arithmetic circuits as discussed in Remark 4.3, we focus specifically on a large matrix multiplication circuit, which enables precise and interpretable cost estimates and reflects a representative and practically relevant workload for (F)HE-based systems. Extending such concrete measurements to fully generic circuits requires further optimization in the handling of shared wires across the distributed sub-circuits, which we leave for future work.

We begin by describing the problem of MM and then provide a detailed breakdown of the costs incurred by each operation on both the prover side and verifier side.

Problem description. We consider the BFV scheme [11, 25] as the underlying fully homomorphic encryption scheme, which is a RLWE-based encryption scheme over $\mathcal{R}_{q'} := \mathbb{Z}_{q'}[X]/(X^D+1)$, where q' is the ciphertext modulus, and D, the ring dimension, is a power-of-two. We denote the plaintext modulus as t' and the underlying inputs of the MM algorithm as the matrices $A \in \mathcal{R}_{t'}^{D\times N}$ and $B \in \mathcal{R}_{t'}^{N\times M}$. W.l.o.g., we set the dimension of $A := (a_{i,j})$ to be $D \times D$, i.e., $N = D$. If $N > D$, we split the matrix into $\lceil \frac{N}{D} \rceil$ pieces and repeat our algorithm in a straightforward way. To further facilitate the FHE operations, we store A in its *rotated diagonal* form A', with each row i defined as $(a_{1,i \bmod D}, a_{2,i+1 \bmod D}, \ldots, a_{D,i+D-1 \bmod D})$.

By taking advantage of the SIMD property of BFV, the user (who is also the designated verifier in our system) encrypts each row $\vec{a}'_j$ of A' and each column $\vec{b}_{j'}$ of B as separate ciphertexts, denoted as $\mathsf{ct}_j^{(A)}, \mathsf{ct}_{j'}^{(B)}$ respectively, for $j \in [D], j' \in [M]$. Thus, the server (who is also the prover) takes

as input $\left\{\mathsf{ct}_j^{(A)}\right\}_{j\in[D]}$ and $\left\{\mathsf{ct}_{j'}^{(B)}\right\}_{j'\in[M]}$, evaluates the MM algorithm via baby-step-giant-step [30, 49], formally presented in Algorithm 1. [9] At a high level, the baby-step-giant-step algorithm (on the plaintext level) works as follows. Recall that each row of the rotated matrix A' is of form: $\vec{a}'_j := (a_{1,j \bmod D}, a_{2,j+1 \bmod D}, \ldots, a_{D,j+D-1 \bmod D})$. Then, when $\mathsf{rt} = \sqrt{D}$, $A\vec{b}$ is expressed as $\sum_{i\in[\mathsf{rt}]} \mathsf{Rot}\left(\sum_{k\in[\mathsf{rt}]}\left(\vec{a}_{j'} \cdot \mathsf{Rot}(\vec{b}, k\cdot\mathsf{rt})\right), i\right)$, where $\vec{b}$ is one column in the matrix B, $j' = i + k\cdot\mathsf{rt}$, and $\mathsf{Rot}(\vec{x}, y)$ outputs the vector $\vec{x}'$ which is $\vec{x}$ rotated y-slots to the left. Thus, it is clear to see that in addition to D multiplications, which are unavoidable, this algorithm only requires $2\sqrt{D}$ rotations and $\sqrt{D}+1$ rotation keys.

Eventually, the server, which is also the prover side of our protocol, outputs the results $\left\{\mathsf{res}_{j'} := \boxed{A\vec{b}_{j'}}\right\}_{j'\in[M]}$, together with a proof π demonstrating its honest evaluation.

Algorithm 1 Matrix Multiplication via BSGS

procedure $\mathsf{matMul}(\left\{\mathsf{ct}_j^{(A)}\right\}_{j\in[D]}, \left\{\mathsf{ct}_{j'}^{(B)}\right\}_{j'\in[M]}, \mathsf{ek}_{\mathsf{rot}})$
  $\mathsf{rt} \leftarrow \sqrt{D}$
  for $j' \in [M]$ **do**
    Initialize $\mathsf{res}_{j'}$ to encrypt all zeros
    for $j \in [\mathsf{rt}]$ **do**
      $\mathsf{ct}^{(B)}_{j',j} \leftarrow \mathsf{BFV.Rot}(\mathsf{ct}^{(B)}_{j'}, j\cdot\mathsf{rt}, \mathsf{ek}_{\mathsf{rot}})$ ▷ prepare the rt rotated encrypted vectors
    for $j \in [\mathsf{rt}]$ **do**
      Initialize res'_j to encrypt all zeros
      for $k \in [\mathsf{rt}]$ **do** ▷ for each baby step, multiply the rotated data vectors with the rotated diagonal vectors of the encrypted matrix under that small chunk
        $\mathsf{tmp} \leftarrow \mathsf{BFV.Eval}(\times, \mathsf{ct}^{(B)}_{j',k}, \mathsf{ct}^{(A)}_{k'})$, where $k' = j + k\cdot\mathsf{rt} \mod D$
        $\mathsf{res}'_j \leftarrow \mathsf{BFV.Eval}(+, \mathsf{res}'_j, \mathsf{tmp})$
    for $j \in [\mathsf{rt}-1]$ **do** ▷ rotate and add up all shares under each baby step
      $\mathsf{tmp} \leftarrow \mathsf{BFV.Rot}(\mathsf{res}'_{\mathsf{rt}-j+1}, 1, \mathsf{ek}_{\mathsf{rot}})$
      $\mathsf{res}'_{\mathsf{rt}-j} \leftarrow \mathsf{BFV.Eval}(+, \mathsf{tmp}, \mathsf{res}'_{\mathsf{rt}-j})$
    $\mathsf{res}_{j'} := \mathsf{res}'_1$ ▷ extract the result of $\boxed{A\vec{b}_{j'}}$ which is now added up res'_1
  return $\{\mathsf{res}_{j'}\}_{j'\in[M]}$

8.1 Estimations

We first provide the estimated runtime of our protocol, followed by a discussion of the detailed breakdowns. We run our micro-benchmarks on AWS EC2 c6i.4xlarge, with 16 vCPUs. The underlying FHE scheme has ring dimension $D = 4096$, ciphertext modulus $q' \approx 2^{54}$, and plaintext

[9] There exists a lot of work improving the runtime of homomorphically evaluating the matrix-to-vector or matrix-to-matrix multiplication under certain specific settings, e.g., when the matrix dimensions are much smaller than the ring dimension[70] or when the matrices are sparse [67]. Under the most general case, we use the most efficient known way of performing matrix multiplication, which adapts a baby-step-giant-step framework, as introduced in [30, 49]. This method is also used in other FHE algorithms to enhance performance, such as bootstrapping [47, 48]. We refer readers to the original paper [30, 49] for a correctness proof of this MM algorithm.

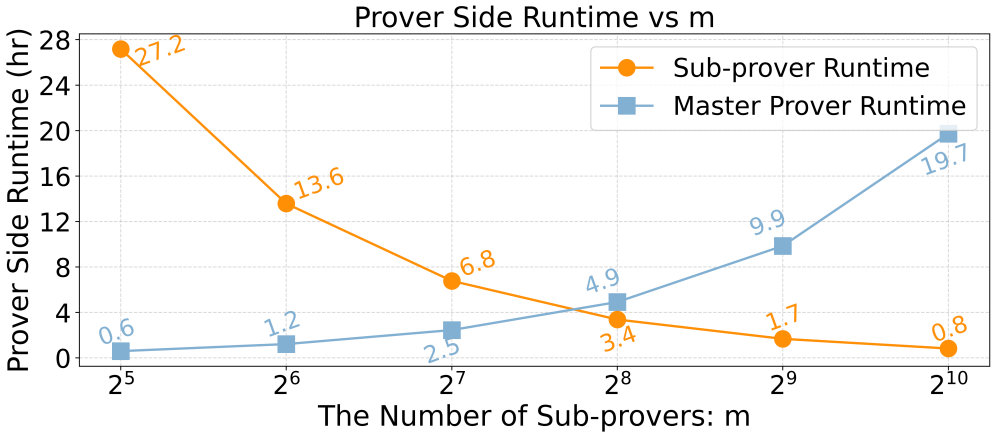

Figure 13: Prover side runtime under different numbers of sub-provers m. We set matrix dimensions to $N = D = M = 4096$.

space $t' = 2^{16}$. The encoding scheme Π_1 for sub-circuits has ring dimension $D_1 = D = 4096$, $q \approx 2^{109}$, and plaintext space q'. The encoding scheme Π_2 for aggregation circuits has ring dimension $D_2 = 16384$, $Q \approx 2^{300}$, and plaintext space q. All scheme has more than 128 bits of security based on the RLWE assumption [3].

The setup phase consists of three components, the interpolation of all sub-polynomials $\{Z_{i,k}(x), \bar{Z}_{\bar{k}}\}$, the encryption of all evaluations w.r.t. the random points $s, \bar{s}$, and the pre-processing for verification that depends on the structure of the sub-circuits. Based on our microbenchmark, for an original circuit of size 2^{27} and $m = 128$, the setup phase takes around 47 hours and can go up to 228 hours if the original circuit size is over 2^{31}. The bottleneck happens during the polynomial interpolations, and we leave the improvement for this one-time setup as future work.

We first set $N = D = M = 4096$ and range the number of sub-provers from $\{32, 64, 128, 256, 512, 1024\}$. The prover runtime is shown in Fig. 13, from which we observe that $m = 128$ provides a better tradeoff between sub-prover runtime and the aggregation overhead. Intuitively, the aggregation circuit should be slightly smaller than the size of a sub-circuit, so that we distribute the computation work to as many sub-provers as possible while keeping the overhead of aggregation small. Specifically, with $m = 128$, the sub-circuit is of size $\sim 2^{25}$ and the aggregation circuit is of size $> 2^{22}$, which means that the original circuit size is more than 2^{32}. Dinocchio generates a succinct proof of size 11.4 MB in 9.23 hours, and the verification takes less than 16 seconds.

To further demonstrate the scalability of the matrix multiplication $A \cdot B$, where $A \in \mathcal{R}_{t'}^{D \times N}$ and $B \in \mathcal{R}_{t'}^{N \times M}$, we fix $m = 128$, $N = D = 4096$, and evaluate our protocol for $M \in \{128, 256, 512, 1024, 2048, 4096, 8192\}$. This results in sub-circuits with size ranging from $\sim 2^{20}$ to $\sim 2^{26}$, and the original circuit is then of size ranging from $\sim 2^{27}$ to $\sim 2^{33}$. The runtimes of the prover side are shown in Fig. 14. Moreover, in Fig. 15, we show the prover side runtime for $N = D$ ranging from $\{4096, 8192, 16384\}$, with $m = 128$ sub-provers to balance the overhead of aggregation. [10]

The microbenchmarks of the runtimes for each encryption/encoding operation based on the implementation of Rinocchio [60] under two sets of encoding parameters for Π_1, Π_2 are shown in

[10]We set 4096 as the smallest parameter since, based on the security and correctness requirement, we have that the ring dimension D of Π_1 has to be at least 4096. To make full use of all slots in one ciphertext, we make the matrix dimension aligned with D. For larger N, we simply repeat the procedure, and the runtime increases proportionally.

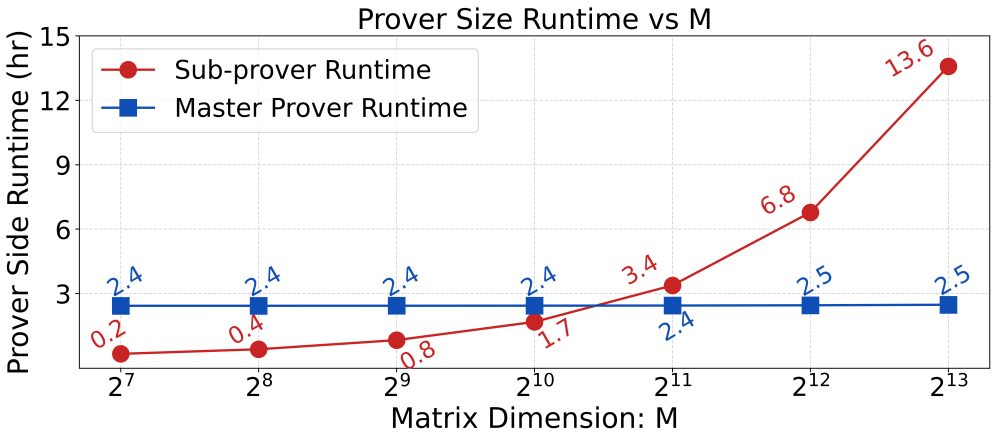

Figure 14: Prover side runtime under different matrix dimension M. We fix the other matrix dimensions to be $N = D = 4096$, and the number of sub-provers to be $m = 128$.

Fig. 16.

Comparison with prior works. When $N = D = 4096, M = 8192$ and $m = 128$, the Rinocchio circuit, i.e., without the distributed-prover setting, has $\sim 2^{33}$ constraints. The runtime of the prover thus takes more than $1,740$ hours, while our prover side only takes 16.06 hours, which is a $108\times$ improvement. Notice that we did not directly gain the full $128\times$ improvement because the master prover incurs an overhead due to aggregation, which asymptotically grows as $O(m \log m)$.

For verifiable matrix multiplication, DataSeal [57] represents one of the state-of-the-art approaches. However, it requires the user to embed additional redundancy into all inputs during the encryption phase. As a result, their scheme does not support server-side input privacy, since the server cannot contribute its own private data. On the other hand, our protocol theoretically supports such applications. At a high level, the server side, which also serves as our prover side, first commits to its private inputs via a public commitment. Each sub-circuit then embeds Merkle path proofs of the private inputs w.r.t. the public commitment. The verifier thus checks the satisfiability of the circuits by treating the commitment as a public input. We leave the non-trivial implementation of this Merkle path verification to future work.

Moreover, one recent work [59] also proposes an efficient verifiable matrix-to-*vector* multiplication protocol, by utilizing the sumcheck protocol and bridging the gap between rings and fields. We ran their benchmark with $N = D = 4096$ and 16 cores. Their prover requires 20.95 seconds. Thus, if we directly adapt their scheme for matrix multiplication by setting $M = 8192$, i.e., recursively invoking their proof protocol M times on different encrypted data vectors, their protocol produces a proof in approximately 47.67 hours, about $2.97\times$ slower than Dinocchio. The resulting proof exceeds 2GB in size, two orders of magnitude larger than ours, and the verification takes over one hour, which is also two orders of magnitude worse. Note that this comparison reflects a straightforward adaptation on [59], while optimizations to their protocol may exist for matrix multiplication, but exploring them lies beyond the scope of this evaluation.

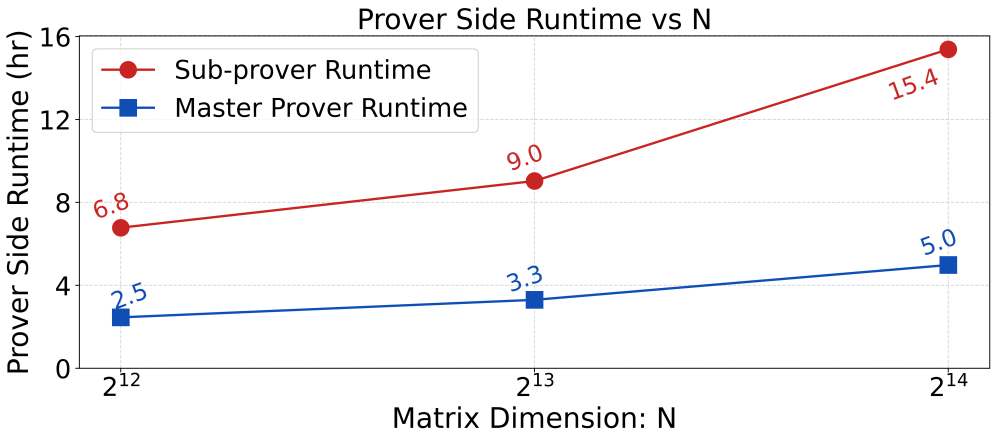

Figure 15: Prover side runtime under different matrix dimension N. Notice that the ring dimension D of Π_1 varies accordingly so that we always have $N = D$. We fix the other dimension of matrix B to be $M = 4096$, and the number of sub-provers to be $m = 64$.

	NTT (μs)	**Encoding-Encoding Add** (μs)	**Encoding-Encoding Mul** (μs)	**Dec** (μs)
$\mathbf{\Pi_1}$	53.09	3.02	11.11	207.87
$\mathbf{\Pi_2}$	1196.51	67.29	224.68	1881.36

Figure 16: The runtime of each operation under parameter sets for Π_1 and Π_2 with ring dimension $D_1 = 4096, D_2 = 16384$ and ciphertext moduli $q \approx 2^{109}$ and $Q \approx 2^{300}$ respectively.

8.2 Detailed Breakdown

To showcase the validity of our estimations, we describe the procedure of the sub-circuit construction for m sub-provers and the corresponding aggregation circuit for the FHE algorithm in Algorithm 1. We also provide concrete details for circuit components and the number of operations the prover side needs to perform based on the circuits constructed.

The ring circuit of Algorithm 1 is a data-parallel circuit. Hence, as stated in Section 4.1, each sub-prover $\mathcal{P}_i$ takes M/m (w.l.o.g., we assume that $m \mid M$) encrypted columns $\left\{\mathsf{ct}_{j'}^{(B)}\right\}_{j' \in \{(i-1)M/m+1,\ldots,iM/m\}}$ of matrix B as its unique public inputs, while $\left\{\mathsf{ct}_{j}^{(A)}\right\}_{j \in [d]}$ are shared among all sub-provers.

The main bottleneck in the sub-circuit is proving the rotation step performed on line 6 and 13 of Algorithm 1. To derive the rotated encrypted ciphertexts, we perform two steps: automorphism and key switching. The former procedure is realized in three steps. First, we decompose the original ring element into D coefficients and employ 1 multiplication gate to verify that recombining all D slots yields the correct ring element. Second, to re-index a coefficient, we multiply it with another monomial to shift it to the corresponding slot. Third, the shifted coefficients are then added up into a single *rotated* ring element. During the key switching procedure, each rotation key is oh, because originally i want to find the details about decomposition, removed! an encryption of the rotated secret key under the original secret key, decomposed w.r.t. base t', where t' is the plaintext modulus of the (F)HE scheme. Thus, key switching requires $2\log_{t'} q'$ ring multiplication gates, where q' is the ciphertext modulus of the (F)HE scheme. In total, it takes $4\sqrt{D}\,(D + \log_B q') + 1$

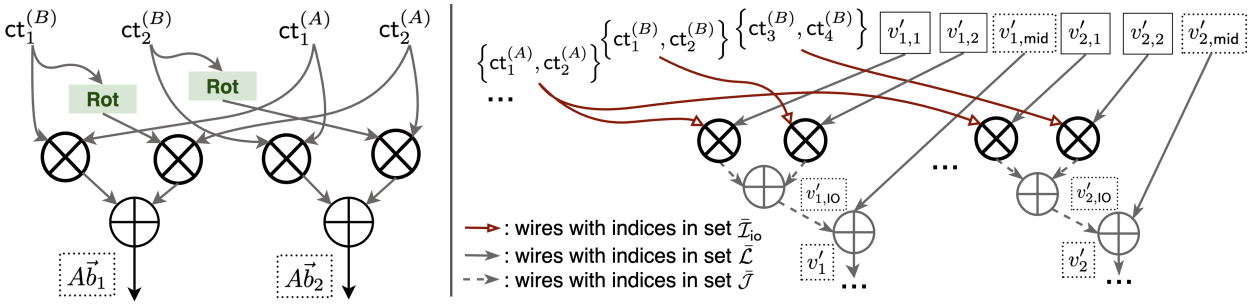

Figure 17: Illustration of the augmented circuits. The augmented sub-circuit C_1 is shown on the left and the augmented aggregation circuit is shown on the right. W.l.o.g, we assume that the computation is the multiplication between $A \in \mathcal{R}_t^{D\times D}$ and $B \in \mathcal{R}_t^{D\times 4}$. Therefore, with $m = 2$ sub-provers, each sub-prover receives two encrypted columns $\left\{\mathsf{ct}_j^{(B)}\right\}$ as inputs.

ring multiplication gates to perform the rotation steps on line 6 and 13.

The construction of the aggregation circuit follows from Definition 4.1, where we treat $\left\{\mathsf{ct}_j^{(A)}\right\}_{j\in[D]}$ as the shared public inputs as discussed in Step 3 of Fig. 5. A simple illustration is shown in Fig. 17.

Sub-circuit. We now give the detailed breakdown for each part of the sub-circuit, which is summarized in Fig. 18.

1. Each ciphertext-to-ciphertext multiplication (i.e., $\mathsf{ct}_j^{(A)} \cdot \mathsf{ct}_{j'}^{(B)}$) in the original HE evaluation is a tensor product between two ciphertexts in $\mathcal{R}_q^2$, which needs 4 multiplication gate. In total, for each sub-prover, with input $\frac{M}{m}$ vectors and matrix dimension N, we have $4\frac{MN}{m}$ multiplication gates.

2. For each encrypted vector, we need $4\sqrt{D}\,(D + \log_{t'} q') + 1$ multiplication gates to derive the rotated ciphertexts in Algorithm 1, and we have M/m such ciphertexts. Thus, in total, we need $\frac{\left(4\sqrt{D}(D+\log_{t'} q')+1\right)M}{m}$ multiplication gates. A detailed construction of rotation is demonstrated in Fig. 19. In Fig. 17, we abstract this rotation procedure as a black-box to simplify the illustration.

In total, each sub-circuit contains $|C_i| := \frac{\left(4\sqrt{D}(D+2\log_{t'} q')+4N+1\right)M}{m}$ multiplication gates, and correspondingly $3D + \frac{2M}{m} + 2\sqrt{D}\log_{t'} q' + |C_i|$ wires. Among these, $2(D + M/m)$ wires carry the public inputs $\left\{\mathsf{ct}_j^{(A)}\right\}, \left\{\mathsf{ct}_{j'}^{(B)}\right\}$ (notice that each encoding consists of two ring elements), D wires carry the monomials $\{X^\iota\}_{\iota\in[D]}$ used for the coefficient re-indexing in the rotation procedure as shown in Fig. 19, and another $2\sqrt{D}\log_{t'} q'$ wires carry the gadget decomposed rotation keys.

Each sub-prover computes $H_i(x)$ as shown in Step 3 of Fig. 10. We present an optimization that enables an efficient interpolation of $H_i(x)$. In particular, we require that $|C_i| \mid q' - 1$, where q' is the plaintext modulus of $\Pi_E^{(1)}$. This guarantees the existence of $|C_i|$-th roots of unity in the plaintext ring $\mathcal{R}_{q'}$. Notice that for C_i, we have $V_i(x), W_i(x), Y_i(x), t_i(x) \in \mathcal{R}_{q'}[x]$ to be polynomials of degree $|C_i|$ with each coefficient in $\mathcal{R}_{q'}$ as in Definition 2.2. Denote the multiplication gates in C_i by $\{\mathsf{rg}_1, \ldots, \mathsf{rg}_{|C_i|}\}$. Each sub-prover initially holds the evaluations $V_i(\mathsf{rg}_j) = c_{i,k}$ for all $j \in [|C_i|]$, where $c_{i,k}$ corresponds to the left input of the multiplication gate indexed by rg_j. Similarly for $W_i(\mathsf{rg}_j)$

	# Mul Gates	# Wires	# Pl-to-Enc Mul under $\Pi_E^{(1)}$	# Pl-to-Enc Mul under $\Pi_E^{(2)}$
Sub-circuit	$\frac{4MN}{m}+$ $\frac{\left(4\sqrt{D}\,(D+\log_B q')+1\right)M}{m}$	$3D+\frac{2M}{m}+$ $2\sqrt{D}\log_B q'+$ # Mul Gates	5·(# Mul Gates)	# Mul Gates
AC	$9Dm+12M+20m$ $6\sqrt{D}m\log_B q'$	$3D+26M+$ $34m+18Dm+$ $6\sqrt{D}m\log_B q'+$ $2\sqrt{D}\log_B q'$	-	$98m+9$# Mul Gate)

Figure 18: Breakdown for the sub-circuit and the aggregation circuit. q' is the ciphertext modulus of the underlying (F)HE scheme and B is the gadget decomposition base chosen for the rotation key, which is set to t', the plaintext modulus of (F)HE. Notice that $|\mathcal{I}_{i,\mathsf{mid}}| = |C_i|$, since all inputs of the sub-circuits are public, i.e., coming from $\mathbb{x}$. Each sub-prover needs to perform $3 \cdot |C_i|$ plaintext-to-encoding multiplications under $\Pi_E^{(1)}$ to derive $\left\{\boxed{\zeta'_{i,\mathsf{mid}}}_1\right\}$, for $\zeta \in \{v, w, y\}$, and another $2 \cdot |C_i|$ such multiplications for deriving $\boxed{h'_i}_1, \boxed{h''_i}_1$. The computation work of computing $\boxed{l_{\bar{\mathcal{L}}}}$ is also distributed to each sub-prover, s.t., each needs to perform $|C_i|$ multiplications under $\Pi_E^{(2)}$. For the aggregation circuit, we have $|\bar{\mathcal{I}}_{\mathsf{mid}}| = 14m + |C_{\mathsf{AC}}|$. Thus, the master prover performs $7 \cdot |\bar{\mathcal{I}}_{\mathsf{mid}}| + 2 \cdot |C_{\mathsf{AC}}|$ multiplications under $\Pi_E^{(2)}$, where the former term comes from deriving $\left\{\boxed{\bar{\zeta}'_{\mathsf{mid}}}_2, \boxed{\bar{\zeta}''_{\mathsf{mid}}}_2\right\}, \boxed{\bar{l}_{\bar{\mathcal{F}}}}_2$ and the latter one is due to computing $\boxed{\bar{h}'}_2, \boxed{\bar{h}''}_2$.

and $Y_i(\mathsf{rg}_j)$. To calculate the coefficients of $H_i(x) = \frac{V_i(x)W_i(x)-Y_i(x)}{t_i(x)}$, let $P_i(x) = V_i(x)W_i(x) - Y_i(x)$, the sub-prover first derives $P_i(\mathsf{rg}_j) = V_i(\mathsf{rg}_j)W_i(\mathsf{rg}_j) - Y_i(\mathsf{rg}_j)$ for all j using $|C_i|$ ring multiplications (and additions). A naive approach is to compute $H_i(\mathsf{rg}_j) = \frac{P_i(\mathsf{rg}_j)}{t_i(\mathsf{rg}_j)}$; however, since $t_i(\mathsf{rg}_j) = 0$ for all j, this approach fails. Thus, without additional structure on the evaluation points $\{\mathsf{rg}_j\}$, the prover side needs to interpolate $P_i(x)$ via Lagrange interpolation and then perform polynomial long division by $t_i(x)$, incurring a cost of $O(|C_i|^2)$ ring multiplications.

Instead, since $q' - 1$ is divisible by $|C_i|$, let $\{\xi_j\}_{j\in[|C_i|]}$ be the $|C_i|$-th root of unity, we set $\{\mathsf{rg}_j\}$ to be $\{\xi_j\}$, and choose $\{\xi'_j := \rho\xi'\}_{j\in[|C_i|]}$ to be a multiplicative coset of $\{\xi_j\}$, where $\rho \in \mathbb{Z}_{q'}$ is chosen beforehand during setup s.t. $t_i(\xi'_j) \neq 0$ for all j. Similarly, $\left\{t_i(\xi'_j)\right\}$ are also pre-computed during setup. Now, to compute the coefficients of $H_i(x)$, the sub-prover first obtains the evaluations $\left\{P_i(\xi'_j)\right\}$ based on $\{P_i(\xi_j)\}$ via one FFT, then computes $H_i(\xi'_j) = \frac{P_i(\xi'_j)}{t_i(\xi'_j)}$ since $t_i(\xi'_j) \neq 0$, and finally applies an inverse FFT to recover the coefficients of $H_i(x)$.

Now, referring to **Procedure 3** in Fig. 10, we can see that the runtime of a sub-prover consists of the following components:

1. It performs $3 \cdot |\mathcal{I}_{i,\mathsf{mid}}|$ plaintext-to-encoding multiplications under $\Pi_E^{(1)}$ to derive $\left\{\boxed{\zeta'_{i,\mathsf{mid}}}_1\right\}$ for $\zeta \in \{v, w, y\}$. Since, for matrix multiplication, all inputs to the sub-circuit multiplication gates are public with wire indices belonging to $\mathcal{I}_{i,\mathsf{io}}$, we have that $|\mathcal{I}_{i,\mathsf{mid}}| = |C_i|$.

2. To derive $H_i(x)$, the sub-prover computes first the evaluations of $H_i(\xi')$ which incurs $2|C_i|$

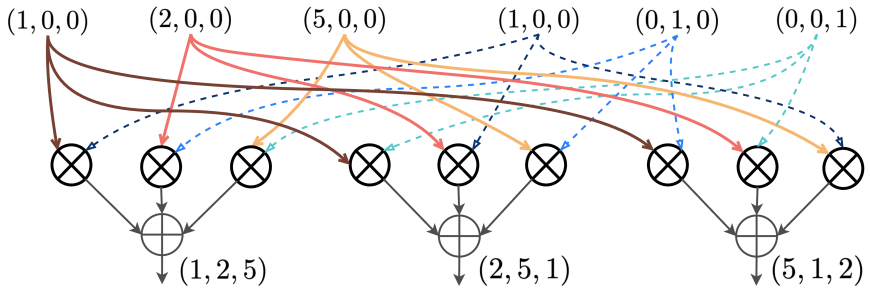

Figure 19: Toy example illustrating rotation for the ring element $(1 + 2X + 5X^2)$, represented in vector form as $(1, 2, 5)$. We denote by (a_1, a_2, a_3) the ring element $a_1 + a_2X + a_3X^2$. The monomials $1, X, X^2$ are represented as $(1, 0, 0), (0, 1, 0), (0, 0, 1)$, as shown on the top right of the graph, with dashed lines indicating the corresponding wires. On the top left, each coefficient of the original ring element is extracted as a constant polynomial, with their wires being represented in solid lines. By multiplying these constant coefficients with different monomials, the coefficients get *shifted* to different positions in the vector representation, thereby producing rotated polynomials.

ring multiplications, and then two FFTs over $\mathcal{R}_{q'}[x]$ to recover the coefficients of $H_i(x)$, where each coefficient lies in $\mathcal{R}_{q'}$ with ring dimension D and q' is the plaintext modulus of $\Pi_E^{(1)}$.

3. To compute $\boxed{h'_i}_1$ and $\boxed{h''_i}_1$, it performs $2|C_i|$ plaintext-to-encoding multiplications under $\Pi_E^{(1)}$ and $2 \cdot |C_i| \log |C_i|$ ring multiplications.

4. To prepare $\boxed{l_{i,\bar{\mathcal{L}}}}_2$, it performs $|\mathcal{I}_{i,\mathsf{mid}}| = |C_i|$ plaintext-to-encoding multiplications under $\Pi_E^{(2)}$.

To summarize, each sub-prover performs two FFTs over $\mathcal{R}_{q'}[x]$, together with $5 \cdot |C_i|$ plaintext-to-encoding multiplications under $\Pi_E^{(1)}$, $|C_i|$ plaintext-to-encoding multiplications under $\Pi_E^{(2)}$ and $2 \cdot |C_i| \log |C_i|$ standard ring multiplications.

Aggregation circuit. For the aggregation circuit, we have the following components:

1. For the evaluation of f_{AC} (Definition 4.1), notice that the public inputs and outputs of each sub-circuit consist of $\frac{M}{m}$ wires carrying $\left\{\mathsf{ct}_{j'}^{(B)}\right\}$, $\frac{M}{m}$ wires carrying the multiplication results $\left\{\mathsf{res}_{j'}\right\}$, D wires carrying $\left\{\mathsf{ct}_j^{(A)}\right\}$, D wires carrying $\{X^\iota\}_{\iota \in [D]}$, and another $\sqrt{D} \log_{t'} q'$ wires carrying the gadget decomposed rotation keys. All inputs (or outputs) except the monomials $\{X^\iota\}$ are ciphertexts, which contain two ring elements. Therefore, we need $\left(3D + \frac{4M}{m} + 2\sqrt{D} \log_{t'} q'\right) 3m$ multiplication gates to reconstruct $\boxed{v'_{i,\mathsf{io}}}_1, \boxed{w'_{i,\mathsf{io}}}_1, \boxed{y'_{i,\mathsf{io}}}_1$ for $i \in [m]$. Finally, it takes $12m$ multiplication gates to compute all "$\otimes$" operations when deriving f_{AC} (recall that each $\otimes$ operation involves 4 ring multiplications).

2. Similarly, it takes $8m$ multiplication gates to evaluate f'_{AC} (Definition 4.1).

Hence, the aggregation circuit contains $|C_{\mathsf{AC}}| := 9Dm + 12M + 20m + 6\sqrt{D}m \log_{t'} q'$ multiplication gates. When counting the total wires, notice that we have $2(D + 2M)$ ring elements for the encrypted matrices and the output results, D ring elements for the monomials $\{X^\iota\}$, and $2\sqrt{D} \log_{t'} q'$ wires for the gadget decomposed rotation keys. When reconstructing $\boxed{v'_{i,\mathsf{io}}}_1, \boxed{w'_{i,\mathsf{io}}}_1, \boxed{y'_{i,\mathsf{io}}}_1$, we also have $9Dm +$

$12M$ wires carrying the sub-polynomials $\boxed{V_{i,\kappa}}_1, \boxed{W_{i,\kappa}}_1, \boxed{Y_{i,\kappa}}_1$, for $\kappa \in \mathcal{I}_{i,\mathsf{io}}$, which only depend on the circuit structure. Lastly, each of the remaining $7m$ inputs to derive $f_{\mathsf{AC}}, f'_{\mathsf{AC}}$, h containing 2 ring elements. In total, the aggregation circuit has $3D + 16M + 14m + 9Dm + 2\sqrt{D}\log_{t'} q' + |C_{\mathsf{AC}}|$ wires.

Referring to **Procedure 4**, the runtime of the master prover is broken into the following steps:

1. It first computes the outputs of $|C_{\mathsf{AC}}|$ multiplication gates via $|C_{\mathsf{AC}}|$ ring multiplications.
2. As above, we require that $|C_{\mathsf{AC}}| \mid q-1$, where q is the plaintext modulus of $\Pi_E^{(2)}$. The coefficients of $\bar{H}(x)$ are computed via $2|C_{\mathsf{AC}}|\log|C_{\mathsf{AC}}|$ ring multiplications and two FFTs over $\mathcal{R}_q[x]$. It further performs $2|C_{\mathsf{AC}}|$ plaintext-to-encoding multiplications under $\Pi_E^{(2)}$ for $\boxed{\bar{h}'}_2, \boxed{\bar{h}''}_2$.
3. We have that $|\bar{\mathcal{I}}_{\mathsf{mid}}| = 14m + |C_{\mathsf{AC}}|$, where the term $14m$ accounts for the non-public inputs of $f_{\mathsf{AC}}, f'_{\mathsf{AC}}$, i.e., the sub-proof encodings submitted from the sub-provers. Consequently, the master prover performs $6 \cdot |\bar{\mathcal{I}}_{\mathsf{mid}}|$ plaintext-to-encoding operations under $\Pi_E^{(2)}$ to derive $\left(\boxed{\bar{\zeta}'_{\mathsf{mid}}}_2, \boxed{\bar{\zeta}''_{\mathsf{mid}}}_2\right)$ for $\zeta \in \{v, w, y\}$.
4. To compute $\boxed{\bar{l}_{\bar{\mathcal{L}}}}_2$, the master prover only needs m ring additions, and for $\boxed{\bar{l}_{\bar{\mathcal{J}}}}_2$, it performs $|\bar{\mathcal{I}}_{\mathsf{mid}}|$ plaintext-to-encoding multiplications under $\Pi_E^{(2)}$.

In total, the master prover evaluates two FFTs over $\mathcal{R}_q[x]$, together with $7 \cdot |\bar{\mathcal{I}}_{\mathsf{mid}}| + 2 \cdot |C_{\mathsf{AC}}|$ plaintext-to-encoding multiplications under $\Pi_E^{(2)}$ and $2 \cdot |C_{\mathsf{AC}}|\log|C_{\mathsf{AC}}| + |C_{\mathsf{AC}}|$ ring multiplications.

Verification. On the verifier side, the dominant cost lies in reconstructing $\bar{V}_{\mathsf{io}}(s), \bar{W}_{\mathsf{io}}(s), \bar{Y}_{\mathsf{io}}(s)$ from the public parameters in VK. Most of the computation depends solely on the circuit structure, so it can be amortized via a one-time preprocessing as discussed in Section 4.6. Consequently, the online cost of deriving $\bar{Z}_{\mathsf{io}}(s)$ for $Z \in \{V, W, Y\}$ is $3 \cdot 2(D + 2M) + 3D$ ring multiplications, where we have D monomials (each is a single ring element) used in rotations, and $D + 2M$ encodings $\{\mathsf{ct}_j^{(A)}\}_{j\in[D]}, \{\mathsf{ct}_{j'}^{(B)}, \mathsf{res}_{j'}\}_{j'\in[M]}$, each being a pair of ring elements. When executing the procedure in Fig. 12, the verifier also needs to perform 11 decryptions, 9 scalar multiplications, and 2 ring multiplications. Therefore, the total work is $O(|\mathbb{x}|)$, independent of the number of sub-provers m.

9 Conclusion

We present Dinocchio, a distributed SNARK for circuits over rings. The key technical contribution is a novel approach for aggregating sub-proofs through an aggregation circuit. Dinocchio achieves significant performance improvements, while the proof size remains constant, and the online verification time does not scale with the number of sub-provers. For a circuit with 2^{32} constraints, when distributing proof generation across $m = 128$ sub-provers, our protocol produces a succinct proof of size 11.4 MB in 9.23 hours, while verification completes in under 16 seconds.

While our protocol offers significant advantages, there are a few limitations to note. First, our evaluation of practicability is currently restricted to data-parallel circuits. For general circuits, embedding the necessary consistency proofs across sub-circuits may incur substantial overhead. Second, our base protocol, Rinocchio, treats underlying FHE ciphertexts as plaintexts of the encoding scheme during proof generation, which limits its scalability to more complex FHE circuits that involve maintenance operations and bootstrapping. Third, the fundamental linear-only assumption

underlying Rinocchio and our protocol, Dinocchio, is *not* post-quantum secure. However, post-quantum security is not necessary when the goal is to prove ring-friendly operations.

Overcoming the first two limitations is left for future work. Replacing the linear-only assumption with a post-quantum secure alternative while maintaining the same framework also remains an open question [1, 22]. Our main theoretical contribution, i.e., using an aggregation circuit and hierarchically proving the aggregation logic via a master prover, may be of independent interest. Exploring adaptations of this approach to other ring-arithmetic-friendly ZKP protocols is a promising direction for future research.

References

[1] Mohammad Sadegh Ahmadi, Taraneh Eghlidos, Behzad Abdolmaleki, and Ngoc Khanh Nguyen. A lattice-based designated verifier zksnark from standard assumptions. *Cryptology ePrint Archive*, 2025.

[2] Martin R Albrecht, Valerio Cini, Russell WF Lai, Giulio Malavolta, and Sri AravindaKrishnan Thyagarajan. Lattice-based snarks: publicly verifiable, preprocessing, and recursively composable. In *Annual International Cryptology Conference*, pages 102–132. Springer, 2022.

[3] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Cryptology ePrint Archive*, 2015.

[4] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. 2018.

[5] Diego F Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. Heliopolis: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 302–334. Springer, 2025.

[6] Shahla Atapoor, Karim Baghery, Hilder VL Pereira, and Jannik Spiessens. Verifiable fhe via lattice-based snarks. *IACR Communications in Cryptology (CiC)*, 1(1), 2024.

[7] Ward Beullens and Gregor Seiler. Labrador: compact proofs for r1cs from module-sis. In *Annual International Cryptology Conference*, pages 518–548. Springer, 2023.

[8] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, AFT '20, page 12–24, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Dan Boneh, Yuval Ishai, Amit Sahai, and David J Wu. Lattice-based snargs and their application to more efficient obfuscation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 247–277. Springer, 2017.

[10] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO 2012*. Springer-Verlag, 2012.

[11] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in LWE-based homomorphic encryption. pages 1–13, 2013.

[12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[13] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. pages 97–106, 2011.

[14] Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. Verifiable computation for approximate homomorphic encryption schemes. *Cryptology ePrint Archive*, 2025.

[15] Sylvain Chatel, Christian Knabenhans, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. Verifiable encodings for secure homomorphic analytics. *arXiv preprint arXiv:2207.14071*, 2022.

[16] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. 2018.

[17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. 2017.

[18] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33. Springer, 2016.

[20] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. 2021.

[21] Ashwini Dalvi, Apoorva Jain, Smit Moradiya, Riddhisha Nirmal, Jay Sanghavi, and Irfan Siddavatam. Securing neural networks using homomorphic encryption. In *2021 International Conference on Intelligent Technologies (CONIT)*, pages 1–7, 2021.

[22] Thomas Debris-Alazard, Pouria Fallahpour, and Damien Stehlé. Quantum oblivious lwe sampling and insecurity of standard model lattice-based snarks. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 423–434, 2024.

[23] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640. Springer, 2015.

[24] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://ia.cr/2012/144`.

[25] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://ia.cr/2012/144`.

[26] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. Thorpir: Single server pir via homomorphic thorp shuffles. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2024.

[27] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: Snarks for ring arithmetic. *Journal of Cryptology*, 36(4):41, 2023.

[28] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 626–645. Springer, 2013.

[29] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. Stanford university, 2009.

[30] Shai Halevi and Victor Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481, 2020.

[31] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 396–416, New York, NY, USA, 2023. Association for Computing Machinery.

[32] Janik Huth, Antoine Joux, and Giacomo Santato. Fherret: Proof of fhe correct-and-honest evaluation with circuit privacy from mpcith. *Cryptology ePrint Archive*, 2025.

[33] Wen jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. 2021.

[34] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. 2018.

[35] Ahmed Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. MIRAGE: Succinct arguments for randomized algorithms with applications to universal zk-SNARKs. Cryptology ePrint Archive, Paper 2020/278, 2020.

[36] Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.

[37] Joon-Woo Lee, Eunsang Lee, Young-Sik Kim, and Jong-Seon No. Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In *Advances in Cryptology – ASIACRYPT 2023*, pages 36–68. Springer Nature Singapore, 2023.

[38] Keewoo Lee and Yongdong Yeo. SophOMR: Improved oblivious message retrieval from SIMD-aware homomorphic compression. Cryptology ePrint Archive, Paper 2024/1814, 2024.

[39] Chongrong Li, Pengfei Zhu, Yun Li, Cheng Hong, Wenjie Qu, and Jiaheng Zhang. Hyperpianist: Pianist with linear-time prover and logarithmic communication cost. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3383–3401. IEEE, 2025.

[40] Shimin Li, Xin Wang, and Rui Zhang. Privacy-preserving homomorphic macs with efficient verification. In *Web Services – ICWS 2018: 25th International Conference, Held as Part of the Services Conference Federation, SCF 2018, Seattle, WA, USA, June 25-30, 2018, Proceedings*, page 100–115. Springer-Verlag, 2018.

[41] Fengrun Liu, Haofei Liang, Tianyu Zhang, Yuncong Hu, Xiang Xie, Haisheng Tan, and Yu Yu. Hasteboots: Proving fhe bootstrapping in seconds. *Cryptology ePrint Archive*, 2025.

[42] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkRollups via Fully Distributed Zero-Knowledge Proofs. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 1777–1793.

[43] Zeyu Liu, Katerina Sotiraki, Eran Tromer, and Yunhao Wang. Snake-eye resistant PKE from LWE for oblivious message retrieval and robust encryption. pages 126–156, 2025.

[44] Zeyu Liu and Eran Tromer. Oblivious message retrieval. In *Annual International Cryptology Conference*, pages 753–783. Springer, 2022.

[45] Zeyu Liu, Eran Tromer, and Yunhao Wang. Group oblivious message retrieval. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4367–4385, 2024.

[46] Zeyu Liu, Eran Tromer, and Yunhao Wang. PerfOMR: Oblivious message retrieval with reduced communication and computation. Usenix Security'24, 2024.

[47] Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7ms, with $\tilde{O}(1)$ polynomial multiplications. Cryptology ePrint Archive, Paper 2023/910, 2023.

[48] Zeyu Liu, Yunhao Wang, and Ben Fisch. IND-CPA-d of relaxed functional bootstrapping: A new attack, a general fix, and a stronger model. Cryptology ePrint Archive, Paper 2025/1627, 2025.

[49] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1057–1073. IEEE, 2021.

[50] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 2013.

[51] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. 2022.

[52] Deepika Natarajan, Andrew Loveless, Wei Dai, and Ronald Dreslinski. Chex-mix: Combining homomorphic encryption with trusted execution environments for two-party oblivious inference in the cloud. *Cryptology ePrint Archive*, 2021.

[53] Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: Fast polynomial commitments from lattices. In *Annual International Cryptology Conference*, pages 243–275. Springer, 2024.

[54] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.

[55] Kabir Peshawaria, Zeyu Liu, Ben Fisch, and Eran Tromer. Laminate: Succinct simd-friendly verifiable fhe. *Cryptology ePrint Archive*, 2025.

[56] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. Hekaton: Horizontally-scalable zksnarks via proof aggregation. *Cryptology ePrint Archive*, 2024.

[57] Muhammad Husni Santriaji, Jiaqi Xue, Yancheng Zhang, Qian Lou, and Yan Solihin. Dataseal: Ensuring the verifiability of private computation on encrypted data. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2378–2394. IEEE, 2025.

[58] Louis Tremblay Thibault and Michael Walter. Towards verifiable fhe in practice: Proving correct execution of tfhe's bootstrapping using plonky2. *Cryptology ePrint Archive*, 2024.

[59] Louis Tremblay Thibault, Michael Walter, and Jiapeng Zhang. Practical snargs for matrix multiplications over encrypted data. *Cryptology ePrint Archive*, 2026.

[60] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption. *arXiv preprint arXiv:2301.07041*, 2023.

[61] Wenhao Wang, Fangyan Shi, Dani Vilardell, and Fan Zhang. Cirrus: Performant and Accountable Distributed SNARK, 2024.

[62] Yunhao Wang and Fan Zhang. Qelect: Lattice-based single secret leader election made practical. Usenix Security 2025, 2025.

[63] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. {DIZK}: A distributed zero knowledge proof system. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 675–692, 2018.

[64] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless Cross-chain Bridges Made Practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, pages 3003–3017, New York, NY, USA, November 2022. Association for Computing Machinery.

[65] Hua Xu, Mariana Gama, Emad Heydari Beni, and Jiayi Kang. Frittata: Distributed proof generation of fri-based snarks. *Cryptology ePrint Archive*, 2025.

[66] Jiping Yu, Kun Chen, Xiaoyu Fan, Yunyi Chen, Xiaowei Zhu, and Wenguang Chen. Lodia: Towards optimal sparse matrix-vector multiplication for batched fully homomorphic encryption. *Cryptology ePrint Archive*, 2025.

[67] Jiping Yu, Kun Chen, Xiaoyu Fan, Yunyi Chen, Xiaowei Zhu, and Wenguang Chen. Lodia: Towards optimal sparse matrix-vector multiplication for batched fully homomorphic encryption. Cryptology ePrint Archive, Paper 2025/1425, 2025.

[68] Qiao Zhang, Chunsheng Xin, and Hongyi Wu. Gala: Greedy computation for linear algebra in privacy-preserved neural networks. *arXiv preprint arXiv:2105.01827*, 2021.

[69] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, and Xianhui Lu. Fhe-snark vs. snark-fhe: From analysis to practical verifiable computation. *Cryptology ePrint Archive*, 2025.

[70] Xiaopeng Zheng, Hongbo Li, and Dingkang Wang. A new framework for fast homomorphic matrix multiplication. *Cryptology ePrint Archive*, 2023.

[71] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. Optimal single-server private information retrieval. pages 395–425, 2023.

[72] Zhelei Zhou, Yun Li, Yuchen Wang, Zhaomin Yang, Bingsheng Zhang, Cheng Hong, Tao Wei, and Wenguang Chen. Zhe: Efficient zero-knowledge proofs for he evaluations. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 3328–3346. IEEE, 2025.