Lasagne: Practical Verifiable Computation over Encrypted Data

Xinxuan Zhang[1,2], Ruida Wang*[1,2], Qingyun Niu[1,2], Peixin Liu[3], Xianhui Lu[1,2], Lutan Zhao[1], Rui Hou[1], and Yi Deng[4]

[1]State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS
[2]School of Cyber Security, University of Chinese Academy of Sciences
[3]School of Microelectronics, Southern University of Science and Technology
[4]Xidian University

Abstract

Verifiable Computation on Encrypted Data (VCoED) addresses the computational integrity gap in Fully Homomorphic Encryption (FHE). While recent protocols have made significant strides in making VCoED feasible, server-side proof generation remains computationally intensive, often requiring hours for a modest 2^{20}-gate payload circuit (e.g., 2.27 hours for Phalanx, 9.26 hours for Blind Fractal). Moreover, most existing schemes lack support for payload circuits that are homomorphically executed with SIMD operations.

In this work, we present *Lasagne*, a new efficient VCoED scheme for BGV/BFV-type FHE schemes. Lasagne offers the following:

1. It supports multiplicative layered payload circuits and allows them to be homomorphically evaluated under SIMD message encoding, which aligns with the requirements of practical FHE deployment.
2. It achieves efficient prover time while maintaining acceptable communication/verification overhead, and supports flexible parameters choosing to trade off prover time and communication overhead.

For a 16-layer, 2^{20}-gate payload circuit, Lasagne generates a 30 MB proof in just 6–12 minutes(single core), achieving an 11×–23× speedup over Phalanx (2.27 hours, 61.4 MB). When the payload natively supports SIMD execution (2^4 slots), the proving time further reduces to 4–5 minutes (a 27×–34× speedup). We instantiate *Lasagne* for a face recognition application built on FHE. Empirical results confirm the validity of our time estimates and the practical viability of the scheme.

*Corresponding author: wangruida@iie.ac.cn

1 Introduction

Fully homomorphic encryption (FHE) stands as an advanced cryptographic primitive, enabling general-purpose computation directly on encrypted data. Since Gentry provided the first construction of FHE in 2009 [18], the field has developed rapidly over the past two decades, giving rise to numerous efficient schemes such as B/FV [5, 13], BGV [6], TFHE [11], FHEW [12], and CKKS [10]. These schemes have begun to find applications in settings like PSI [25, 9], PIR [27], and beyond [21, 22].

However, FHE do not guarantee the integrity of computation. In a typical outsourced scenario, a server faces homomorphic evaluation overheads that are over three orders of magnitude higher than computing in the plaintext [26]. This creates an incentive for a malicious—or economically driven "lazy"—server to cheat by returning a forged result. Without an integrity verification mechanism, the client would remain unaware of such deviations.

Verifiable Computation over Encrypted Data. To address this issue, Gennaro, Gentry, and Parno[17] introduced the concept of Verifiable Computation over Encrypted Data (VCoED), allowing a client to outsource its computation to an untrusted server while protecting the data privacy and enabling the client to verify computation correctness. The initial paradigm proposed alongside this concept requires the server to generate a zero knowledge proof that attests to the correct execution of the homomorphic evaluation. While theoretically sound, this paradigm suffers from severe performance degradation. This inefficiency stems from a mismatch between the ring arithmetic of FHE ciphertexts with the large prime fields supported by mainstream SNARK systems [31].

FHE-SNARK Paradigm to Construct VCoED. Recently, Garg, Goel, and Wang proposed a new paradigm for VCoED to bypass this barrier [16]. Informally, to prove that an evaluation $c_{\text{out}} = \mathsf{Eval}(f, c_{\text{in}})$ is performed correctly, the server homomorphically computes the SNARK proof for the payload circuit f over the encrypted inputs. Due to the homomorphic property, this process yields an encrypted proof, which the client can decrypt it and verify the computational integrity. This paradigm offers a crucial structural advantage: the arithmetic domain of the SNARK scheme perfectly matches the plaintext space of the FHE scheme. Furthermore, a substantial portion of the public proof generation procedures, such as the cryptographic hash functions utilized in the Fiat-Shamir transform, can be executed outside the homomorphic evaluation. Consequently, this FHE-SNARK paradigm represents a promising approach toward practical verifiable computation over encrypted data.

While multiple constructions under this paradigm have led to rapid improvements, existing schemes remain far from practical. Specifically, they suffer from two fundamental and intertwined bottlenecks: prohibitive proving times and incompatibility with SIMD-optimized payload circuits.

The Arithmetization Dilemma: Blind Fractal and Phalanx. Blind Fractal and Phalanx provided the first concrete FHE-SNARK instantiations for VCoED, operating over general arithmetic circuits. However, their proving

times are highly impractical—requiring 9.26 hours and 2.27 hours, respectively, to prove a 2^{20}-gate circuit. More critically, while their underlying scheme relies on BFV, the R1CS constraint system [3] is incompatible with the SIMD optimizations typically used for the BFV *payload circuit*, which are ubiquitous in real-world FHE applications. To generate a proof, these schemes require the intermediate ciphertexts from the payload evaluation to be non-packed or packed into specific formats. If the payload circuit is evaluated utilizing SIMD parallelism, aligning the data with the prover requires a massive extraction and repacking step. This forces a bitter dilemma: either abandon SIMD optimization during the payload execution (drastically degrading FHE throughput) or incur astronomical preprocessing overheads before the proof generation.

Aligning Proofs with SIMD: The Laminate Schemes. To address the structural mismatch causing by the R1CS, Laminate (the initial version[1]) leverages a GKR-based proof system, achieves minimal proof size and support for SIMD execution of the layered payload circuit.

However, besides the mismatch between layered circuits and BGV/BFV-type FHE applications (which are typically multiplicative layered circuits), it fails to efficiently support flexible packing strategies. The proof circuit has to be run with the same SIMD width (*In this context, the SIMD width refers to the number of data slots packed within a single ciphertext for parallel computation.*) of payload circuit, making it inefficiency for payload circuit with small SIMD width. Specifically, for k-parallel homomorphic executions (under SIMD optimizations) of a single 2^{20}-sized circuit, the absolute proving time (even when $k = 1$) reaches as high as 30 to 70 hours, rendering it deeply impractical for real-world deployment.

Motivated by the limitations of existing methods, we arrive at the central question of this work:

Can we construct a proof-efficient VCoED scheme that supports proving homomorphic executions of arbitrary SIMD-width circuits, while achieving minute-level proving time for million-scale circuits?

1.1 Our result

In this work, we present *Lasagne*, a new efficient VCoED scheme for BGV/BFV-type FHE schemes. Lasagne offers the following:

1. It supports multiplicative layered payload circuits and allows them to be homomorphically evaluated under SIMD message encoding, which aligns with the requirements of practical FHE deployment.

2. It achieves efficient prover time while maintaining acceptable communication/verification overhead, and supports flexible parameters choosing to

[1]Laminate's latest update (dated April 10), which is concurrent with ours, reduces proof time and partially addresses the issues of the circuit mismatch and the requirement that SIMD widths be exactly identical. We detail the differences in Section 1.3.

trade off prover time and communication overhead.

For a 16-layer, 2^{20}-gate payload circuit, Lasagne generates a 30 MB proof in just 6–12 minutes (single core), achieving an 11×–23× speedup over Phalanx (2.27 hours, 61.4 MB). When the payload natively supports SIMD execution (2^4 slots), the proving time further reduces to 4–5 minutes (a 27×–34× speedup).

To demonstrate practical viability, we instantiate Lasagne for secure face matching, delivering **the first concrete application implementation for VCoED**. For a batched evaluation over 1,000 encrypted facial templates—yielding a circuit of approximately 2-layer and 2^{20} gates—Lasagne generates a 4.25 MB proof in just 26.96 seconds. Client verification is highly efficient, completing in 261.82 milliseconds. These empirical results not only validate our theoretical estimates but also demonstrate that Lasagne establishes a practical foundation for a broader class of verifiable FHE applications.

1.2 Related work

SNARK-FHE is another paradigm to achieve VCoED scheme, where the server directly use SNARK to prove the integrity of homomorphic evaluation.

Recent Advancements in SNARK-FHE. Early efforts such as Rinocchio [15] extend SNARKs from finite-field arithmetic to ring arithmetic by introducing Quadratic Ring Programs, thereby better matching the algebraic structure of RLWE-based ciphertext computations. However, such ring-based formulations mainly capture algebraic ciphertext operations and still incur substantial overhead when expressing non-algebraic maintenance operations such as modulus switching and key switching. Subsequent work on verifiable FHE via lattice-based SNARKs [2] exploits the double-CRT/RNS representation of modern FHE schemes to decompose ring computations into field-level proof instances, and further supports key switching and modulus switching, providing one of the first practical baselines for proving multi-multiplication homomorphic evaluations. Another line of work focuses on proving bootstrapping. Thibault and Walter [30] demonstrate the first practical SNARK proof for TFHE programmable bootstrapping by arithmetizing the bootstrapping procedure and leveraging recursive proving in plonky2, reducing bootstrapping verification to a succinct proof but still requiring tens of minutes of proving time. HasteBoots [23] further tailors the proof system to FHEW/TFHE NAND with bootstrapping by designing specialized PIOPs for FHE-specific relations such as NTT, accumulator updates, and modulus switching, reducing proof generation to seconds at the cost of larger proofs and higher verification overhead. For RLWE schemes with SIMD batching and approximate arithmetic, recent work on verifiable computation for approximate HE [7] designs PIOPs and polynomial commitments over R_q, enabling native ciphertext arithmetic and efficient handling of CKKS maintenance operations such as rescaling, key switching, and modulus switching. In parallel, ZHE [32] adopts a VOLE-ZKP approach and introduces efficient proofs for modulo operations and NTT/INTT, achieving substantially lower

prover overhead for BGV and CKKS evaluations while supporting server-side private inputs.

Consequently, while SNARK-FHE provides uncompromising security guarantees equivalent to 2PC, the computational overhead induced by the algebraic mismatch renders it impractical for evaluating large-scale outsourced computations. This inherent limitation serves as the primary motivation for exploring and optimizing the FHE-SNARK paradigm in this work.

1.3 Concurrent Work

This work follows an earlier version of Laminate and is concurrent with its current version (updated on April 10).

Earlier Laminate. As discussed in the introduction, the earlier Laminate suffers from two drawbacks: it mismatches the circuit types used in practical FHE deployments, and its proof circuit must duplicate the payload circuit's SIMD width. Our scheme addresses both issues: it supports multiplication-layered circuits (the natural fit for FHE), and it allows the proof circuit to utilize full FHE slots even when the payload circuit uses fewer. Consequently, our scheme achieves better proving time across all SIMD widths. In particular, for a million-gate non-SIMD payload circuit, our proof takes only 6–12 minutes, whereas the earlier Laminate requires over 37-75 hours. Although our scheme requires a larger proof size (tens of megabytes vs. 0.1–0.2 MB), it remains practical.

Current Laminate. Their new schemes reduces proof time and partially addresses above issues: They replace the layered circuit with a special one consisting of alternating multiplication layer and affine layer, and propose a new scheme called $\text{Laminate}_{\text{laned}}$, which supports efficient proving for payload circuits with small SIMD width, though such circuits cannot contain rotation gates.

Compared with the current version of Laminate, our scheme still offers significant advantages in proving efficiency for many parameter choices. For instance, when proving k-parallel executions of a single 2^{20}-gate payload circuit, our scheme outperforms $\text{Laminate}_{\text{laned}}$ for $k \leq 2^{10}$. In the practically important case of $k = 1$ (i.e., a single non-SIMD circuit), our proof completes in just 6–12 minutes, which is 5–10$\times$ faster than the over-an-hour required by $\text{Laminate}_{\text{laned}}$. Furthermore, our scheme provides two additional advantages: (1) it supports flexible multiplication-layered circuits, and (2) it accommodates payload circuits that contain rotation gates.

Technique Differences. The new layered circuit in the current Laminate is close to our multiplication-layered strategy, however, the core technical differences between our scheme and the current version of Laminate is as follows:

- Layered strategy: Their approach separates multiplication gates into dedicated layers and compresses the additions between them into independent affine layers (so that multiplication layers contain only multiplication gates). In contrast, our scheme compresses the addition layers directly into the multiplication layers.

- Communication overhead reduction: Laminate adds an outer SNARK to reduce proof size. This drastically cuts communication ($< 0.2MB$) but greatly increases proving overhead, especially for payload circuits with small SIMD width. In contrast, we reduce both the number of sumcheck rounds and per-round communication, keeping proof size in the tens of megabytes while achieving much faster proof generation.

- Packing strategy: We adopt a different packing strategy that allows our scheme to support payload circuits with rotation gates under SIMD execution. Notably, we believe our packing strategy can be applied to their scheme to overcome this limitation.

2 Technical Overview

The Blind Fractal and Phalanx schemes start from SNARKs for R1CS instances ($Az \circ Bz = Cz$, where z is the statement/witness vector, while A, B, C are public sparse matrices) and adapt them into HE-friendly versions. However, these schemes are not well suited for VCoED, because computation and packing Az, Bz, Cz is expensive and, due to matrices' sparsity, unfriendly to SIMD-executed payload circuits (where z is a packed ciphertext). Recently, the Laminate scheme pointed out that the GKR protocol for layered circuits offers better compatibility. Nevertheless, directly using a HE version of GKR still faces challenges in circuit model compatibility and communication overhead.

Multiplication-Layered Circuit. Traditional GKR proof systems require circuits to be strictly layered, meaning all gates topologically synchronized level by level. This rigid synchronization introduces *structural mismatches* and inefficiencies when applied to VCoED. Since in BGV-like FHE schemes multiplication mainly dictates the noise growth; consequently, practical algorithms usually minimize multiplication depth without restricting addition depth.

To bridge this gap, we introduce *multiplication-layered circuits*, a tailored abstraction that relaxes synchronization constraints on addition gates. Specifically, we allow a gate in a layer to take input from *addition* gates in the same layer. It allows us to treat *addition* gates as flexible connectors within the same multiplicative level, and avoid depth inflation. Thus, layering constraints can apply solely to critical paths through *multiplication* gates, aligning the circuit topology with the natural multiplicative depth of FHE evaluation.

We generalize the traditional GKR protocol to multiplication-layered circuits. As in standard GKR, we encode the output wires of each layer as a multilinear polynomial, and represent this polynomial as a sum about the encoding polynomials of the next layer's output (which serve as the input to the current layer). This sum formulation allows us to reduce a query on one layer's encoding polynomial to a query on the next layer using the sumcheck protocol, exactly as in GKR. Repeating this reduction recursively completes the entire proof.

Two Challenges on homomorphically running GKR protocol. Running a homomorphic version of our GKR proof system to prove the correct evaluation of a payload circuit presents two main challenges. First, unlike Phalanx, which invokes sumcheck only twice in total, our GKR protocol requires two sequential invocations of sumcheck *per layer*, leading to a significant increase in communication overhead. Second, we need a packing strategy that enables our homomorphic GKR protocol to fully leverage the SIMD capabilities of the underlying FHE scheme, even when the payload circuit has only a small SIMD width.

Flexible Sumcheck & Multivariate Constant-Degree Polynomial Encoding. To reduce communication overhead in the homomorphic version of sumcheck protocol, we first adopt two straightforward techniques:

- Ciphertext packing: repack the encrypted messages to be sent so that only a single packed ciphertext is transmitted per round.
- Early termination: let the sumcheck protocol terminate early, and the verifier completes the remaining part in plaintext. This does not significantly increase the verifier's workload, as the extra plaintext computation is outweighed by the decryption savings from reduced communication.

Despite these techniques, proof size can still reach hundreds of megabytes for large and high-depth circuits. Since sumcheck now sends only one packed ciphertext per round, the only remaining way to further reduce communication is to cut the number of rounds—naturally leading us to consider round compression.

A straightforward method is to compress multiple rounds into one, for example by eliminating two variables per round so that every two rounds merge into a single round. In such a compressed round, the prover sends a bivariate polynomial of in the form of $\sum_x g(r, k_0, k_1, x)$ instead of a univariate one, and the verifier responds with two random challenges r_0, r_1. However, this significantly increases the prover's per-round overhead: whereas a univariate degree-k polynomial requires $k+1$ evaluation points to be uniquely determined, a bivariate polynomial of degree k in each variable requires $(k+1)^2$ points.

The above overhead analysis raises a fundamental question: why do SNARKs need sumcheck at all? Recall that a SNARK encodes statement and witness as the values of multilinear polynomials over $\{0,1\}^n$, and then the correctness of instances reduced to relations (including sums) among these polynomials. This suggests a different direction: *Can we encode the witness as values of a degree-$(d-1)$ multivariate polynomial over $[d]^{n/\log_2 d}$?*

The answer is yes. Multilinear extension generalizes to arbitrary multivariate polynomials: values over $[d]^n$ uniquely determine a degree-$(d-1)$ polynomial via Lagrange interpolation. We modify our GKR-style protocol and encode each layer's wire values into such polynomials, reducing the number of variables by a factor of $\log_2 d$. We then extend sumcheck to support summation over $[d]^n$ and call this variant *Flexible Sumcheck*. In fact, this framework can be further extended to non-uniform polynomials.

This generalization directly reduces the number of sumcheck rounds. For example, by setting $d = 4$, we can halve the number of rounds, while the degree

of the polynomial being summed increases by a factor of 3. Consequently, the prover only needs to compute $3k + 1$ evaluation points, a trade-off that circumvents the quadratic blowup of the naive round-compression approach.

This approach also offers an additional benefit. Since the first sumcheck round operates largely in the base field while subsequent rounds require expensive extension field arithmetic, shifting more work to the first round reduces the cost in the extension field for later rounds, partially offsetting the overhead from the increased number of evaluation points.

Packing Strategy for SIMD-evaluated payload circuit. To minimize proof overhead, our proof circuit utilizes the full FHE slot capacity. However, the SIMD execution of the payload circuit may only have a smaller SIMD width or produce packed ciphertexts with an incompatible structure.

To address this, we propose a packing strategy that simultaneously adjusts both the payload circuit and the proof circuit, allowing the ciphertexts required by the proof circuit to be directly packed from the intermediate ciphertexts of the payload circuit without any extraction. The core techniques are as follows:

- For a payload circuit that originally uses d_{pslot} slots, we migrate it to run on ciphertexts with d_{slot} slots (where $d_{\text{pslot}} \mid d_{\text{slot}}$). Its original packed ciphertext of $(a_0, a_1, \ldots, a_{d_{\text{pslot}}-1})$ is replaced by the following form:

$$\text{Enc}\big(pk,\ (\underbrace{a_0, a_0, \ldots, a_0}_{d_{\text{slot}}/d_{\text{pslot}} \text{ times}},\quad \underbrace{a_1, \ldots, a_1}_{d_{\text{slot}}/d_{\text{pslot}} \text{ times}}\quad ,\ \ldots,\ a_{d_{\text{pslot}}-1})\big).$$

  This encoding still supports the SIMD execution: we run the corresponding SIMD gates and replace each rotation $\text{rot}(ct, j)$ with $\text{rot}(ct,\ j \cdot d_{\text{slot}}/d_{\text{pslot}})$.

- We adjust the wire index assignment in the GKR scheme so that each packed ciphertext with d_{slot} slots required by the proof protocol can be obtained from $d_{\text{slot}}/d_{\text{pslot}}$ ciphertexts of the above form using only plaintext–ciphertext multiplications and additions.

Key Optimization for Packing. The packing phase requires ciphertext multiplications, dominated by NTT transforms. Thanks to our multiplication-layered circuit design, a considerable number of the intermediate ciphertexts needed for packing are exactly the inputs to the payload circuit's multiplication gates. During homomorphic evaluation, these inputs are already in the NTT domain; the prover can directly reuse them, bypassing redundant NTT transformations and significantly reducing packing overhead.

3 Preliminaries

Notions. Let λ denote the security parameter. For any positive integer m, we denote by $[m]$ the set $\{0, 1, \cdots, m-1\}$.We use the standard abbreviation PPT to denote probabilistic polynomial time. In this paper, we frequently encounter operations of the form $a \cdot b$ with $a \in \mathbb{F}_p$ and $b \in \mathbb{F}_{p^R}$, or evaluations $f(r)$ where

f is a polynomial over $\mathbb{F}_p$ and $r \in \mathbb{F}_{p^R}$. In such cases, we implicitly treat a (or the polynomial f) as an element (or polynomial) over the extension field $\mathbb{F}_{p^R}$ via the natural embedding.

3.1 Proof Systems

Proof systems serve as a fundamental building block in this paper, ensuring the correctness of computations. Some portions of these definitions are taken verbatim from [8, 31].

Indexed Relation. An indexed relation $\mathcal{R}$ is a set of triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i}$ is the index, $\mathtt{x}$ the instance, and $\mathtt{w}$ the witness; the corresponding indexed language $L(\mathcal{R})$ is the set of pairs $(\mathtt{i}, \mathtt{x})$ for which there exists a witness $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$.

Arithmetic Circuit Index Relation. The indexed relation $\mathcal{R}_{arit}$ of arithmetic circuits satisfiability consists of triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i}$ is the description of an arithmetic circuit f, $\mathtt{x} = (x, y)$ is an assignment to some of the input wires x and all of the output wires y, and $\mathtt{w}$ is an assignment to the remaining input wires w that makes $f(x, w) = y$.

Definition 1 (Interactive Proof) *Let $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be an interactive protocol described as follows:*

- $\mathsf{Setup}(1^\lambda)$*: On input a security parameter 1^λ, Setup outputs a common reference string crs.*
- $\mathcal{I}(\mathsf{crs}, \mathtt{i})$*: On input crs and the index $\mathtt{i}$, the* ***deterministic*** *algorithm $\mathcal{I}$ outputs the verifier and prover parameters $(\mathsf{vp}, \mathsf{pp})$.*
- $\langle \mathcal{P}(\mathsf{pp}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathsf{vp}, \mathtt{x}) \rangle$*: $\langle \mathcal{P}, \mathcal{V} \rangle$ is a protocol between the prover and verifier, where the prover wants to show that $(\mathtt{i}, \mathtt{x}; \mathtt{w}) \in \mathcal{R}$ and the verifier outputs a bit indicating acceptance or not. We denote the output of the verifier by $0/1 \leftarrow \langle \mathcal{P}(\mathsf{pp}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathsf{vp}, \mathtt{x}) \rangle$.*

Such an interactive protocol is called a proof system for indexed relation $\mathcal{R}$ if the following properties hold:

- *Completeness: for all $(\mathtt{i}, \mathtt{x}; \mathtt{w}) \in \mathcal{R}$,*

$$\Pr\left[\langle \mathcal{P}(\mathsf{pp}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathsf{vp}, \mathtt{x}) \rangle = 1 \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{crs}, \mathtt{i}) \end{array}\right] = 1$$

- *Soundness: For every pair of unbounded adversary algorithm $(\mathcal{A}_1, \mathcal{A}_2)$, the following probability is negligible:*

$$\Pr\left[\langle \mathcal{A}_2(\mathsf{pp}, \mathtt{x}, st), \mathcal{V}(\mathsf{vp}, \mathtt{x}) \rangle = 1 \wedge (\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R}) \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathtt{i}, \mathtt{x}, st) \leftarrow \mathcal{A}_1(\mathsf{crs}) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{crs}, \mathtt{i}) \end{array}\right].$$

Furthermore, an argument system is defined similarly to a proof system, except that it is only required to be sound against PPT adversaries. An interactive protocol is called *public-coin* if all messages sent by the verifier are uniformly random elements from some subset of the plaintext space, independent of the current partial transcript. If a proof (argument) system Π involves only one message in its online phase $\langle \mathcal{P}, \mathcal{V} \rangle$, specifically, the prover sends a proof $\pi \leftarrow \mathcal{P}(\cdot)$ to the verifier, who then verifies the proof using $\mathcal{V}$, we call it a non-interactive proof (argument) system.

Definition 2 (Knowledge Soundness) *A proof system (or argument system) is called a proof of knowledge (or argument of knowledge, respectively) if it satisfies the knowledge soundness defined as follows: There exists a probabilistic polynomial time oracle machine $\mathcal{E}$, called the extractor, such that for any pair of unbounded (or probabilistic polynomial-time, respectively) adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, the following probability is negligible:*

$$\Pr\left[\begin{array}{l|r} \langle \mathcal{A}_2(pp, \mathtt{x}, st), \mathcal{V}(vp, \mathtt{x}) \rangle = 1 \wedge & crs \leftarrow Setup(1^\lambda) \\ (\mathtt{i}, \mathtt{x}, \mathtt{w}) \notin \mathcal{R} & (\mathtt{i}, \mathtt{x}, st) \leftarrow \mathcal{A}_1(crs) \\ & (vp, pp) \leftarrow \mathcal{I}(crs, \mathtt{i}) \\ & \mathtt{w} \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}(crs, \mathtt{i}, \mathtt{x}) \end{array}\right]$$

Definition 3 (Succinct Non-interactive Arguments of Knowledge) *A non-interactive argument of knowledge is considered succinct if both the verification time and communication size are sublinear in the size of the index and instance.*

Succinct Non-interactive Arguments of Knowledge (SNARKs) can be constructed from various information-theoretic proof systems that give the verifier oracle access to prover messages. These information-theoretic proof systems can be compiled using various cryptographic tools, such as hash functions and polynomial commitments. A typical information-theoretic proof system is the polynomial interactive oracle proof.

SNARKs for Encrypted Instances. In this paper, we need to prove an instance in which the statement and witness might be encrypted. This SNARK scheme can be formalized as **SNARKs for encrypted instances**[16, 1, 14], where only the verifier possesses the secret key for the encryptions and the prover only knows the public key. This scheme satisfies similar completeness and computational binding properties with only trivial syntactic changes. Additionally, the knowledge soundness properties are modified to allow the extractor access to the secret key (or a decryption oracle).

3.2 BGV Leveled Homomorphic Encryption

The BGV leveled homomorphic encryption scheme was introduced by Brakerski, Gentry, and Vaikuntanathan [6] based on the (Ring) Learning With Errors (LWE) assumption.

Given a polynomial $y \in \mathcal{R}_t = \mathbb{Z}_t[X]/(X^N+1)$, the BGV scheme encrypts it into a ciphertext consisting of two polynomials, where each polynomial is from a larger cyclotomic ring $\mathcal{R}_Q = \mathbb{Z}_Q[X]/(X^N+1)$ for some $Q > t$. We refer to t as the plaintext modulus, Q as the ciphertext modulus, and N as the ring dimension. t satisfies that $t \equiv 1 \pmod{2N}$, where N is a power of two. [2]

Plaintext encoding. In practice, instead of having a polynomial in $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N+1)$ to encrypt, applications usually have a vector of messages $\vec{m} = (m_1, \ldots, m_N) \in \mathbb{Z}_t^N$. Thus, to encrypt such input messages, BGV first encodes it by constructing another polynomial $y(X) = \sum_{i\in[N]} y_i X^{i-1}$ where $m_i = y(\zeta_j)$, $\zeta_j := \zeta^{3^j} \pmod{t}$, and ζ is the $2N$-th primitive root of unity of t. Such encoding can be done using an Inverse Number Theoretic Transformation (INTT), which is a linear transformation represented as matrix multiplication.

Encryption and decryption. The BGV ciphertext encrypting $\vec{m}$ under $\mathsf{sk} \leftarrow \mathcal{D}$ has the following format: $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$, which satisfies $b - a \cdot \mathsf{sk} = y + t \cdot e$ where y is the polynomial encoded in the way above, and e is a small error term sampled from a Gaussian distribution over $\mathcal{R}_Q$ with some constant standard deviation.

Symmetric key encryption can be done by simply sampling a random a and constructing b accordingly using sk. Public key encryption can also be achieved easily but it is not relevant to our paper so we refer the readers to prior works (e.g., [6]) for details.

Decryption is thus calculating $y' \leftarrow (b - a \cdot \mathsf{sk} \pmod{Q}) \pmod{t} \in \mathcal{R}_t$ (note that $(b - a \cdot \mathsf{sk})$ is first computed over $\mathcal{R}_Q$), and then decodes it by applying a procedure to revert the encoding process (which is also a linear transformation). For simplicity, we assume $\mathsf{BGV.Dec}$ also embeds the decoding procedure and thus outputs plaintext $y' \in \mathbb{Z}_t^N$ in the decoded form directly (instead of a polynomial $y \in \mathcal{R}_t$). Similarly, we assume $\mathsf{BGV.Enc}$ contains the encoding process, thus taking a plaintext $y' \in \mathbb{Z}_t^N$. In addition, define $\mathsf{PartialDec}(\mathsf{sk}, \mathsf{ct} = (a, b) \in \mathcal{R}_Q^2) := b - a \cdot \mathsf{sk} \in \mathcal{R}_Q$ (i.e., decryption without performing the modulo t reduction).

BGV operations. BGV essentially supports addition, multiplication, rotation, and polynomial function evaluation, satisfying the following properties:

- (Addition) $\mathsf{BGV.Dec}(\mathsf{ct}_1 + \mathsf{ct}_2) = \mathsf{BGV.Dec}(\mathsf{ct}_1) + \mathsf{BGV.Dec}(\mathsf{ct}_2)$
- (Multiplication) $\mathsf{BGV.Dec}(\mathsf{ct}_1 \times \mathsf{ct}_2) = \mathsf{BGV.Dec}(\mathsf{ct}_1) \times \mathsf{BGV.Dec}(\mathsf{ct}_2)$
- (Rotation) $\mathsf{BGV.Dec}(\mathsf{rot}(\mathsf{ct}, j))[i] = \mathsf{BGV.Dec}(\mathsf{ct})[i + j \pmod{N}]$, $\forall i, j \in [N]$
- (Polynomial evaluation) $\mathsf{BGV.Dec}(\mathsf{BGV.Eval}(\mathsf{ct}, f)) = f(\mathsf{BGV.Dec}(\mathsf{ct}))$, where $f : \mathbb{Z}_t \to \mathbb{Z}_t$ is a polynomial function. Note that this is implied by addition and multiplication.

[2] Note that this relationship between t and N does not need to be satisfied in general (e.g., see [20, 19] for the general encoding). However, throughout our paper, we suppose it holds to maximize the concrete efficiency and thus introduce it this way for simplicity.

- (Vector-matrix multiplication) $\mathsf{BGV.Dec}(\mathsf{ct} \times A) = \mathsf{BGV.Dec}(\mathsf{ct}) \times A$, where $A \in \mathbb{Z}_t^{N \times D}$ for any $D > 0$.

All operations are operated over the entire plaintext vector $\vec{m} \in \mathbb{Z}_t^N$ (element-wise). Thus, all messages need to be evaluated using the same polynomial f by default. This is also known as the Single Instruction Multiple Data (SIMD) property of BGV. Note that vector-matrix multiplication can be realized using scalar multiplication (implied by addition) and rotation. All of these BGV operations are used as blackboxes in our main constructions and we refer the readers to [6, 20, 19] to see how these operations are accomplished in detail. In this paper, we sometimes directly refer to the interfaces (e.g., Dec) for short without the BGV prefix (e.g., $\mathsf{BGV.Dec}$).

3.3 Verifiable Computation over Encrypted Data

A verifiable computation over encrypted data (VCoED) scheme allows a client to outsource its computation to an untrusted server while protecting data privacy and enabling the client to verify computation correctness. We adapt the definition from [31] and formalize it as follows:

Definition 4 ***(VCoED).*** *A verifiable computation over encrypted data scheme* VC *consists of four polynomial-time algorithms:*

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, C)$*: given a circuit* C*, the setup algorithm outputs a key pair* $(\mathsf{sk}, \mathsf{pk})$*.*
- $(\mathsf{Enc}(x), \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, x)$*: the encoding algorithm takes an input* x *and public key* pk*, and outputs an encoded value* $\mathsf{Enc}(x)$ *and a state* st*.*
- $(\mathsf{Enc}(\mathsf{out}), \pi) \leftarrow \mathsf{VC.Compute}(\mathsf{pk}, \mathsf{Enc}(x), C, y)$*: the computation algorithm takes the public key* pk*, the encoded input* $\mathsf{Enc}(x)$*, the circuit* C*, and the server's input* y*, and outputs an encoded computation result* $\mathsf{Enc}(\mathsf{out})$ *and a proof* π*.*
- $\mathsf{out}/\ \perp \leftarrow \mathsf{VC.Verify}(\mathsf{sk}, \mathsf{Enc}(\mathsf{out}), \mathsf{st}, \pi)$*: given the secret key* sk*, the encoded result* $\mathsf{Enc}(\mathsf{out})$*, the state* st*, and the proof* π*, the verification algorithm returns the plaintext* out *corresponding to* $\mathsf{Enc}(\mathsf{out})$ *if* $\mathsf{out} = C(x, y)$*, or outputs* $\perp$ *otherwise.*

A VCoED scheme should satisfy the following properties.

Correctness. *The correctness property ensures that the honest server will pass the verification of an honest client. Specifically, the following probability is negligible.*

$$\Pr\left[\perp \leftarrow \mathsf{VC.Verify}(\mathsf{sk}, \mathsf{Enc}(\mathsf{out}), \mathsf{st}, \pi) \;\middle|\; \begin{array}{c}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, C) \\ (\mathsf{Enc}(x), \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, x) \\ (\mathsf{Enc}(\mathsf{out}), \pi) \leftarrow \mathsf{VC.Compute}(\mathsf{pk}, \mathsf{Enc}(x), C, y)\end{array}\right]$$

Outsourceability. *A VCoED scheme is outsourceable if the time required by the client to run* $\mathsf{VC.ProbGen}$ *and* $\mathsf{VC.Verify}$ *and the size of* π *are* $o(|C|)$*, where* $|C|$ *is the size of circuit* C*.*

Security. The security of VCoED consists of two parts: the security of the client and the security of the server (if the server also has input).

Client security: It consists of the following two properties.

- Client-privacy: meaning that a malicious server should not learn any information about the client's input. More formally, a VCoED scheme is client-private, if for any PPT adversary $\mathcal{A}$, its advantage in the following game $\mathsf{G}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda)$, defined as
$\mathsf{Adv}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda) \to 1] - 1/2 = \mathsf{negl}(\lambda)$.

$\underline{\mathsf{G}^{\mathsf{client-privacy}}_{\mathsf{VC},\mathcal{A}}(\lambda)}$:

1: $b \xleftarrow{\$} \{0,1\}$

2: $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathrm{C})$

3: $(m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{pk}, \mathrm{C})$

4: $(c_{m_b}, \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, m_b)$

5: $\hat{b} \leftarrow \mathcal{A}(c_{m_b}, \mathsf{state})$

6: **return** $b \stackrel{?}{=} \hat{b}$

- Soundness/Knowledge soundness: In the case where the server has no input, a VCoED scheme should be sound such that no adversary can convince the client of a false result except with negligible probability. When the server also has its own input, following [31], the notion must be strengthened to knowledge soundness, defined as follows: there exists a probabilistic polynomial-time oracle machine $\mathcal{E}$, called the extractor, such that for any PPT adversary $\mathcal{A}$, the following probability is negligible:

$$\Pr\left[\begin{array}{r|r} \mathsf{VC.Verify}(\mathsf{sk}, c, \mathsf{st}, \pi) = \mathsf{out} \neq \perp \ \wedge & (\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathrm{C}) \\ \mathsf{out} \neq \mathrm{C}(x, y) & (\mathsf{Enc}(x), \mathsf{st}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{pk}, x) \\ & (c, \pi) \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(x), \mathrm{C}) \\ & y \leftarrow \mathcal{E}^{\mathcal{A}}((\mathsf{pk}, \mathsf{Enc}(x), \mathrm{C}), (c, \pi)) \end{array}\right]$$

Server security: Following [1], we provide a simulation-based definition for server-privacy.

- Server-privacy: A VCoED protocol is server-private if there exists a PPT simulator $\mathcal{S} = (S_1, S_2)$ such that for every circuit C, every input y, and every adversary $\mathcal{A}$, the following distributions are computational indistinguishable:

  $\mathcal{D}_1$: $\{\mathsf{pk}, \mathsf{sk}, r, \delta_{out}, \pi\}_\lambda$, where $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{VC.Setup}(1^\lambda, \mathrm{C})$, $\delta_x \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{sk}; r)$, r is the randomness of $\mathcal{A}$, and $(\delta_{out}, \pi) \leftarrow$ $\mathsf{VC.Compute}(\mathsf{pk}, \delta_x, \mathrm{C}, y)$.

$\mathcal{D}_2$: $\{\mathsf{pk}, \mathsf{sk}, r, \mathcal{S}_2^{\mathcal{A},\mathcal{O}_y}(1^\lambda, \tau, \delta_x)\}_\lambda$, where $(\mathsf{sk}, \mathsf{pk}, \tau) \leftarrow \mathcal{S}_1(1^\lambda, C)$, $\delta_x \leftarrow \mathcal{A}(\mathsf{pk}, \mathsf{sk}; r)$, r is the randomness of $\mathcal{A}$, and $\mathcal{O}_y$ is a one-time oracle ($\mathcal{S}_2$ can query this oracle for only once) that on input x, it outputs $\mathrm{C}(x, y)$.

Furthermore, a VCoED protocol is called honest-client server-private if it satisfies the property above against an honest-client.

Limitation of VCoED. As noted in previous work [16, 1, 14, 31], VCoED constructions based on "homomorphically computing the proof" suffer from a one-bit leakage in certain situations. Specifically, if a malicious prover can learn whether the proof it generated (under homomorphic evaluation) passes verification, it gains 1 bit of information about the encrypted input. [31] proves that the only gap between VCoED and 2PC is precisely this one-bit leakage regarding the acceptability of the proof. Therefore, in practice, the client must prevent the server from learning whether the proof is acceptable.

3.4 Multivariate Lagrange Interpolation

We review the standard notion of multivariate Lagrange interpolation on a rectangular grid. Given a function defined on a product set $D_1 \times \cdots \times D_k$, there exists a unique polynomial whose degree in each variable x_i is at most $|D_i| - 1$ and which agrees with the function on all grid points. The explicit construction uses products of univariate Lagrange basis polynomials.

Multivariate Lagrange Interpolation. Let $\mathbb{F}_p$ be a field and $k \geq 1$ an integer. For each $i \in \{1, \ldots, k\}$, let $d_i \geq 0$ be an integer and let

$$D_i = \{0, 1, \ldots, d_i\} \subset \mathbb{F}_p$$

Define the grid $\mathcal{G} = D_1 \times D_2 \times \cdots \times D_k \subset \mathbb{F}^k$. Then for any function $f : \mathcal{G} \to \mathbb{F}$, there exists a *unique* polynomial

$$P(x_1, \ldots, x_k) \in \mathbb{F}[x_1, \ldots, x_k]$$

satisfying:

1. **Degree constraints:** $\deg_{x_i}(P) \leq d_i$ for each $i = 1, \ldots, k$;
2. **Interpolation condition:** for every $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathcal{G}$, $P(\alpha) = f(\alpha)$.

In the rest of this paper, we refer to the polynomial P as the *bounded-degree Lagrange extension* of f.

Explicit Construction. For each coordinate i and each $a \in D_i$, define the univariate Lagrange basis polynomial (of degree $\leq d_i$):

$$\ell_{i,a}(x_i) = \prod_{\substack{c \in D_i \\ c \neq a}} \frac{x_i - c}{a - c}.$$

It satisfies $\ell_{i,a}(a) = 1$ and $\ell_{i,a}(c) = 0$ for all $c \in D_i \setminus \{a\}$.

For a grid point $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathcal{G}$, define the multivariate basis polynomial:

$$L_\alpha(x_1, \ldots, x_k) = \prod_{i=1}^{k} \ell_{i,\alpha_i}(x_i).$$

Then the unique interpolating polynomial is given by:

$$P(x_1, \ldots, x_k) = \sum_{\alpha \in \mathcal{G}} f(\alpha) \cdot L_\alpha(x_1, \ldots, x_k).$$

Remark. The set $\{L_\alpha\}_{\alpha \in \mathcal{G}}$ is linearly independent and each L_α respects $\deg_{x_i} \leq d_i$. Moreover, the space of polynomials with $\deg_{x_i} \leq d_i$ has dimension $\prod_{i=1}^{k}(d_i + 1) = |\mathcal{G}|$, so the interpolation map is a bijection. Special cases: if $d_i = 1$ and $D_i = \{0, 1\}$ for all i, we recover the uniqueness of multilinear extensions on the Boolean hypercube; if all $d_i = d$, we obtain the uniform-degree version.

4 Multiplication-Layered Circuits

Traditional GKR proof systems typically require circuits to be strictly layered, meaning all gates—both addition and multiplication—topologically synchronized level by level. While conceptually elegant, this rigid synchronization introduces *structural mismatches* and inefficiencies when applied to VCoED. Specifically, in BGV-like FHE schemes, only multiplication dictates the noise growth and structural depth; consequently, practical algorithms are designed to minimize multiplication depth without imposing any constraints on addition depth. Forcing addition gates to synchronize with multiplication gates artificially inflates the depth and overhead of the proof system.

To bridge the gap between the proof system's architecture and the underlying FHE evaluation logic, we introduce a novel circuit abstraction: *multiplication-layered circuits*. By relaxing the synchronization constraints on addition gates and enforcing layering solely on multiplication gates, we align the circuit representation with the natural multiplicative depth of FHE.

Definition 5 (Multiplication-Layered Circuit) *A circuit is called multiplication-layered if all its gates can be assigned "multiplicative" levels* $\{0, 1, \cdots, d-1\}$ *such that:*

1. *Gates at level* $d - 1$ *are all input gates, whose inputs are the circuit's inputs.*
2. *For every gate at level* $i < d - 1$*, all inputs of* g *are outputs of : 1) multiplication gates at level* $i + 1$*, or 2) addition gates at level* i *or* $i + 1$*.*
3. *Gates at level* 0 *are the output gates of the circuit.*

We denote by ℓ_i the set of gates at level i, by L_i the set of output wires from ℓ_i that go to level $i-1$ (often called simply the output wires of layer i), and by L_d the set of primary input wires.

Remark. Standard layered circuits are a special case of multiplication-layered circuits (by taking inputs only from level $i+1$). However, our definition is strictly more general: it permits gates to also receive inputs from the addition gates in the same level i, thereby avoiding artificial depth inflation due to addition-synchronization. That is, the layering constraint can apply only to the *critical paths* that pass through multiplication gates, while addition gates act as flexible "free" connectors.

Every acyclic circuit can be turned into a multiplication-layered circuit by assigning each multiplication gate its multiplicative depth as its level and inserting additive identity gates (which compute $x + 0$) to resolve mismatches in the arrival depths of addition gates. The transformation never adds multiplication gates, so the multiplicative depth remains unchanged.

4.1 A Simple GKR-style Proof System for Multiplication-Layered Circuits

In this section, we present a simple GKR-style proof system for multiplication-layered circuits. In later sections, we show how to transform it into a homomorphically encrypted version that works for homomorphic execution of multiplication-layered circuits with SIMD operations, along with several *critical optimization techniques* that significantly reduce overhead.

First, for a layer i of the multiplication-layered circuit, let L_i be its output wires and L_{i+1} be the output wires of the layer $i+1$ (which also provide the inputs to layer i). Assume $|L_i| \leq 2^{s_i}$ and $|L_{i+1}| \leq 2^{s_{i+1}}$. Assign each wire in L_i a distinct index $z \in \{0,1\}^{s_i}$. Define $L_i(\cdot)$ as the function on $\{0,1\}^{s_i}$ where $L_i(z)$ is the value of the indexed wire (or 0 if no such wire exists). And $L_{i+1}(\cdot)$ is defined analogously.

Consider a wire $L_i(z)$ that is the output of an addition gate in ℓ_i, so $L_i(z) = g_0 + g_1$ for input wires g_0, g_1. If both $g_0, g_1 \in L_{i+1}$ (i.e., $g_0 = L_{i+1}(x)$ and $g_1 = L_{i+1}(y)$ for some x, y), we are done. Otherwise, any input that is itself the output of another addition gate in the same layer ℓ_i is recursively expanded. Repeating this process ultimately expresses $L_i(z)$ as a linear combination of wires in L_{i+1}:

$$L_i(z) = \sum_{x \in \{0,1\}^{s_{i+1}}} a_{z,x} \cdot L_{i+1}(x),$$

where each $a_{z,x}$ is an integer.

Consider a wire $L_i(z)$ output by a multiplication gate in ℓ_i, so $L_i(z) = g_0 \cdot g_1$. Each input g_j is either a wire in L_{i+1} or the output of an addition gate in ℓ_i; in the latter case, the addition gate can be expanded into a linear combination of wires in L_{i+1} as shown above. Therefore, $L_i(z)$ becomes the product of two

linear combinations:

$$L_i(z) = \left(\sum_{x \in \{0,1\}^{s_{i+1}}} b_{z,x}\, L_{i+1}(x) \right) \left(\sum_{x \in \{0,1\}^{s_{i+1}}} c_{z,x}\, L_{i+1}(x) \right),$$

where $b_{z,x}, c_{z,x} \in \mathbb{Z}$.

We further define $a_{z,x} = 0$ if $L_i(z)$ is not an addition-gate output, and $b_{z,x} = c_{z,x} = 0$ if $L_i(z)$ is not an multiplication-gate output. Let $add_i(z,x) = a_{z,x}, mul_{l,i}(z,x) = b_{z,x}$ and $mul_{r,i}(z,x) = c_{z,x}$. Then we have that: for all $z \in \{0,1\}^{s_i}$

$$\begin{aligned} L_i(z) = & \sum_{x \in \{0,1\}^{s_{i+1}}} add_i(z,x) \cdot L_{i+1}(x) \\ & + \Big(\sum_{x \in \{0,1\}^{s_{i+1}}} mul_{\ell,i}(z,x) \cdot L_{i+1}(x) \Big) \cdot \Big(\sum_{x \in \{0,1\}^{s_{i+1}}} mul_{r,i}(z,x) \cdot L_{i+1}(x) \Big). \end{aligned}$$

Denote by $\tilde{L}_i(z)$, $\tilde{L}_{i+1}(z)$, $\widetilde{add}_i(z,x)$, $\widetilde{mul}_{l,i}(z,x)$ and $\widetilde{mul}_{r,i}(z,x)$ the corresponding multilinear extension. Define multilinear polynomials $h_{l,i}(z)$ and $h_{r,i}(z)$ as:

$$h_{l,i}(z) = \Sigma_{x \in \{0,1\}^{s_{i+1}}} \widetilde{mul}_{l,i}(z,x) \cdot \tilde{L}_{i+1}(x), \tag{1}$$

$$h_{r,i}(z) = \Sigma_{x \in \{0,1\}^{s_{i+1}}} \widetilde{mul}_{r,i}(z,x) \cdot \tilde{L}_{i+1}(x). \tag{2}$$

Then, by the uniqueness of multilinear extension,

$$\tilde{L}_i(z) = \sum_{x \in \{0,1\}^{s_{i+1}}} \widetilde{add}_i(z,x) \cdot \tilde{L}_{i+1}(x) + \sum_{y \in \{0,1\}^{s_i}} eq(y,z) \cdot h_{l,i}(y) \cdot h_{r,i}(y)$$

Now, we can adopt a GKR-like approach together with the well-known sumcheck protocol to build the proof system. Specifically, for each layer i, we reduce verifying $\tilde{L}_i(z_i)$ for any given z_i to a single query of $\tilde{L}_{i+1}(z_{i+1})$ via two consecutive invocations of the sumcheck protocol: one over y and one over x. By recursively performing these steps, the correctness check of the output wire L_{out} (i.e., verifying its value at a random point, $L_{\text{out}}(z_0)$) reduces to point queries on the input wires, which the client can compute on its own (assuming the client holds the input). The overall soundness error is $O(d \log \mathrm{C}/|\mathbb{F}_{p^R}|)$, where the challenges are chosen from $\mathbb{F}_{p^R}$. The detailed construction is in Fig.1

Remark. In the above construction, we assume that (1) the input input is known to the verifier, and (2) the polynomials $\widetilde{add}$, $\widetilde{mul}_\ell$, and $\widetilde{mul}_r$ can be efficiently computed by the verifier. In some scenarios, these requirements may not hold. In such cases, we can instead require the prover to commit to these polynomials via polynomial commitments and assist the verifier in obtaining the necessary values by sending corresponding evaluation proofs.

Let C be a multiplication-layered circuit over $\mathbb{F}_p$ with depth d. And let input (resp. out) is its input (resp. output) known by both the prover and verifier. Let R be such that challenges are drawn from $\mathbb{F}_{p^R}$ (to ensure soundness). The prover wants to convince the verifier that $\mathsf{out} = C(\mathsf{input})$. They run as follows:

1. Define $\tilde{L}_{\text{out}}$ as the multilinear extension of the primary output wire. The verifier chooses a random string $z_0 \in \mathbb{F}_{p^R}$, and both party compute $\tilde{L}_{out}(z_0)$. Set $k_0 = \tilde{L}_{out}(z_0)$.
2. From $i = 0$ to $d - 1$, both party run as follows:
   (a) The prover computes and sends $t_i = \sum_{y \in \{0,1\}^{s_i}} eq(y, z_i) \cdot h_{l,i}(y) \cdot h_{r,i}(y)$ to the verifier, where $h_{l,i}(y)$ and $h_{r,i}(y)$ are defined in eq.1 and eq.2.
   (b) Run the sumcheck protocol on above equation $t_i = \sum_{y \in \{0,1\}^{s_i}} eq(y, z_i) \cdot h_{l,i}(y) \cdot h_{r,i}(y)$. This reduces the verification to queries
   $$a_1 = h_{l,i}(r') = \Sigma_{x \in \{0,1\}^{s_{i+1}}} \widetilde{mul}_{l,i}(r', x) \cdot \tilde{L}_{i+1}(x),$$
   $$a_2 = h_{r,i}(r') = \Sigma_{x \in \{0,1\}^{s_{i+1}}} \widetilde{mul}_{r,i}(r', x) \cdot \tilde{L}_{i+1}(x).$$
   (c) The verifier sends $r_1, r_2, r_3 \in \mathbb{F}_{p^R}$ to the prover. Then, run the sumcheck protocol for
   $$\sum_{x \in \{0,1\}^{s_{i+1}}} (r_1 \cdot \widetilde{mul}_{l,i}(r', x) + r_2 \cdot \widetilde{mul}_{r,i}(r', x) + r_3 \cdot \widetilde{add}(z_i, x))\tilde{L}_{i+1}$$
   $$= r_1 a_1 + r_2 a_2 + r_3(k_i - t_i)$$
   (d) At the end of the above sum-check protocol, the verification is reduced to a query $k_{i+1} = \tilde{L}_{i+1}(z_{i+1})$.
3. The verifier checks whether $k_d = \tilde{L}_d(z_d)$.

Figure 1: A GKR-style proof system for multiplication-layered circuits

5 Flexible Sumcheck over Encrypted Polynomials

To run the above GKR-style protocol homomorphically for encrypted statements, we need to run the sumcheck protocol over encrypted polynomials. The Phalanx scheme provides such a construction, which reduces the multiplicative depth of the scheme making it friendly for homomorphic evaluation. However,

unlike Phalanx which invokes the sumcheck protocol only twice, our GKR protocol requires two sequential invocations of sumcheck *per layer*, leading to a significant increase in communication overhead. Moreover, for sumcheck protocols over polynomials in a small field, all rounds except the first require computations in an extension field, substantially increasing the prover's overhead.

To address these issues, we develop a new variant of the sumcheck protocol over encrypted polynomials that reduces communication overhead while minimizing the increase in both prover and verifier time. It features constant multiplication depth and supports SIMD operations.

Compared with previous schemes, our scheme has the following features:

1. **Flexible summation space with fewer rounds.** It supports summation over more flexible hypercubes (e.g., $[k]^n$ or $[k_0] \times [k_1] \times \cdots \times [k_{n-1}]$) rather than just $\{0,1\}^n$. This allows us to encode circuit values using multivariate polynomials with higher degrees in the overall GKR scheme, rather than relying on multilinear polynomials. As a result, the number of rounds in the sumcheck protocol is significantly reduced, as is the computational overhead of extension field arithmetic.

2. **Packed Proofs.** The ciphertexts to be sent in each round of the sumcheck protocol are packed into a single ciphertext, effectively reducing per-round communication overhead. Consequently, reducing the number of rounds (as achieved by the flexible summation space above) directly reduces the total communication overhead.

3. **Early termination.** Similar to Phalanx, we require the sumcheck protocol to terminate early after a predefined number of rounds, reducing both the communication and prover overhead. This does not increase the verifier's overhead, since the additional computations over plaintexts are far cheaper than decrypting ciphertexts in the original proofs

Specifically, in our construction, we need to run the sumcheck protocol on two sums of the form $\sum_y \mathrm{eq}(y,z) \cdot h_l(y) \cdot h_r(y)$ and $\sum_x M(x) \cdot L(x)$, where h_l, h_r, and L are encrypted polynomials. Below, we present the concrete sumcheck protocol constructions for each of these two claims.

First Sumcheck Protocol. Consider (encrypted) multivariate polynomials $h_l(y)$ and $h_r(y)$, where $y = (y_0, y_1, \cdots, y_{s-1})$ and each polynomial has degree d_i in y_i. Then, both polynomials are determined by their evaluations over $\mathcal{D} = [d_0+1] \times [d_1+1] \times \cdots \times [d_{s-1}+1]$. Let $eq(\cdot,\cdot)$ be the extended equality function that for any $x, y \in \mathcal{D}$, $eq(x,y) = 1$ if and only if $x = y$. The verifier holds the secret key of the underlying encryption scheme, while the prover holds the public key. The prover also knows the packed ciphertexts of the evaluations as follows:

For any fixed $\mathbf{a} = (a_0, \ldots, a_{s-t-1}) \in \prod_{i=0}^{s-t-1} [d_i+1]$ to the first $s-t$ variables,

define the set of evaluations over the remaining t variables:

$$V_{\mathbf{a}}^{(l)} = \left\{ h_l(\mathbf{a}, y_{s-t}, \ldots, y_{s-1}) \;\middle|\; (y_{s-t}, \ldots, y_{s-1}) \in \prod_{j=s-t}^{s-1} [d_j + 1] \right\}.$$

The prover holds the **packed ciphertext** $\mathsf{ct}[h_l(\mathbf{a}, \cdot)] = \mathsf{Enc}\big(V_{\mathbf{a}}^{(l)}\big)$, and analogously $\mathsf{ct}[h_r(\mathbf{a}, \cdot)]$ for h_r.

The sumcheck protocol for $v = \sum_{y \in \mathcal{D}} eq(y, z_i) \cdot h_l(y) \cdot h_r(y)$, which terminates after s^* round (with $s - s^* \leq t$), is shown in Fig.2.

The Second Sumcheck. This sumcheck works for statements of the form $\sum_{y \in \mathcal{D}} M(y) \cdot L(y)$, where $L(y)$ is an encrypted polynomial in $y = (y_0, y_1, \ldots, y_{s-1})$ with degree d_i in each y_i, and $M(y)$ is a public polynomial of the same degrees. The prover is given the packed ciphertexts $\mathsf{ct}[L(\mathbf{a})]$ defined analogously to the first sumcheck.

The protocol proceeds in the same manner as the first sumcheck, with the following modifications. First, the polynomial $\mathrm{eq}(y, z_i)$ is replaced by the public polynomial $M(y)$. Second, because only one encrypted polynomial $L(y)$ is involved, the protocol is simplified compared to the original, and in particular, no ciphertext-ciphertext multiplication is required throughout.

More concretely, define $g(y) := M(y) \cdot L(y)$. The interactive reduction is identical to that in the first sumcheck, except that every occurrence of $\mathrm{eq}(\cdot, z_i) \cdot h_l(\cdot) \cdot h_r(\cdot)$ is replaced by $M(\cdot) \cdot L(\cdot)$.

Additionally, if computing all evaluations $M(r, y')$ for points $y' \in \prod_{\ell=s^*}^{s-1} [d_\ell + 1]$ in the final step is inefficient for the verifier, the prover may instead send these values in plaintext. Let $M_r(y')$ denote the polynomial defined by the received evaluations. The verifier then performs the check in Step (3) via $v_{s^*+1} = \sum_{y' \in \prod_{\ell=s^*}^{s-1} [d_\ell+1]} M_r(y') \cdot L'(r, y')$. To ensure the correctness of $M_r(y')$, the verifier samples a random r' and checks that $M_r(r') = M(r, r')$, either via oracle access to M or by direct evaluation when $M(r, r')$ can be computed efficiently.

6 Construction of the Lasagne VCoED Scheme

The core of VCoED scheme is to prove the integrity of payload circuit evaluations. A straightforward solution is to use a homomorphic version of previously described GKR-style protocol for encrypted instances. However, this approach introduces excessive overall communication overhead and doesn't support proofs for SIMD-evaluated payload circuits. To address these issues,we introduce the following two key improvements to the proof scheme for encrypted instances:

Multivariate constant-degree polynomial encoding. Unlike traditional SNARK schemes, our scheme encodes the instance as a multivariate polynomial of constant degree. To this end, we restructure the indexing of wire values

[1]For $j > 1$, $g_j^*(k) \in \mathbb{F}_{p^R}$; thus $\mathsf{Enc}(g_j^*(k))$ consists of R ciphertexts over $\mathbb{F}_p$. That's a mild abuse of notation.

Suppose $v = \sum_{y \in \mathcal{D}} eq(y, z_i) \cdot h_l(y) \cdot h_r(y)$ is the proved statement except that polynomials h_l and h_r is now encrypted by $\{\mathsf{ct}[h_l(\mathbf{a}, \cdot)]\}_{\mathbf{a} \in \prod_{i=0}^{s-t-1}[d_i+1]}$ and $\{\mathsf{ct}[h_r(\mathbf{a}, \cdot)]\}_{\mathbf{a} \in \prod_{i=0}^{s-t-1}[d_i+1]}$, and v is also encrypted. Both parties execute as follows:

1. Let $g(y) := eq(y, z_i) \cdot h_l(y) \cdot h_r(y)$. Let r be an empty vector, $v_1 = v$, for j from 1 to s^*, run as follows:
   (a) Consider the following $3d_{j-1}$-degree *univariate* polynomial:
   $$g_j^*(Y) := \sum_{y' \in \prod_{\ell=j}^{s-1}[d_\ell+1]} g(r, Y, y').$$
   For each $k \in [3d_{j-1} + 1]$, the prover computes $\mathsf{ct}_k^* = \mathsf{Enc}(g_j^*(k))$[1] from the ciphertexts $\mathsf{ct}[h_l(\mathbf{a}, \cdot)]$ and $\mathsf{ct}[h_r(\mathbf{a}, \cdot)]$. The detailed computation is shown in Fig.3.
   (b) (Ciphertext Pack, optional) Pack all ciphertexts of $\{\mathsf{ct}_k^*\}_{k \in [3d_{j-1}+1]}$ into a single packed ciphertext using plaintext-ciphertext multiplications, rotations and additions. Then send the packed ciphertext to the verifier.
   (c) The verifier decrypts the received ciphertext and checks whether $\sum_{k \in [d_{j-1}+1]} g_j^*(k) = v_j$. If so, it reconstructs $g_j^*(Y)$ using all $g_j^*(k)$. It then samples and sends $r' \leftarrow \mathbb{F}_{p^R}$ to the prover. Set $v_{j+1} := g_j^*(r')$ and update $r := (r, r')$.
2. Now the claim is reduced to the sumcheck claim that $v_{s^*+1} := \sum_{y' \in \prod_{\ell=s^*}^{s-1}[d_\ell+1]} g(r, y')$. The prover computes the packed ciphertexts of set $\{h_l(r, y')\}_{y' \in \mathcal{D}'}$ and $\{h_r(r, y')\}_{y' \in \mathcal{D}'}$, where $\mathcal{D}' \in \prod_{\ell=s^*}^{s-1}[d_\ell + 1]$; the detailed computation is shown in Fig.3. Optionally, the prover may repack the obtained ciphertexts to reduce their number (similar to step 1(b)). Finally, the prover sends the ciphertexts to the verifier.
3. The verifier decrypts the received ciphertexts and obtains the two corresponding polynomials $h_l'(r, y')$ and $h_r'(r, y')$. It then checks whether
   $$v_{s^*+1} = \sum_{y' \in \prod_{\ell=s^*}^{s-1}[d_\ell+1]} \mathrm{eq}(y', z_i) \cdot h_l'(r, y') \cdot h_r'(r, y').$$
   If the equality holds, the verifier samples a random $r' \in \mathbb{F}_{p^R}^{s-s^*}$ and computes $v_l = h_l'(r, r')$ and $v_r = h_r'(r, r')$. It then queries the (encrypted) polynomial oracles of h_l and h_r at the point (r, r'), and outputs 1 if the responses are v_l and v_r, respectively.

Figure 2: Flexible Sumcheck over Encrypted Polynomial

The algorithm for computing ct_k^* in step 1(a) of Fig.2:

If $s - j - 1 \geq t$, run as follows:

1. For each $k \in [3d_{j-1} + 1]$ and $y'' \in \prod_{\ell=j}^{s-t-1}[d_\ell + 1]$, the prover computes the packed ciphertext $\mathsf{ct}[h_l(r, k, y'', \cdot)]$ which encrypts the set $\{h_l(r, k, y'', y^*)\}_{y^* \in \prod_{\ell=s-t}^{s-1}[d_\ell+1]}$ by homomorphically computing

$$\sum_{b \in \prod_{\ell=0}^{j-1}[d_\ell+1]} h_l(b, y'', y^*) \cdot eq(b, (r, k))$$

   using only *scalar multiplications* on all packed ciphertexts $\mathsf{ct}[h_l(\mathbf{a}, \cdot)]$. Note that $h_l(r, k, y'', y^*)$ lies in $\mathbb{F}_{p^R}$ (except when $j = 1$, in which case it remains in $\mathbb{F}_p$); therefore, each $\mathsf{ct}[h_l(r, k, y'', \cdot)]$ consists of R ciphertexts over $\mathbb{F}_p$. The packed ciphertexts $\mathsf{ct}[h_r(r, k, y'', \cdot)]$ are computed analogously.

2. For the same indices k and y'', The prover computes the packed ciphertexts $\mathsf{ct}[g(r, k, y'', \cdot)]$ of $\{g(r, k, y'', y^*)\}_{y^* \in \prod_{\ell=s-t}^{s-1}[d_\ell+1]}$ via homomorphically computing

$$\big(eq((r, k, y'', y^*), z_i) \cdot h_l(r, k, y'', y^*)\big) \cdot h_r(r, k, y'', y^*)$$

   from $\mathsf{ct}[h_l(r, k, y'', \cdot)]$ and $\mathsf{ct}[h_r(r, k, y'', \cdot)]$, using first plaintext-ciphertext multiplications followed by ciphertext-ciphertext multiplications.

3. The prover computes $\mathsf{ct}_k^* = \sum_{y' \in \prod_{\ell=j}^{s-1}[d_\ell+1]} \mathsf{Enc}(g(r, k, y'))$ by performing additions and rotations on $\mathsf{ct}[g(r, k, y'', \cdot)]$.

If $s - j - 1 < t$, run as follows:

1. Divide $r = (r_1, r_2)$ with $r_1 \in \mathbb{F}_{p^R}^{s-t}$. The prover computes the packed ciphertexts $\mathsf{ct}[h_l(r_1, \cdot)]$ of $\{h_l(r_1, y^*)\}_{y^* \in \prod_{\ell=s-t}^{s-1}[d_\ell+1]}$ in the same manner as above. The packed ciphertexts for h_r are computed analogously.

2. For each $k \in [3d_{j-1} + 1]$ and $y' \in \prod_{\ell=j}^{s-1}[d_\ell + 1]$, the prover rewrites $eq((r, k, y'), z_i) \cdot h_l(r, k, y')$ as

$$\begin{aligned} & eq((r, k, y'), z_i) \cdot \left(\sum_{b \in \prod_{\ell=s-t}^{j-1}[d_\ell+1]} h_l(r_1, b, y') \cdot eq(b, (r_2, k)) \right) \\ = & \sum_{b \in \prod_{\ell=s-t}^{j-1}[d_\ell+1]} h_l(r_1, b, y') \cdot (eq(b, (r_2, k)) \cdot eq((r, k, y'), z_i)), \end{aligned}$$

   and computes the packed ciphertexts of $\{h_l(r_1, y^*) \cdot (eq(b, (r_2, k)) \cdot eq((r, k, y'), z_i))\}_{y^* = (b, y') \in \prod_{\ell=s-t}^{s-1}[d_\ell+1]}$ using plaintext-ciphertext multiplications on $\mathsf{ct}[h_l(r_1, \cdot)]$.

3. From the above packed ciphertexts, the prover computes the packed ciphertexts of $\{eq((r, k, y'), z_i) \cdot h_l(r, k, y')\}_{y' \in \prod_{\ell=j}^{s-1}[d_\ell+1]}$ using only additions and rotations. The packing ciphertexts of $\{h_r(r, k, y')\}_{y' \in \prod_{\ell=j}^{s-1}[d_\ell+1]}$ are computed similarly, and finally ct_k^* is obtained directly from them.

The computation for the packed ciphertexts of $\{h_l(r, y')\}_{y' \in \mathcal{D}'}$ and $\{h_r(r, y')\}_{y' \in \mathcal{D}'}$, where $\mathcal{D}' \in \prod_{\ell=s^*}^{s-1}[d_\ell + 1]$, can be done using the same strategy as above (case $s - j - 1 < t$).

Figure 3: Algorithms for computing ct_k^*

in each layer of the GKR protocol, as well as the relationship between wires across adjacent layers, ensuring compatibility with our Flexible Sumcheck protocol. This approach reduces the number of sumcheck rounds, and significantly decreases the proof size.

Packing strategy for SIMD-evaluated payload circuit. Unlike the case of non-SIMD execution, where intermediate ciphertexts of the payload circuit evaluation can be arbitrarily packed to suit the encrypted proof algorithm, the intermediate states in SIMD-executed circuits are themselves packed ciphertexts. Extracting and repacking them is extremely time-consuming, rendering the entire scheme impractical. To address this, we propose a novel index assignment and packing strategy that enables the proof algorithm to leverage these packed intermediate ciphertexts via packing directly on top of these ciphertexts without any extraction.

6.1 Multivariate Constant-Degree Polynomial Encoding.

We modify the construction of our GKR-Style proof scheme as follows. First, consider any layer i, let L_i be the set of output wires of this layer, and let $\mathcal{D}_i = [d_0 + 1] \times [d_1 + 1] \times \cdots \times [d_{s-1} + 1]$ be a hypercube such that $|\mathcal{D}_i| \geq |L_i|$. For each wire in L_i, we assign a distinct index $z \in \mathcal{D}_i$. Define $L_i(\cdot)$ as a function on $\mathcal{D}_i$ where $L_i(z)$ equals the value of the wire indexed by z (with $L_i(z) = 0$ if z does not correspond to any wire). Let $L_{i+1}(\cdot)$ be a function on $\mathcal{D}_{i+1}$ defined analogously for layer $i + 1$.

Next, following the same analysis strategy as in the previous GKR-style proof protocol, we have for all $z \in \mathcal{D}_i$:

$$L_i(z) = \sum_{x \in \mathcal{D}_{i+1}} add_i(z, x) \cdot L_{i+1}(x) + \left(\sum_{x \in \mathcal{D}_{i+1}} mul_{l,i}(z, x) \cdot L_{i+1}(x) \right) \cdot \left(\sum_{x \in \mathcal{D}_{i+1}} mul_{r,i}(z, x) \cdot L_{i+1}(x) \right)$$

where $add_i(z, x)$, $mul_{l,i}(z, x)$, and $mul_{r,i}(z, x)$ are functions on $\mathcal{D}_i \times \mathcal{D}_{i+1}$ that depend only on the payload circuit.

Following Section 3.4, we denote by $\tilde{L}_i(z)$ the bounded-degree Lagrange extension of $L_i(z)$; that is, $\tilde{L}_i(z)$ is the *unique* multivariate polynomial satisfying $\deg_{z_j} \tilde{L}_i(z) \leq d_j$ for each j and $\tilde{L}_i(z) = L_i(z)$ for all $z \in \mathcal{D}_i$. We define $\tilde{L}_{i+1}(z)$, $\widetilde{add}_i(z, x)$, $\widetilde{mul}_{l,i}(z, x)$, and $\widetilde{mul}_{r,i}(z, x)$ analogously. Let $h_{l,i}(z)$ and $h_{r,i}(z)$ be the multivariate polynomials given by

$$h_{l,i}(z) = \Sigma_{x \in \mathcal{D}_{i+1}} \widetilde{mul}_{l,i}(z, x) \cdot \tilde{L}_{i+1}(x), \tag{3}$$

$$h_{r,i}(z) = \Sigma_{x \in \mathcal{D}_{i+1}} \widetilde{mul}_{r,i}(z, x) \cdot \tilde{L}_{i+1}(x). \tag{4}$$

Then, from the *uniqueness* of bounded-degree Lagrange extension, we have

that:

$$\tilde{L}_i(z) = \sum_{x \in \mathcal{D}_{i+1}} \widetilde{add}_i(z, x) \cdot \tilde{L}_{i+1}(x) + \sum_{y \in \mathcal{D}_i} eq(y, z) \cdot h_{l,i}(y) \cdot h_{r,i}(y)$$

So far, we have established the relationship between the wire values of adjacent layers. Similarly, we can complete the construction of the GKR-style protocol using appropriate Sumcheck variants. In particular, when the wire values are encrypted and packed properly, we can homomorphically execute this GKR protocol using the Flexible Sumcheck protocol constructed earlier. We leave the detailed description to the following sections.

6.2 Packing strategy for SIMD-evaluated payload circuit.

In this section, we present a packing strategy for both the payload circuit and the proof algorithm. Suppose that in our GKR-style scheme, all $\mathcal{D}_i$ are chosen such that they share the same last t dimensions (i.e., the same d_ℓ for $\ell = s-t, \ldots, s-1$[3]). Assume the encryption scheme has $d_{\text{slot}} = \prod_{\ell=s-t}^{s-1}(d_\ell+1)$ slots, and the payload circuit can be SIMD-evaluated using $d_{\text{pslot}} = \prod_{\ell=s-t}^{s-t+\gamma-1}(d_\ell+1)$ slots, where $\gamma \geq s-t$ and consequently $d_{\text{pslot}} \mid d_{\text{slot}}$.

SIMD-evaluated multiplicative-layered payoad circuit. We say that a multiplication-layered payload circuit *supports SIMD execution* if it can be represented as a "circuit" containing SIMD multiplication, SIMD addition, and rotation gates satisfying the following requirements:

- Treating each SIMD multiplication/addition gate as d_{pslot} parallel multiplication/addition gates, and each rotation gate as wire connections, recovers the origin arithmetic circuit.
- The arithmetic gates decomposed from a SIMD gate always stay within the same layer, and their output wires are either all output wires of that layer, or none of them.

Wire index assignment. For the assignment of wires' indices at each layer i, we only need to satisfy the following core principles:

- The indices z of every wire belong to $\mathcal{D}_i$.
- For the d_{pslot} wires split from the same SIMD gate, their indices must share the same first $s-t$ entries and the same last $t-\gamma$ entries.
- The wires of multiplication gate and addition gate always have different first $s-t$ entries.

[3]For simplicity, we assume that all $\mathcal{D}_i$ have the same number of dimensions, namely s. The case where different $\mathcal{D}_i$ have different numbers of dimensions can be handled by a straightforward extension.

This assignment is straightforward because the middle γ components alone provide exactly $d_{\text{pslot}} = \prod_{\ell=s-t}^{s-t+\gamma-1}(d_\ell + 1)$ distinct index combinations, enough to encode all wires split from the same SIMD gate. We perform the assignment lexicographically under above requirements.

Payload Circuit Packing & Evaluation. We first describe how to adjust the packing strategy of the payload circuit so that its intermediate packed ciphertexts can be directly used by the proof algorithm. Suppose the payload circuit C takes $x = (x_0, \cdots, x_n)$ as input where each x_i is a set $\{x_{i,0}, \cdots, x_{i,d_{\text{pslot}}-1}\}$ consisting d_{pslot} elements (used by SIMD or rotation gates). We encrypt it into a single packed ciphertext via $\text{Enc}_{d_{\text{slot}},d_{\text{pslot}}}(pk, x_i)$ defined as:

$$\text{Enc}\big(pk,\ (\underbrace{x_{i,0}, x_{i,0}, \ldots, x_{i,0}}_{d_{\text{slot}}/d_{\text{pslot}}\text{ times}},\ \underbrace{x_{i,1}, \ldots, x_{i,1}}_{d_{\text{slot}}/d_{\text{pslot}}\text{ times}}\ ,\ \ldots,\ x_{i,d_{\text{pslot}}-1})\big),$$

where each $x_{i,j}$ is repeated $d_{\text{slot}}/d_{\text{pslot}}$ times.

Now, we modify SIMD homomorphically evaluation of the original circuit C so that it can run over our packed ciphertexts: SIMD multiplication and addition are replaced by the corresponding SIMD operations on ciphertexts with d_{slot} slots, while each rotation $\text{rot}(ct, j)$ is replaced by $\text{rot}(ct,\ j \cdot d_{\text{slot}}/d_{\text{pslot}})$. Then, it is easy to see that the modified evaluation, when given the above packed ciphertexts as input, outputs the correct output ciphertexts packed in the same way. We denote by the modified SIMD homomorphically evaluation as Eval_{mod}[4].

Proof System Packing Algorithm. Once the payload circuit evaluation is complete, the prover must pack the intermediate ciphertexts into the specific format required by the proof system. Packing $d_{\text{slot}}/d_{\text{pslot}}$ ciphertexts into one can be simply achieved through plaintext-ciphertext multiplications and additions. In particular, for the wires of a SIMD gate whose last $t - \gamma$ entries are z', we multiply the packed output ciphertext by the vector $(e_{z'}, e_{z'}, \ldots, e_{z'})$, where $e_{z'}$ is a vector of length $d_{\text{slot}}/d_{\text{pslot}}$ with a 1 at position z' and 0 elsewhere. The resulting ciphertext then has the property that for every possible $z^* \in \prod_{\ell=s-t}^{s-t+\gamma-1}[d_\ell + 1]$, the slot indexed by (z^*, z') encrypts the value of the wire with index (z^*, z'), while all other slots encrypt 0.

The prover packing algorithm (for each layer) proceeds as follows:

1. For each first $s - t$ prefix $\mathbf{a}$, where all wires with this prefix originate from $d_{\text{slot}}/d_{\text{pslot}}$ SIMD multiplication gates. We retrieve their left packed inputs and pack them into a single packed ciphertext using the above approach. The resulting ciphertext is exactly $\mathsf{ct}[h_l(\mathbf{a}, \cdot)]$. The ciphertexts $\mathsf{ct}[h_r(\mathbf{a}, \cdot)]$ can be computed analogously. These ciphertexts are required by the first flexible sumchecks.

[4]We note that this redundant packing strategy introduces some computational overhead during the homomorphic evaluation of the payload circuit. However, this overhead is strictly marginal compared to the dominant cryptographic proving time. More importantly, this stragy is essential, as it significantly reduce the computational complexity in the proof system packing algorithm.

2. By computing $\mathsf{ct}[h_l(\mathbf{a},\cdot)] \times \mathsf{ct}[h_r(\mathbf{a},\cdot)]$, we obtain the packed ciphertexts for the wires with prefix $\mathbf{a}$.

3. For each first $s-t$ prefix $\mathbf{b}$, where all wires with this prefix come from $d_{\text{slot}}/d_{\text{pslot}}$ SIMD addition (or rotation) gates, we take their output packed ciphertexts and pack them into a single ciphertext using the above approach. At this point, we have packed ciphertexts for $\{L_i(\mathbf{b},\cdot)\}$, which again meet the requirements of our flexible sumcheck protocol.

Key Optimization for packing. The packing algorithm in our proof system necessitates ciphertext multiplications. At the implementation level, these operations are dominated by NTT transforms. This is where the structural advantage of our *multiplication-layered circuit design* becomes highly apparent. Because the proof system is layered specifically around multiplication gates, the intermediate ciphertexts required for the proof system's packing phase are identical to the inputs of the multiplication gates in the payload circuit. During the native homomorphic evaluation of the payload circuit, these specific inputs are inherently converted into their NTT-domain representations to execute the payload's own multiplication operations. Consequently, the prover can directly fetch and reuse these states. This optimization completely bypasses the need for redundant, expensive NTT transformations, thereby significantly reducing the total computational overhead of the packing phase.

6.3 Our proof system for encrypted computation

Next, we can construct our GKR-style proof scheme for *encrypted* computation by combining the encoding and packing strategies presented earlier, the GKR-style proof protocol, and the Flexible Sumcheck protocol for encrypted polynomials. The formalized construction is as follows:

Theorem 1 *The protocol constructed in fig.4 is a public-coin proof system with soundness error* $O(d \log C/p^R)$.

proof sketch. The proof of our proof system follows a strategy similar to that of the original GKR protocol. Roughly speaking, consider any $\mathsf{output} \neq C(\mathsf{input})$. Let L'_i be the true values computed by $C(\mathsf{input})$; then $L'_0 \neq \mathsf{output}$. Since the total degree of $\tilde{L}'_0$ is $O(\log C)$, the Schwartz–Zippel lemma implies that the probability $\tilde{L}'_0(z_0) = k_0 = \tilde{L}_{\text{out}}(z_0)$ is $O(\log C/p^R)$.

Then for each $i \in [d]$ in step(3), the probability that, the prover transform a false claim $k_i \neq \tilde{L}'_i(z_i)$ into a true claim $k_{i+1} = \tilde{L}'_{i+1}(z_{i+1})$ at the ending in this recursion, is at most $O(\log C/p^R)$. Roughly speaking, the prover can succeed only in the following two scenarios: either (i) in one round of the flexible sumcheck, the verifier unluckily picks a "bad" r' such that the encrypted polynomial $f(X) = \sum_x g(r, X, x)$ sent by the prover coincides at $X = r'$ with the true polynomial $\sum_x g'(r, X, x)$, despite the two being different; or (ii) in step (3.c), the verifier's random linear combination turns a set of false claims into a

Let C be a multiplication-layered circuit over $\mathbb{F}_p$ of depth d. Let input (resp. output) denote its input (resp. output), both known to the verifier. The prover, on the other hand, only knows their packed encryptions *following our packing strategy*, along with all packed ciphertexts obtained during the modified SIMD homomorphic evaluation of C. Suppose the indices of output wires in each layer are defined as discussed in Sections 6.1 and 6.2. The protocol then proceeds as follows:

1. The verifier chooses and sends a random string $z_0 \in \mathbb{F}_{p^R}$, and computes $\tilde{L}_{out}(z_0)$. The prover homomorphically computes $\tilde{L}_{out}(z_0)$. Set $k_0 = \tilde{L}_{out}(z_0)$.
2. The prover runs the packing algorithm in Section 6.2.
3. From $i = 0$ to $d - 1$, both party run as follows:
   (a) The prover homomorphically computes and sends $t_i = \sum_{y \in \mathcal{D}_i} eq(y, z_i) \cdot h_{l,i}(y) \cdot h_{r,i}(y)$ to the verifier, where $h_{l,i}(y)$ and $h_{r,i}(y)$ are defined in eq.3 and eq.4.
   (b) Run the flexible sumcheck protocol on above equation $t_i = \sum_{y \in \mathcal{D}_i} eq(y, z_i) \cdot h_{l,i}(y) \cdot h_{r,i}(y)$. This reduces the verification to queries
   $$a_1 = h_{l,i}(r') = \Sigma_{x \in \mathcal{D}_{i+1}} \widetilde{mul}_{l,i}(r', x) \cdot \tilde{L}_{i+1}(x),$$
   $$a_2 = h_{r,i}(r') = \Sigma_{x \in \mathcal{D}_{i+1}} \widetilde{mul}_{r,i}(r', x) \cdot \tilde{L}_{i+1}(x).$$
   (c) The verifier sends $r_1, r_2, r_3 \in \mathbb{F}_{p^R}$ to the prover. Then, run the flexible sumcheck on
   $$\sum_{x \in \mathcal{D}_{i+1}} (r_1 \cdot \widetilde{mul}_{l,i}(r', x) + r_2 \cdot \widetilde{mul}_{r,i}(r', x) + r_3 \cdot \widetilde{add}(z_i, x)) \tilde{L}_{i+1}$$
   $$= r_1 a_1 + r_2 a_2 + r_3 (k_i - t_i)$$
   (d) At the end of the above sum-check protocol, the verification is reduced to a query $k_{i+1} = \tilde{L}_{i+1}(z_{i+1})$.
4. The verifier checks whether $k_d = \tilde{L}_d(z_d)$.

Figure 4: A GKR-style proof system for SIMD executed multiplication-layered circuits

correct one. By the Schwartz–Zippel lemma, each of these events occurs with probability $O(1/p^R)$.

Non-interactive, Fiat-Shamir heuristic. The above proof actually demonstrates that our scheme simultaneously satisfies round-by-round soundness. Consequently, we can apply the Fiat–Shamir heuristic to convert it into a non-interactive protocol, where each round's challenge is obtained by hashing the current transcript (the hash is computed in the plaintext, not homomorphically). The soundness error of the resulting non-interactive protocol is $O(Qd \log C/p^R)$, where Q denotes the number of random oracle queries.

Multiplicative depth. Including the packing stage, our proof scheme requires a multiplicative depth of 5. However, since the packing of $\mathrm{ct}[h_l(\mathbf{a}, \cdot)]$ and $\mathrm{ct}[h_r(\mathbf{a}, \cdot)]$ is derived from the multiplication inputs of the payload circuit, we only need additional 4 layers of multiplicative depth after the execution of the payload circuit.

6.4 Construction of the Lasagne

Now, we can use the constructed proof system for SIMD executed multiplication-layered circuits to construct our Lasagne VCoED scheme. The construction is relatively straightforward. In the Setup phase, it only needs to generate the public and secret keys of the FHE scheme along with the used random hash function. The ProbGen algorithm simply packs and encrypts the input x following the method described in Section 6.2. In the Computation phase, after homomorphically evaluating the payload circuit using SIMD operations, the server runs the non-interactive proof system as obtained by applying the Fiat–Shamir heuristic in Fig. 4, and outputs the resulting proof. The verification phase then verifies the proof and outputs the verification result.

Our VCoED scheme using d_{slot}-slot FHE for multiplicative layered circuit supporting SIMD operations over d_{pslot} slots (with $d_{pslot}|d_{slot}$), is shown as follows:

Theorem 2 *The protocol puts forward in Fig.5 is a VCoED scheme with client-privacy and soundness.*

proof sketch. Client privacy follows directly from the IND-CPA security of the FHE scheme, and soundness follows directly from the soundness of the underlying proof system.

The case of server-side private input. The above construction only considers the scenario where only the client has input. This scheme can be directly extended to the case where the server also provides input. The main adjustment is to adopt the approach proposed by Phalanx, namely adding plaintext polynomial commitments, to convert the scheme into one with knowledge soundness. Furthermore, by using a zero-knowledge version of the proof system and a circuit-private FHE scheme, client privacy is ensured.

$\mathsf{VC.Setup}(1^\lambda, \mathrm{C})$: Run the key generation algorithm of FHE encryption $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Key.Gen}(1^\lambda)$, and sample a random hash function $hash$. Output $\mathsf{PK} = (hash, \mathsf{pk}), \mathsf{SK} = (hash, \mathsf{sk})$.

$\mathsf{VC.ProbGen}(\mathsf{PK}, x)$: On input $x = (x_0, x_1, \cdots, x_{n-1})$ where $x_i \in \mathbb{F}_p^{d_{pslot}}$, encrypt $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{d_{pslot}, d_{slot}}(\mathsf{pk}, x_i)$ for each $i \in [n]$. Output $\mathsf{Enc}(x) = (\mathsf{ct}_0, \cdots, \mathsf{ct}_{n-1})$ and $\mathsf{st} = x$.

$\mathsf{VC.Compute}(\mathsf{PK}, \mathsf{Enc}(x), \mathrm{C})$: Compute the output of the circuit $\mathsf{Enc}(\mathsf{out}) \leftarrow \mathsf{Eval}_{mod}(\mathsf{pk}, \mathrm{C}(\cdot), \mathsf{Enc}(x))$. Run our non-interactive proof system as obtained by applying the Fiat–Shamir heuristic in Fig. 4, and obtain π. Output $\mathsf{Enc}(\mathsf{out})$ and π.

$\mathsf{VC.Verify}(\mathsf{SK}, \mathsf{Enc}(\mathsf{out}), \mathsf{st}, \pi)$: Check the validness of proof π.

Figure 5: Our Lasagne VCoED scheme

7 Implementation

Experimental Setup. To facilitate a comprehensive evaluation of the computational overhead and practical viability of Lasagne, we benchmark our scheme using a carefully selected set of parameters consistent with prior work [28].

The experiments were conducted on a machine equipped with an Intel Xeon Platinum 8592V processor with 256 logical cores, and and 512 GB of main memory. The underlying Fully Homomorphic Encryption operations are implemented utilizing the BGV scheme provided by the Microsoft SEAL library [29].

We instantiate the BGV ciphertexts with a slot count of 2^{14}, a plaintext modulus of 20 bits, and a ciphertext modulus of 50 bits. The extension parameter for the proof system is set to $R = 8$. To assess the scheme's adaptability to different circuit topologies, we consider two primary types of grid configurations ($\mathcal{D}_i$) for our proof scheme:

- **Time-Optimized Configuration:** Set $\mathcal{D}_i = [4] \times [2] \times \cdots \times [2]$, where the sumcheck protocol terminates when the number of remaining polynomial variates is 10.
- **Size-Optimized Configuration:** Set $\mathcal{D}_i = [4] \times [4] \times \cdots \times [4]$, where the sumcheck protocol terminates when the number of remaining variates is 5.

In our benchmarks, we assume the payload circuit is structured as a d-layer multiplication-layered circuit, where each layer consists of 2^n gates, *half of which are multiplication gates.*

7.1 Microbenchmarks of Basic Operations

To ground our computational complexity models in empirical reality, we first profile the execution time of the fundamental cryptographic operators utilized

in Lasagne. Table 1 summarizes the execution times of base FHE operations, extension field arithmetic, and SIMD manipulations measured in our environment. We accelerate FHE polynomial-vector computations using AVX-512 instructions.

Table 1: Microbenchmarks of Cryptographic Operators

Operator	**Description**	**Time (ms)**
`ct-ct mul`	Ciphertext-ciphertext multiplication	0.045
`pt-ct mul`	Plaintext-ciphertext multiplication	0.149
`scalar mul`	Ciphertext scalar multiplication	0.016
`add`	Ciphertext homomorphic addition	0.008
`rot`	Ciphertext SIMD rotation	0.775
`rel`	Ciphertext relinearization	0.775
`encrypt`	Plaintext encrypted into ciphertext	2.413
`decrypt`	Ciphertext decrypted into plaintext	0.258

These localized microbenchmarks provide the constant coefficients for the concrete cost functions derived in the subsequent section.

7.2 Concrete Computational Overheads

By substituting the empirical microbenchmark timings from Table 1 into the theoretical operation counts of our protocol, we establish a concrete complexity model for the prover. The overall computational workload is fundamentally bifurcated into two parts: the **Per-Layer Packing Phase** and the **Per-Layer Proving Phase**.

1. Per-Layer Packing Phase. This phase aligns the intermediate ciphertexts from the homomorphic evaluation into the specific witness formats required by the proof algorithm for each layer. Assume the payload circuit supports SIMD evaluation with 2^t SIMD width. The computational complexity diverges based on whether the ciphertexts are in NTT or non-NTT form. Depending on the multiplication gate output fraction (i.e., the proportion of output wires that come from multiplication gates), the theoretical analysis yields the following cost function:

Case 1: All output wires of this layer come from multiplication gates

$$Cost_{pack}(n,t) = \underbrace{6 \cdot 2^{n-t} \cdot 0.016}_{\text{NTT form}} \text{ ms} \tag{5}$$

Case 2: Half output wires of this layer come from multiplication gates

$$Cost_{pack}(n,t) = \underbrace{6 \cdot 2^{n-t} \cdot 0.016}_{\text{NTT form}} + \underbrace{5 \cdot 2^{n-1-t} \cdot 0.149}_{\text{non-NTT form}} \text{ ms} \tag{6}$$

2. Per-Layer Proving Phase. The prover must execute the proof algorithm iteratively for each of the d layers of the circuit. By bounding the exact number

of FHE operations per layer, we derive the single-layer proof cost function, denoted as $Cost_{prove}$. Depending on the multiplication gate output fraction and grid topology, the exact theoretical polynomials for $Cost_{prove}$ are as follows:

Case 1: All output wires of this layer come from multiplication gates

- *Time-optimized configuration* (Grid $\mathcal{D}_i = [4] \times [2] \times \cdots \times [2]$):

$$Cost_{prove}^{Time}(n) = 2^{n-15}(6.208n - 22.598) + 976.08n - 6478.096 \text{ ms} \quad (7)$$

- *Size-optimized configuration* (Grid $\mathcal{D}_i = [4] \times [4] \times \cdots \times [4]$):

$$Cost_{prove}^{Size}(n) = 2^{n-15}(9.312n - 75.366) + 1464n - 11256.064 \text{ ms} \quad (8)$$

Case 2: Half output wires of this layer come from multiplication gates

- *Time-optimized configuration*:

$$Cost_{prove}^{Time}(n) = 2^{n-15}(7.616n - 22.122) + 976.08n - 6110.016 \text{ ms} \quad (9)$$

- *Size-optimized configuration*:

$$Cost_{prove}^{Size}(n) = 2^{n-15}(11.424n - 85.45) + 1464n - 10703.944 \text{ ms} \quad (10)$$

3. Total Proving Cost. Finally, the total proving computational cost for a d-layer circuit is determined by assembling the packing phase and the multi-layer proving phase:

$$Cost_{total} = d \cdot Cost_{pack}(n, t) + d \cdot Cost_{prove}(n) \quad (11)$$

A granular, level-by-level breakdown deriving these single-layer proving phase polynomials is extensively documented in Appendix A.

7.3 Proof Performance Evaluation

We project the end-to-end proving time and proof sizes across various circuit scales. Table 2 summarizes the performance, extracting representative data points spanning from million-sized circuits (2^{20}) to massive-sized circuits (2^{34}). For a 2^n-sized payload circuit we set its SIMD width to 2^{n-20}, consistent with Laminate's baseline.

To contextualize our performance gains, Table 3 presents a head-to-head comparison against state-of-the-art VCOED schemes, including Phalanx[5], Blind Fractal, and Laminate (current version), for a standard non-SIMD payload circuit containing 2^{20} gates.

Based on the data from our benchmarks and the SOTA comparison, several architectural advantages of Lasagne emerge:

[5]We consider only the version that takes 2.27 hours, since the 0.68-hour version (Phalanx*) requires all wires inside the circuit to be in NTT form, which is only feasible in certain scenarios.

Table 2: Proof Performance for 16-Layer 2^n-sized payload circuit with 2^{n-20} SIMD width

Size (n)	Time-Optimized Configuration				Size-Optimized Configuration			
	All Mul. Outputs		Half Mul. Outputs		All Mul. Outputs		Half Mul. Outputs	
	Time	Size (MB)	Time	Size (MB)	Time	Size (MB)	Time	Size (MB)
20	5.17 min	45.00	11.53 min	45.00	6.11 min	30.00	12.47 min	30.00
21	5.18 min	45.00	11.67 min	48.75	6.12 min	30.00	12.81 min	33.75
22	5.54 min	52.50	12.05 min	56.25	7.02 min	37.50	13.44 min	37.50
23	6.02 min	60.00	12.60 min	63.75	7.25 min	37.50	14.05 min	41.25
24	6.74 min	67.50	13.44 min	71.25	8.62 min	45.00	15.23 min	45.00
25	7.97 min	75.00	14.94 min	78.75	9.64 min	45.00	16.95 min	48.75
26	10.29 min	82.50	17.81 min	86.25	13.11 min	52.50	20.64 min	52.50
27	14.86 min	90.00	23.56 min	93.75	17.85 min	52.50	27.42 min	56.25
28	24.18 min	97.50	35.31 min	101.25	30.65 min	60.00	42.27 min	60.00
29	43.38 min	105.00	59.58 min	108.75	52.14 min	60.00	1.20 h	63.75
30	1.39 h	112.50	1.84 h	116.25	1.77 h	67.50	2.27 h	67.50
31	2.77 h	120.00	3.58 h	123.75	3.37 h	67.50	4.41 h	71.25
32	5.64 h	127.50	7.20 h	131.25	7.26 h	75.00	9.08 h	75.00
33	11.61 h	135.00	14.72 h	138.75	14.36 h	75.00	18.50 h	78.75
34	23.97 h	142.50	30.29 h	146.25	31.26 h	82.50	38.81 h	82.50

Table 3: Performance Comparison with State-of-the-Art Schemes (2^{20}-sized non-SIMD payload circuit)

Scheme	Proving Time	Proof Size (MB)
Blind Fractal [14]	9.26 h	116
Phalanx [31]	2.27 h	61.4
$\text{Laminate}_{\text{base}}$ [28]	13.9 h	2604
$\text{Laminate}_{\text{fold}}$ [28]	43.9 h	0.26
$\text{Laminate}_{\text{laned}}$ [28]	1.1 h	0.13
Lasagne (TimeOpt)	**5.17˜11.53 min**	**45.00**
Lasagne (SizeOpt)	**6.11˜12.47 min**	**30.00**

- **Practicality and SOTA Domination:** As illustrated in Table 3, Lasagne achieves unprecedented proving efficiency for typical million-gate circuits (2^{20}). Under the Time-Optimized configuration, generating a proof takes only 5.17˜11.53 minutes with a acceptable size of 45.00 MB. This eclipses prior state-of-the-art paradigms, delivering a roughly 12˜26x speedup over Phalanx and a 48˜107x speedup over Blind Fractal, definitively pushing VCOED into the realm of practicality. Even compare with the concurrent work Laminate (current version), our scheme still achieves a 5.7˜12.8× speedup.

- **Flexible Parameter Trade-offs:** Unlike rigid legacy schemes, Lasagne permits developers to dynamically balance computational and communication overheads. By shifting from the Time-Optimized to the Size-Optimized configuration, the system can significantly compress the proof size with a moderate computational penalty. For instance, at $n = 20$, the proof size drops by over 33% (down to 30.00 MB), while the proving time only modestly increases to 6.11˜12.47 minutes. This adaptability is vital for deployment in bandwidth-constrained environments.

- **Scalability to Massive Computations:** When scaling to a billion-gate circuit ($n = 30$), the proving time remains highly feasible for outsourced cloud computing (e.g., 1.33 hours in the Time-Optimized setting). Crucially, the proof size grows logarithmically rather than linearly. Even at the extreme scale of 2^{34} gates (over 17 billion multiplication operations), the proof size caps at merely 82.50 MB (Size-Optimized setting), imposing strictly negligible network transmission overhead for modern network architectures.

8 Application: Verifiable Secure Face Matching

Before detailing our verifiable construction, we outline the secure face matching framework proposed by Boddeti [4], which serves as the payload circuit for our system. To present a comprehensive view, we first describe the functional workflow of the system, followed by its underlying technical implementation.

Workflow. The system operates within a client-server architecture and consists of two primary phases: enrollment and authentication.

- **Enrollment:** The client extracts a normalized representation from a captured facial image. This representation is encrypted using a public key and transmitted to the server's database alongside the user's identity.

- **Authentication:** When a user requests access, the client generates a new probe representation, encrypts it, and transmits it to the server.

- **Encrypted Matching:** The server evaluates the dissimilarity score between the encrypted probe and the stored encrypted templates directly in

the ciphertext domain. The server then returns the encrypted score to the client.

- **Verification:** The client decrypts the received score using their private key to finalize the authentication process.

Similarity Computation. The dissimilarity score between an enrolled template $x \in \mathbb{R}^d$ and a probe $y \in \mathbb{R}^d$ is fundamentally measured using cosine distance. By enforcing L2-normalization on the feature vectors during the feature extraction phase—yielding $\tilde{x} = \frac{x}{||x||}$ and $\tilde{y} = \frac{y}{||y||}$—the complex division operations are eliminated. The distance calculation is simplified to a dot product:

$$d(x, y) = 1 - \frac{x^T y}{||x||||y||} = 1 - \tilde{x}^T \tilde{y} = 1 - \sum_{i=1}^{d} \tilde{x}_i \tilde{y}_i \tag{12}$$

In the plaintext domain, this calculation consists of exactly d scalar multiplications and d scalar additions.

SIMD BGV Batching and Inner Product. To process this in the encrypted domain efficiently, the quantized vectors are packed into SIMD ciphertexts, reducing both memory and computational overhead. The homomorphic evaluation of the inner product circuit proceeds via a rotate-and-sum strategy:

- **Element-wise Multiplication:** The server first computes the Hadamard product of the two packed ciphertexts, executing a single parallel multiplication across all d slots.
- **Logarithmic Accumulation:** To sum the resulting elements across the slots, the system performs a sequence of cyclic rotations. In each step, the ciphertext is cyclically rotated by 2^j slots (for $j = 0, 1, \ldots, \log_2 d - 1$) and added to itself. This tree-based accumulation computes the total sum of the vector elements in logarithmic time.
- **Masking and Subtraction:** The aggregated sum is multiplied by a plaintext mask vector $[1, 0, \ldots, 0]$ to isolate the final inner product in the first slot. Finally, this value is homomorphically subtracted from a plaintext 1 to yield the final distance metric $d(x, y)$.

While this shallow architecture ensures rapid evaluation of the face templates, it is inherently limited by the assumption of an honest-but-curious server. The integration of our VCoED scheme replaces this assumption by cryptographically guaranteeing the integrity of this specific SIMD evaluation circuit (the multiplications, rotations, and additions).

Performance. We further instantiate Lasagne on the verifiable secure face matching application described above, in order to evaluate its practicality on a realistic FHE workload.

In this experiment, the payload circuit performs batched face matching 1,000 facial images from the SphereFace dataset [24]. For each image, the feature extractor outputs a 512-dimensional feature vector. Since all feature vectors are

Table 4: Performance of verifiable secure face matching

Metric	**Result**
Number of face templates	1,000
Feature dimension	512
Number of multiplication gates	2^{19}
BGV slot count	2^{14}
Number of sumcheck rounds	9
Payload evaluation time	356.39 ms
Proof generation time	26.96 s
Verification time	261.82 ms
Proof size	4.25 MB

normalized before encryption, the matching score between a probe vector and an enrolled template can be computed as a 512-dimensional inner product.

In our implementation, the batched payload circuit gives 2^{19} multiplication gates. Including the corresponding addition gates, the overall payload circuit size reaches approximately 2^{20} gates. This scale is representative of million-gate encrypted computations and is sufficiently large to demonstrate the end-to-end efficiency of our VCoED construction.

Furthermore, the payload circuit is a multiplication-layered circuit consisting of two layers: layer 0 contains only addition gates (the mask multiplications are ignored during proof generation because they do not affect outputs' correctness), and layer 1 contains only multiplication gates. Consequently, its overall proof time should be close to that of a single-layer 2^{20}-sized payload circuit (Case 1). Using the estimation method from the previous section, the projected proof time is approximately 16 seconds, and Proof size is approximately 4 MB.

Table 4 reports the performance and proof size of the face matching application. The payload time denotes the homomorphic evaluation time of the batched inner-product circuit 1,000 face templates. The proving time includes the packing phase and all encrypted sumcheck rounds. The verification time includes proof decryption, plaintext consistency checks, and final acceptance verification. While the empirical proof sizes strictly conform to our theoretical models, the measured proving time exhibits an expected overhead compared to the baseline projection. This variance is a well-understood artifact in FHE deployments, primarily stemming from scheduling overheads and the accumulated memory/cache performance differences when operators are executed within a continuous pipeline rather than as isolated microbenchmarks.

Overall, the results demonstrate that Lasagne can prove a practical million-gate FHE payload with prover time on the order of minutes, while keeping client-side verification lightweight. This highlights the strong practical potential of Lasagne for real-world verifiable FHE applications.

References

[1] 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 Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part V*, volume 15488 of *Lecture Notes in Computer Science*, pages 302–334. Springer, 2024.

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

[3] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Annual cryptology conference*, pages 90–108. Springer, 2013.

[4] Vishnu Naresh Boddeti. Secure face matching using fully homomorphic encryption. In *2018 IEEE 9th international conference on biometrics theory, applications and systems (BTAS)*, pages 1–10. IEEE, 2018.

[5] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Proceedings of the 32nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417*, page 868–886. Springer-Verlag, 2012.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325, 2012.

[7] Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. Verifiable computation for approximate homomorphic encryption schemes. In *Annual International Cryptology Conference*, pages 643–677. Springer, 2025.

[8] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2023.

[9] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled psi from fully homomorphic encryption with malicious security. In *Proceedings*

of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 1223–1237, 2018.

[10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[11] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[12] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.

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

[14] Mariana Gama, Emad Heydari Beni, Jiayi Kang, Jannik Spiessens, and Frederik Vercauteren. Blind zksnarks for private proof delegation and verifiable computation over encrypted data. *IACR Cryptol. ePrint Arch.*, page 1684, 2024.

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

[16] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 449–487. Springer, 2024.

[17] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010.

[18] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[19] Shai Halevi and Victor Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020. https://eprint.iacr.org/2020/1481.

[20] Shai Halevi and Victor Shoup. Bootstrapping for helib. *J. Cryptol.*, 34(1):7, 2021.

[21] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 9466–9471, 2019.

[22] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pages 12403–12422. PMLR, 2022.

[23] 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.

[24] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[25] Rasoul Akhavan Mahdavi, Nils Lukas, Faezeh Ebrahimianghazani, Thomas Humphries, Bailey Kacsmar, John Premkumar, Xinda Li, Simon Oya, Ehsan Amjadian, and Florian Kerschbaum. {PEPSI}: Practically efficient private set intersection in the unbalanced setting. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6453–6470, 2024.

[26] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H. P. Fitzek, and Najwa Aaraj. Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, 110(10):1572–1609, 2022.

[27] Samir Jordan Menon and David J Wu. Spiral: Fast, high-rate single-server pir via fhe composition. In *2022 IEEE symposium on security and privacy (SP)*, pages 930–947. IEEE, 2022.

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

[29] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

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

[31] Xinxuan Zhang, Ruida Wang, Zeyu Liu, Binwu Xiang, Yi Deng, Ben Fisch, and Xianhui Lu. Phalanx: An fhe-friendly SNARK for verifiable computation on encrypted data. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pages 4035–4048. ACM, 2025.

[32] 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.

A FHE operations counts

For ease of exposition, we introduce the following notion of operations, where R denotes the extension degree of the proof scheme ($R = 8$):

- ept-ct: R pt-ct mul
- eadd: R add
- esum: $14R$ add and $14R$ rot
- epack: R pt-ct mul and R add
- erel: R rel
- escalar: R scalar mul
- ept-ect: R^2 pt-ct mul and R^2 add
- ect-ect: R^2 ct-ct mul, R^2 add

Let 2^a be the number of multiplication gates in this layer, and 2^b be the number of output wires in the next layer.
Per-Layer Prove phase: For each layer ($D = [4] \times [2] \times \cdots [2]$):

1. Computing t_i in step (3)(a) in Fig.4: $2^a \cdot 2^{-14}$ ct-ct mul(with rel) at level 3, $2^a \cdot 2^{-14}$ ept-ct at level 2, $2^a \cdot 2^{-14}$ eadd and 1 esum at level 1, 1 epack at level 10.

2. The first sumcheck:

   1) \\ first round

   - $6 \cdot 2^a \cdot 2^{-14}$ scalar mul at both level 4 and level 3, $6 \cdot 2^a \cdot 2^{-14}$ add at both level 3 and level 2; \\ compute step(1) in Fig.3, $k \in [9]$.
   - $9 \cdot 2^{a-2} \cdot 2^{-14}$ ept-ct at level 3, $9 \cdot 2^{a-2} \cdot 2^{-14}$ ect-ct(w/o rel) at level 2; $9 \cdot 2^{a-2} \cdot 2^{-14}$ eadd at level 2, and 9 erel at level 2;\\ compute step(2) and (3) in Fig.3.
   - 9 esum and epack at level 1. \\ ciphertext pack

   2) \\ round from 2 to $a - 15$, $a - 16$ rounds in total (case $s - j - 1 \geq t$ except the first round).

   - $3{\cdot}2^a{\cdot}2^{-14}{\cdot}(a{-}16)$ escalar at both level 4 and level 3, $3{\cdot}2^a{\cdot}2^{-14}{\cdot}(a{-}16)$ eadd at both level 3 and level 2;
   - $3 \cdot 2^{a-2} \cdot 2^{-14}$ ept-ect at level 3, $3 \cdot 2^{a-2} \cdot 2^{-14}$ ect-ect(w/o rel.) at level 2; $3(a - 16)$ erel at level 2;
   - $3(a - 16)$ esum and epack at level 1.

   3) round from $a - 14$ to $a - 11$, 4 rounds in total, case $s - j - 1 < t$.

- $2 \cdot 2^a \cdot 2^{-14}$ escalar at level 4, $2 \cdot 2^a \cdot 2^{-14}$ eadd at level 3.
- $2 \cdot 3 \cdot 4$ ept-ect(w rel) at level 3; $2 \cdot 3 \cdot \frac{1+2+3+4}{14}$ esum at level 2; $3 \cdot 4$ ect-ect(w/o rel) at level 2 and $3 \cdot 4$ erel at level 2.
- $3 \cdot 4$ esum and epack at level 1.

4) 2 ept-ect at level 2; 2 esum and epack at level 1.

3. The second sumcheck:

1) $3 \cdot 2^b \cdot 2^{-14}$ scalar mul at level 3, $3 \cdot 2^b \cdot 2^{-14}$ add at level 2; $6 \cdot 2^{b-2} \cdot 2^{-14}$ ept-ct at level $2.6 \cdot 2^{b-2} \cdot 2^{-14}$ eadd at level 1; 6 esum and epack at level 1.

2) $2 \cdot 2^{b-14} \cdot (b-16)$ escalar at level 3, $2 \cdot 2^{b-14} \cdot (b-16)$ eadd at level 2; $2 \cdot 2^{b-2} \cdot 2^{-14}$ ept-ect at level 2; $2(b-16)$ esum and epack at level 1.

3) 2^{b-14} escalar at level 3, 2^{b-14} eadd at level 2; $2 \cdot 4$ ept-ect at level 2; $2 \cdot 4$ esum and epack at level 1.

4) 1 ept-ect at level 2. 1 esum and epack with level 1.

The operations used in the Per-Layer Prove phase for the case $\mathcal{D} = [4] \times [4] \times \cdots \times [4]$ can be listed similarly; we omit them here.

The operation counts are shown in Table **??** and Table 6. By substituting $R = 8$ and $a = b = n-1$ into the workload expressions of Table **??** (and Table 6, respectively), we obtain Eq. 7 (and Eq. 8, respectively); similarly, setting $R = 8$ and $a = b - 1 = n - 1$ yields Eq. 9 and Eq. 10.

Table 5: Computational Workload per Level ($\mathcal{D} = [4] \times [2] \times \cdots \times [2]$)

Level	Operation	Count	Time/op (ms)	Total (ms)
4	scalar mul	$6 \cdot 2^{a-14}$	0.08	$0.48 \cdot 2^{a-14}$
	escalar mul	$2^{a-14}(3(a-16)+2)$	$0.08R$	$(0.24a - 3.68)R \cdot 2^{a-14}$
3	ct-ct mul (w)	2^{a-14}	3.28	$3.28 \cdot 2^{a-14}$
	add	$6 \cdot 2^{a-14}$	0.032	$0.192 \cdot 2^{a-14}$
	ept-ct	$9 \cdot 2^{a-16}$	$0.596R$	$1.341R \cdot 2^{a-14}$
	eadd	$2^{a-14}(3(a-16)+2)$	$0.032R$	$(0.096a - 1.472)R \cdot 2^{a-14}$
	ept-ect	$3 \cdot 2^{a-16} + 24$	$0.628R^2$	$(0.471 \cdot 2^{a-14} + 15.072)R^2$
	scalar	$3 \cdot 2^{b-14} + 6 \cdot 2^{a-14}$	0.064	$0.192 \cdot 2^{b-14} + 0.384 \cdot 2^{a-14}$
	escalar	$2^{b-14}(2(b-16)+1)$ $+ 3 \cdot 2^{a-14}(a-16)$	$0.064R$	$R(0.128b - 1.984)2^{b-14} + R(0.192a - 3.072)2^{a-14}$
2	ept-ct	$2^{a-14} + 1.5 \cdot 2^{b-14}$	$0.447R$	$R(0.447 \cdot 2^{a-14} + 0.6705 \cdot 2^{b-14})$
	ect-ct(w/o)	$9 \cdot 2^{a-16}$	$0.135R$	$0.30375R \cdot 2^{a-14}$
	eadd	$9 \cdot 2^{a-16} + 2^{b-14}(2(b-16)+1)$ $+ 3 \cdot 2^{a-14}(a-16)$	$0.024R$	$R(0.072a - 1.098)2^{a-14} + R(0.048b - 0.744)2^{b-14}$
	erel	$3a - 39$	$2.325R$	$(6.975a - 62.775)R$
	ect-ect(w/o)	$3 \cdot 2^{a-16} + 12$	$0.159R^2$	$(0.11925 \cdot 2^{a-14} + 1.908)R^2$
	esum	$2 \cdot 3 \cdot \frac{10}{14}$	$32.922R$	$140.94R$
	add	$3 \cdot 2^{b-14} + 6 \cdot 2^{a-14}$	0.024	$0.072 \cdot 2^{b-14} + 0.144 \cdot 2^{a-14}$
	ept-ect	$2^{b-16} + 11$	$0.471R^2$	$(0.2355 \cdot 2^{b-14} + 5.181)R^2$
1	eadd	$2^{a-14} + 1.5 \cdot 2^{b-14}$	$0.016R$	$R(0.016 \cdot 2^{a-14} + 0.024 \cdot 2^{b-14})$
	esum+epack	$3a + 2b - 60$	$23.005R$	$(69.015a + 46.01b - 943.205)R$

Table 6: Computational Workload per Level ($\mathcal{D} = [4] \times [4] \times \cdots \times [4]$)

Level	**Operation**	**Count**	**Time/op (ms)**	**Total (ms)**
4	scalar mul	$6 \cdot 2^{a-14}$	0.08	$0.48 \cdot 2^{a-14}$
	escalar mul	$2^{a-14}\left(\frac{9}{2}a - 70\right)$	$0.08R$	$(0.36a - 5.6)R \cdot 2^{a-14}$
3	ct-ct mul (w)	2^{a-14}	3.28	$3.28 \cdot 2^{a-14}$
	add	$6 \cdot 2^{a-14}$	0.032	$0.192 \cdot 2^{a-14}$
	ept-ct	$9 \cdot 2^{a-16}$	$0.596R$	$1.341R \cdot 2^{a-14}$
	eadd	$2^{a-14}\left(\frac{9}{2}a - 70\right)$	$0.032R$	$(0.144a - 2.24)R \cdot 2^{a-14}$
	ept-ect	$3 \cdot 2^{a-16} + 36$	$0.628R^2$	$(0.471 \cdot 2^{a-14} + 22.608)R^2$
	scalar	$3 \cdot 2^{b-14} + 6 \cdot 2^{a-14}$	0.064	$0.192 \cdot 2^{b-14} + 0.384 \cdot 2^{a-14}$
	escalar	$2^{b-14}(3b - 47)$ $+ 2^{a-14}\left(\frac{9}{2}a - 72\right)$	$0.064R$	$(0.192b - 3.008)R \cdot 2^{b-14} + (0.288a - 4.608)R \cdot 2^{a-14}$
2	ept-ct	$2^{a-14} + 1.5 \cdot 2^{b-14}$	$0.447R$	$(0.447 \cdot 2^{a-14} + 0.6705 \cdot 2^{b-14})R$
	ect-ct(w/o)	$9 \cdot 2^{a-16}$	$0.135R$	$0.30375R \cdot 2^{a-14}$
	eadd	$9 \cdot 2^{a-16} + 2^{b-14}(3b - 47)$ $+ 9 \cdot 2^{a-14}(a/2 - 8)$	$0.024R$	$(0.108a - 1.674)R \cdot 2^{a-14} + (0.072b - 1.128)R \cdot 2^{b-14}$
	erel	$\frac{9}{2}a - 45$	$2.325R$	$(10.4625a - 104.625)R$
	ect-ect(w/o)	$3 \cdot 2^{a-16} + 18$	$0.159R^2$	$(0.11925 \cdot 2^{a-14} + 2.862)R^2$
	esum	$\frac{36}{7}$	$32.922R$	$253.692R$
	add	$3 \cdot 2^{b-14} + 6 \cdot 2^{a-14}$	0.024	$0.072 \cdot 2^{b-14} + 0.144 \cdot 2^{a-14}$
	ept-ect	$2^{b-16} + 15$	$0.471R^2$	$(0.2355 \cdot 2^{b-14} + 7.065)R^2$
1	eadd	$2^{a-14} + 1.5 \cdot 2^{b-14}$	$0.016R$	$(0.016 \cdot 2^{a-14} + 0.024 \cdot 2^{b-14})R$
	esum+epack	$\frac{9}{2}a + 3b - 56$	$23.005R$	$(103.5225a + 69.015b - 1633.355)R$