Laminate: Succinct SIMD-Friendly Verifiable FHE

Kabir Peshawaria, Boston University
Zeyu Liu, Yale University
Ben Fisch, Yale University
Eran Tromer, Boston University

December 19, 2025

Abstract

In outsourcing computation to untrusted servers, one can cryptographically ensure privacy using Fully Homomorphic Encryption (FHE) or ensure integrity using Verifiable Computation (VC) such as SNARK proofs. While each is practical for some applications in isolation, efficiently composing FHE and VC into *Verifiable Computing on Encrypted Data (VCoED)* remains an open problem.

We introduce Laminate, the first practical method for adding integrity to BGV-style FHE, thereby achieving VCoED. Our approach combines the blind interactive proof framework with a tailored variant of the GKR proof system that avoids committing to intermediate computation states. We further introduce variants employing transcript packing and folding techniques. The resulting encrypted proofs are concretely succinct: 270 kB, compared to 1 TB in prior work, to evaluate a batch of $B = 2^{14}$ instances of size $n = 2^{20}$ and depth $d = 32$. Asymptotically, the proof size and verifier work is $O(d \log(Bn))$, compared to $\Omega(BN \log n)$ in prior work (for ring dimension N).

Unlike prior schemes, Laminate utilizes the full SIMD capabilities of FHE for both the payload circuit evaluation and proof generation; adds only constant multiplicative depth on top of payload evaluation while performing $\tilde{O}(n)$ FHE operations; eliminates the need for witness reduction; and is field-agnostic. The resulting cost of adding integrity to FHE, compared to assuming honest evaluation, is $\sim$12$\times$ to $\sim$36$\times$ overhead (for deep multiplication-heavy circuits of size 2^{20}), which is $> 500\times$ faster than the state-of-the-art.

Contents

1 Introduction **4**
- 1.1 Challenges in FHE-over-SNARK VCoED . . . 4
  - 1.1.1 Challenges for Server Efficiency . . . 5
  - 1.1.2 Challenges for Client Efficiency . . . 6
- 1.2 Our Contribution . . . 7
- 1.3 Related Work . . . 8
- 1.4 Technical Overview . . . 9
- 1.5 Summary of Evaluation . . . 12
  - 1.5.1 Comparisons with Prior Works . . . 12
  - 1.5.2 Comparisons with Honest Payload Generation . . . 13

2 Preliminaries **13**
- 2.1 (Leveled) Homomorphic Encryption . . . 13
- 2.2 Indexed Languages and Proof Systems . . . 14
- 2.3 Blind Indexed Language . . . 14
- 2.4 Public-Coin Blind Holographic Interactive Oracle Proofs . . . 15
- 2.5 The Blind SNARG Compilation . . . 15
- 2.6 Multilinear Polynomials and Multilinear Extensions . . . 16
- 2.7 Multivariate Sumcheck Protocol . . . 17

3 Requisite hIOP and HE Schemes **18**
- 3.1 GKR Protocol . . . 18
  - 3.1.1 The Layer Reduction Subroutine . . . 19
  - 3.1.2 Round Complexity and Proof Size . . . 20
  - 3.1.3 Leveraging Circuit Structure . . . 22
  - 3.1.4 Algorithmically Dealing with Gate Indicator Functions . . . 22
  - 3.1.5 Other Remarks . . . 23
- 3.2 BGV/BFV FHE Protocol . . . 23

4 GKR-inspired Blind Holographic IOPs **25**
- 4.1 Overview and Prerequisites . . . 25
  - 4.1.1 Class of Supported Payload Circuits . . . 26
  - 4.1.2 Data Layout and Notation . . . 26
  - 4.1.3 The folding operator . . . 27
- 4.2 Warmup: BhIOPs for Rotation-Free Payload Circuits . . . 28
  - 4.2.1 Description of Rotation-Free GKR hIOPs . . . 28
  - 4.2.2 Operation Counts . . . 29
- 4.3 General Case: BhIOPs for Layered Payload Circuits . . . 29
  - 4.3.1 (c, f)-layered GKR hIOP Description . . . 31
  - 4.3.2 Operation Counts . . . 32
- 4.4 Summary: Base vs Folded BhIOP variants . . . 33
- 4.5 Active Slots and Batching . . . 34

5 Achieving Efficient VCoED **35**
5.1 Sub-optimality of Previous Blind SNARG Compilation . . . 35
5.2 Transcript Packing: New BIOP to BIP Compiler . . . 35
5.2.1 Compiler Description . . . 36
5.2.2 Proof of Theorem 5.1 . . . 36
5.2.3 Cost Analysis of Computing Outer SNARK π_{outer} . . . 38
5.3 End to End VCoED from Blind Holographic SNARGs . . . 40
5.4 Family of VCoED Protocols . . . 40
5.4.1 Efficiency Parameters and Notation . . . 40
5.4.2 $\mathsf{Laminate}_{\mathsf{base}}$: Base BhIOP with Trivial Compiler . . . 42
5.4.3 $\mathsf{Laminate}_{\mathsf{fold}}$: Folded BhIOP with Transcript Packing . . . 42
5.4.4 $\mathsf{Laminate}_{\mathsf{pack}}$: Base BhIOP with Transcript Packing . . . 43

6 Evaluation **44**
6.1 Implementation Considerations . . . 44
6.2 Methodology . . . 45
6.3 Comparisons . . . 46
6.4 Overhead Compared to Honest Evaluation . . . 48

A Proof of Lemma 3.7 **56**

B Gruen Section 3 Optimization **57**

C BhIOP Operation Counts **59**
C.1 Base BhIOP Prover FHE Operations for Rotation-Free Payload Circuits . . . 59
C.1.1 Overview and Notation . . . 59
C.1.2 Computing the Layer Reduction Sumcheck Round Polynomial under FHE . . . 60
C.1.3 Total cost of GKR prover under FHE . . . 65
C.2 Folded BhIOP Prover FHE Operations for Rotation-Free Payload Circuits . . . 66
C.2.1 Compute-Only Layer Reduction Operation Counts for Folded BhIOP Prover . . . 66
C.2.2 Total Operation Counts . . . 67
C.3 Base BhIOP Prover FHE Operations for (c, f)-Layered Payload Circuits . . . 68
C.3.1 Operation Counts for Compute-Only Layer Reductions . . . 68
C.3.2 Operation Counts for Forward-Only Layer Reduction . . . 68
C.4 Folded BhIOP Prover FHE Operations for (c, f)-Layered Payload Circuits . . . 68
C.4.1 Operation Counts for Compute-Only Layer Reductions . . . 69
C.4.2 Operation Counts for Forward-Only Layer Reductions . . . 69

1 Introduction

Delegation of expensive computation to cloud services is common nowadays, and raises privacy issues when clients' data is private or sensitive. Fully homomorphic encryption (FHE) [41] is a powerful cryptographic tool for *private* delegation of computation: it enables a server to perform arbitrary computations directly on encrypted data, without learning the underlying plaintext. Over the past decade, a series of breakthroughs [16, 31, 25, 23, 15, 14, 32] improved the efficiency of FHE, making it increasingly practical for some classes of privacy-sensitive delegation applications, including machine learning [51, 47, 52, 48, 29], secure data processing [40], private information retrieval [3, 65, 35, 82, 46], private set intersection [21, 20, 26], oblivious message retrieval [58, 59, 60, 53, 57], and single secret leader election [12, 77].

Despite its power, FHE inherently lacks *integrity guarantees* for computation: it is sound only if the server is honest. A malicious server can arbitrarily alter the computation results by evaluating a circuit of their choice instead of the intended one. Even though the server cannot directly observe the decrypted output, it can for example *choose* that output to be a maliciously-chosen value that harms the client, or some function of the secret inputs that would cause observable client behavior from which the secret can be deduced. To address this, Gennaro, Gentry, and Parno [40] introduced the concept of *Verifiable Computation over Encrypted Data (VCoED)*. VCoED enables computation delegation to an untrusted server while (1) preserving data confidentiality and (2) allowing the client to verify the correctness of the computation. However, achieving efficient VCoED remains an open problem for over a decade.

One approach to realizing VCoED is using cryptographic proofs of FHE evaluation. Specifically, SNARKs (Succinct Non-interactive Arguments of Knowledge), allow the server to produce a succinct proof that its output is indeed the result of applying all the designated ciphertext evaluations correctly.[1] While powerful, this approach suffers from very high proving costs, as known SNARKs inefficiently handle statements about *non-algebraic* ciphertext operations (e.g. relinearization).

A recent line of work [38, 4, 36, 81] explores an alternative approach to VCoED: *FHE-over-SNARK*, which flips the order of composition: it uses FHE to homomorphically compute the SNARK proof in encrypted form. The server first homomorphically evaluates the payload circuit storing ciphertexts representing intermediate wires. It then homomorphically evaluates the circuit of a SNARK prover on the statement corresponding to the desired circuit, and with inputs represented by the above ciphertexts. All these values, as well as the resulting SNARK proof, remain encrypted and hidden from the server. A malicious server could still manipulate the payload circuit evaluation and send an incorrect output, but this will be detected when the client decrypts and verifies the received proof.

The FHE-over-SNARK approach appears to yield much better performance than providing SNARK proofs of the whole homomorphic evaluation — intuitively, because one pays the combined overhead of FHE and SNARK not for computing the payload circuit, but rather, only when homomorphically evaluating the SNARK prover applied to the wires of that circuit. Unlike the payload circuit, the SNARK can be chosen to be very structured and particularly amenable to FHE evaluation. However, substantial challenges remain, as explained next.

1.1 Challenges in FHE-over-SNARK VCoED

While the FHE-over-SNARK paradigm is promising, current realizations [4, 36, 81] remain impractical. They impose expensive overheads on the server and require a high bandwidth client capable of processing large proofs.

[1] SNARG (Succinct Non-Interactive Argument) suffices, as knowledge-soundness is not needed here.

1.1.1 Challenges for Server Efficiency

Inherent challenges stem from the setting of BFV/BGV homomorphic evaluation: multiplicative depth is very costly, the native algebraic structure is constrained, and performance relies on utilizing a very restrictive form of parallelism. Key examples follow.

Witness reduction overheads. An often-overlooked cost in SNARK-based verifiable computation is the witness reduction. After evaluating the desired payload computation and storing all the intermediate values, there can be an additional cost to transform these to the representation that the SNARK prover can handle (sometimes referred to as the *extended witness*).

A prominent case of such overhead occurs due to *mismatched fields and non-native arithmetic*. For instance, a payload computation consisting of a single $\mathbb{F}_q$ addition would intuitively require a proof for a single addition. However, if the SNARK prover operates over a different field $\mathbb{F}_p$ (with $p \neq q$), this is not the case: $x_1 + x_2 \equiv x_3 \pmod{p}$ does *not* imply that $x_1 + x_2 \equiv x_3 \pmod{q}$. Hence, the $\mathbb{F}_p$-based constraint system must emulate $\mathbb{F}_q$ arithmetic, which increases the constraint count and requires many additional witness variables to be homomorphically computed.

This problem is particularly severe in the FHE setting, where non-native arithmetic often requires operations like modular inverse, which are prohibitively expensive in terms of multiplicative depth. For example, computing $x^{-1} \pmod{q}$ via $x^{q-2} \pmod{q}$, via Fermat's Little Theorem, requires $\Theta(\log q)$ iterated modular multiplications, adding significant multiplicative depth to the homomorphic evaluation before even starting the prover evaluation.

Prior FHE-over-SNARK schemes largely overlook this challenge. For instance, [81] fixes the SNARK's base field at 50 bits. This is a poor match for many BFV/BGV applications [15, 14, 32], which use much smaller plaintext fields (e.g., < 20 bits in [58, 60, 35]) for efficiency. Forcing a 50-bit field can slow down the underlying FHE computation by over 20×.[2] Similarly, [36] employs generalized BFV/BGV (GBFV/GBGV) instead of standard variants, but this workaround comes at a high cost: GBFV provides 16× fewer plaintext slots than BFV, reducing evaluation throughput by the same factor.

Inefficient proof generation due to polynomial commitments. Prior works' server efficiency is bottlenecked by reliance on polynomial commitments (PCs) for the extended witness, which contains all intermediate values that arose during payload computation. Concretely, [36, 81] use hashing-based PCs, which are typically computed using the Number Theoretic Transform (NTT). Though an NTT over a length-n vector can be done in $O(n \log n)$ time, this can only be achieved at the cost of $\Omega(\log n)$ multiplicative depth. To achieve multiplicative depth $\leq t$, NTT requires $\Omega(t \cdot n^{1+1/t})$ operations.

For an n-gate payload circuit, the Fractal [24] proof system used in [36] requires an NTT over a length n vector. For [81], the situation is more complicated (see Section 1.1.2), but the NTTs that arise are still large enough to preclude a prover circuit that runs in $\tilde{O}(n)$ with constant multiplicative depth. High multiplicative depth in the prover circuit inflates the noise budget required for the *entire* FHE evaluation, including the original payload circuit. That is, a payload circuit of depth d would now require the FHE scheme to support a total depth of $d + \omega(1)$ by reserving sufficient noise or bootstrapping $\omega(1)$ times. FHE performance is highly sensitive to this total depth parameter.

Loss of FHE Parallelism. Prior schemes [36, 81] use BFV/BGV encryption, which offers parallelism via SIMD (Single Instruction, Multiple Data) computation that concurrently operates on

[2] For example, in [58, 60], the multiplicative depth is $\sim \log p$ for a plaintext space $\mathbb{Z}_p$. Consequently, increasing the plaintext modulus from a ∼20-bit field to a 50-bit field raises the required depth from ∼20 to ∼50. Moreover, a larger plaintext space incurs higher noise growth per level, increasing the total noise budget from approximately $\sim(20 \times 30)$ to $\sim(50 \times 60)$. This alone results in a runtime overhead of at least ∼5×. In addition, maintaining the same security level requires increasing the ring dimension by $\geq 4\times$, implying that the overall runtime overhead exceeds 20×.

N "slots", typically with $N \approx 2^{14}$ or larger. However, all this capacity for parallel computation and storage is allocated to the prover circuit, and specifically to accelerating the expensive polynomial commitment schemes. The actual payload computation is performed in a single slot, without parallelism. Conversely, virtually all practical applications using BFV/BGV FHE (e.g., [58, 35, 21, 20, 26]) rely on SIMD parallelism being fully available and utilized by the payload computation. Reducing these to "single-threaded" execution (by using a single slot for the payload circuit) immediately reduces the system throughput by a factor of tens of thousands, making the total cost of adding integrity exorbitantly expensive even accounting for the additional costs of witness reduction and evaluation of the prover circuit.

1.1.2 Challenges for Client Efficiency

Ideally, VCoED scheme minimizes communication and client verification time. In an attempt to improve server efficiency, current FHE-over-SNARK schemes sacrifice this desideratum.

FHE-friendly polynomial commitments imply larger proofs. Previous attempts to realize VCoED through FHE-over-SNARK were built from proof systems based on the Polynomial IOP (PIOP) framework [17]. These proof systems require committing to a polynomial whose total degree is linear in the size of the extended witness. As discussed above, the requisite polynomial commitments do not admit both a constant multiplicative depth and quasi-linear time prover. To accelerate these commitments, prior works heavily exploit SIMD techniques, i.e. the fact that one BGV/BFV operation performs N operations on the underlying plaintext space (where N is the BGV/BFV ring dimension).

Blind Fractal server [36] has $\Omega(N \log n)$ size proofs and performs $\Omega(tn^{1+\frac{1}{t}}N^{-1})$ operations to compute an NTT on an extended witness vector of length n in multiplicative depth t. Notably, this server only runs in quasi-linear time in a parameter regime where $N = \tilde{\Omega}(n^{1/t})$. Thus, to homomorphically evaluate a Fractal prover circuit in $\tilde{O}(n)$ FHE operations and multiplicative depth $\leq t$, proof size and verifier time is at least $\tilde{\Omega}(n^{1/t})$.

Phalanx [81] improves the server efficiency, but with worse asymptotic implications to its client. It uses the Ligero/Brakedown PCS [2, 43],[3] which rather than performing one NTT of a length n vector, performs $\sqrt{n}$ NTTs over length $\sqrt{n}$ vectors.[4] The consequence is that the [81] server can homomorphically evaluate its polynomial commitment in multiplicative depth t with $\Omega(tn^{1+\frac{1}{2t}}N^{-1})$ FHE operations. However, these commitments require opening proofs of size $\Omega(n^{0.5})$. The proof size and client verification time of [81] are at least $\Omega(N \log n + \sqrt{n})$.

Concretely impractical proof sizes and verification time. Prior works such as [36, 81] discuss parameter regimes that make homomorphically evaluating the SNARK proof more tractable. However, the highlighted parameters are too server-friendly, coming at great cost to the client.

Concretely, [36, 81] benchmark their schemes for payload circuits with $n = 2^{20}$ gates using $N = 2^{14}$ SIMD slots. For this choice, [81] yields proofs of 58 MB, while [36] proof sizes are 116 MB (for a single instance). However, simply writing down all 2^{20} wire assignments requires only 17 MB (for a plaintext modulus $q \leq 2^{128}$). The client would be more memory-efficient, *and faster*, if it just performed the payload computation locally without delegation.

[3] Specifically, the Phalanx server [81] homomorphically evaluates the proof system defined by the Spartan [70] PIOP, compiled with a PCS that is implicit in Ligero [2] and explicitly given in Brakedown [43].

[4] More generally, the Brakedown [43] PCS constructs an $m \times n/m$ matrix and performs an NTT on each n/m sized row. An opening proof consists of m field elements. Phalanx [81] does not explicitly state the chosen matrix dimensions, but the setting $m = \sqrt{n}$ is implied by Section 5.1 stating that the depth-t commitment requires $O(tn^{1+\frac{1}{2t}}) = O(n^{0.5} \cdot tn^{0.5(1+1/t)})$ operations. Also, this PCS can be generalized beyond a matrix, to a tensor of dimension up to $\log n$ [13]; this was not explored by [81].

Thus, in this work, we ask the following question:

Can we build an FHE-over-SNARK construction that:
(1) has a quasi-linear prover circuit with constant multiplicative depth;
(2) achieves succinct proof size and verification time (polylogarithmic in payload circuit size);
(3) natively supports SIMD amortization; and (4) is field-agnostic?

We answer affirmatively by showing such a construction and showing its concrete efficiency.

1.2 Our Contribution

We introduce Laminate, a family of three protocols for Verifiable Computation over Encrypted Data that follow the FHE-over-SNARK paradigm. Laminate protocols perform fully-parallel BFV/BGV homomorphic evaluation of payload circuits, combined with homomorphic evaluation of (a variant of) Goldwasser-Kalai-Rothblum [42] proofs to ensure correctness.[5]

Laminate is the first concrete VCoED scheme to achieve the following properties:

Quasi-linear server circuit with constant multiplicative depth overhead. The multiplicative depth overhead for adding integrity is asymptotically *constant*, and concretely 1.5, 2.5, or 3.5 levels depending on the variant of Laminate (i.e., with a tradeoff against proof size).[6] The resulting noise budget overhead is only 50–120 bits (for $<$ 20-bit plaintext field).

Furthermore, the servers performs $O(n \log n)$ FHE operations *in all parameter regimes*. This is possible because Laminate avoids committing to the extended witness.

Native SIMD Amortization. Laminate preserves the native SIMD capabilities of BFV/BGV, allowing the payload circuit to utilize all N available slots, unlike prior works. To quantify this advantage, comparing[7] against Phalanx [81] with a payload circuit of $n = 2^{20}$ gates and ring dimension $N = 2^{14}$:

- For a batch of 32 instances, all variants of Laminate achieve faster server runtimes.
- At full capacity (a batch of N instances), the $\text{Laminate}_{\text{fold}}$ server is 500× faster and the $\text{Laminate}_{\text{base}}$ server is 1000× faster.

Succinct Proof Sizes and Verifier Efficiency. $\text{Laminate}_{\text{fold}}$, our variant optimized for the client, is succinct, e.g. both proof size and verifier runtime are $O(N+d\log(Nn))$. Moreover, this is achieved while keeping the server circuit constant multiplicative depth and quasi-linear (in FHE operations). The novel technical ingredient is a "transcript packing" technique that is described in Theorem 5.1. No prior work achieved all of the above properties; see a comparison in Table 1.[8]

For the concrete common parameter set considered by prior work (i.e. $n = 2^{20}$ gates, $N = 2^{14}$ slots) where we batch N disjoint instances,[9] $\text{Laminate}_{\text{fold}}$ produces proofs that are $3,500,000\times$ smaller

[5] The name Laminate alludes to the proof generation comprising a thin layer of extra computation, with very low multiplicative depth, that is added on top of the bulk of payload computation to ensure its integrity.

[6] We count scalar-by-ciphertext as 0.5 multiplicative levels, because they consume less noise than plaintext-by-ciphertext or ciphertext-by-ciphertext multiplication. We also assume that extension field multiplication can be done by a depth-1 circuit; otherwise add at most 0.5 levels, per Remark 6.1.

[7] The full end-to-end server runtime advantage is greater than the above bounds, since prior work did not account for a few critical costs; see Section 1.5.

[8] Laminate requires the payload circuit to be *layered*, i.e. gates in layer i take inputs from layer $i+1$. Circuits that arise from natural computations are typically layered, or can undergo an inexpensive transformation to become layered (by adding forwarding dummy gates), as discussed in Section 3.1.5. For payload circuits intended to be evaluated under FHE *without bootstrapping*, Laminate variants support a gate set for which the payload circuit can be viewed as having $d = O(1)$ layers (see Remark 4.6).

[9] For ease of comparison to prior works, we benchmark the setting of using a single SIMD slot per instance, though unlike those works, Laminate can also handle more general payload circuits that use multiple slots for a single instance (including cross-slot rotation gates).

Protocol	Mult Depth Overhead	Server FHE Ops	Proof Size	Use Case
$\mathsf{Laminate}_{\mathsf{base}}$	1.5 Levels	$O(n \log n)$	$O(Nd \log n)$	High Throughput ($k \approx N$)
$\mathsf{Laminate}_{\mathsf{pack}}$	2.5 Levels	$O(n \log n)$	$O(\max\{kd \log n, N\})$	Sparse Computation ($k \ll N$)
$\mathsf{Laminate}_{\mathsf{fold}}$	3.5 Levels	$O(n \log(kn) + d \log(kn) \log k)$	$O(\max\{d \log(kn), N\})$	Bandwidth Constrained
Phalanx [81]	$t+3$ Levels	$\Omega(\frac{k}{N}(n \log n + tn^{\frac{2t+1}{2t}}))$	$\Omega(k\sqrt{n} + kN \log n)$	2PC (private server inputs)
Blind Fractal [36]	$3t+3$ Levels	$\Omega(\frac{k}{N}(n \log n + tn^{\frac{t+1}{t}}))$	$\Omega(kN \log n)$	ZKP Delegation

Table 1: Comparison of VCoED Protocols for single-slot payload circuits with n gates and d layers, batched with $k \in [1, N]$ disjoint instances (where N denotes the BGV/BFV SIMD slot capacity). Lower bounds are used for prior work since some end-to-end costs were omitted in their evaluation, as discussed in Section 1.5. The parameter t (for prior work) denotes a tunable multiplicative depth for evaluating an NTT (or inverse NTT) under FHE. For all protocols, the client's runtime is linear in the proof size.

than Phalanx [81], and $\mathsf{Laminate}_{\mathsf{base}}$ (optimized for server efficiency) produces proofs 365× smaller.

Field-agnostic. Laminate protocols are agnostic of the plaintext field, thereby avoiding additional costs that come with forcing the delegated computation to be defined over a fixed large field and emulating non-native arithmetic.

Conceptual simplicity. The protocol separates logically into two phases: the payload evaluation and the proof evaluation. Crucially, the proof generation accepts the intermediate ciphertexts of the payload evaluation *as is*. This removes the need for witness reduction or extended witness formatting. In its simplest form ($\mathsf{Laminate}_{\mathsf{base}}$), the protocol relies solely on a leveled homomorphic evaluation of the multivariate sum-check protocol, avoiding the complexity of Polynomial Interactive Oracle Proofs (PIOPs).

1.3 Related Work

FHE-over-SNARK. The paradigm of homomorphically generating a proof under FHE for Verifiable Computation on Encrypted Data (VCoED) was first proposed by [38]. A sequence of follow-up works [4, 36, 81] attempt to realize this new paradigm with concrete efficiency. HELIOPOLIS [4] instantiates it using a homomorphic evaluation of the FRI Interactive Oracle Proofs (IOP). Blind Fractal [36] utilizes the Fractal proof system, while Phalanx [81] employs the Spartan polynomial interactive oracle proof (PIOP) system. Notably, these works extend the utility of FHE-over-SNARK to broader cryptographic objectives beyond basic VCoED:

- **ZK Proof Delegation:** Blind Fractal [36] applies this paradigm to delegate the generation of a zero-knowledge proof. A client provides the instance and an encrypted witness to a server, which homomorphically evaluates the ZK-SNARK prover circuit. While the resulting proof is initially a designated-verifier proof, the client can subsequently generate a recursive proof to demonstrate knowledge of the secret key required for verification.
- **Maliciously Secure 2PC:** Phalanx [81] leverages FHE-over-SNARK to achieve maliciously secure Two-Party Computation (2PC). Here, FHE ensures the privacy of the client's inputs, while the server homomorphically evaluates a ZK-SNARK and re-randomizes the resulting ciphertexts. The ZK property ensures the server's private inputs remain hidden from the client, while the soundness of the SNARK guarantees the client receives the correct result.

However, all of these works suffer from the issues discussed in Section 1.1. We compare the concrete efficiencies in more detail in Section 6.

SNARK-over-FHE. Many works have tried to use SNARKs to directly prove that homomorphic operations are done correctly. There is a rich history here of co-designing a SNARK and FHE

combination such that the SNARK can efficiently prove supported ciphertext operations, and supporting operations so that the FHE computation (sans proof) itself is not made much less efficient. The first notable work to try this approach was in [34], which built a SNARK whose constraints are native to a *quotient polynomial ring*, like those used in Ring-LWE based HE constructions, rather than a finite field. A followup work [11] modifies the SNARK to operate over an *unquotiented* polynomial ring, citing gains in prover efficiency. However, constraints of either ring still struggle to capture verifying ciphertext operations such as relinearization were performed correctly. Likewise, [37] struggles to verify these same ciphertext operations, and thus cannot support arbitrary depth computations. Later, [5] solved this limitation, designing a SNARK scheme that can prove FHE (as opposed to SHE) computations. However, these approaches still remain of theoretical interest as costs are impractical for circuits that require multiple levels of multiplicative depth. For further discussion, see the analysis in [76].

Other work [75, 56] proves that an FHE bootstrapping procedure was correctly computed, thus allowing one to prove homomorphic evaluation of unbounded depth. However, proving the TFHE bootstrapping procedure, which already takes ∼10ms to perform honestly, takes > 5 seconds (per circuit gate).

ZFHE [83] proves *in zero-knowledge* that it correctly evaluated an FHE payload circuit. Their primary goal is to hide private server inputs from the client, and their proofs are linear in the circuit size (not succinct). Despite sacrificing succinctness, ZFHE still has at least[10] a two order of magnitude overhead over honest execution of the FHE payload computation.

1.4 Technical Overview

Blind Interactive Oracle Proofs. We first recap the *blind interactive oracle proof* framework from [36], which is essential to understanding our design choices. In a public-coin IOP, the prover sends vector oracles to a verifier who replies with random challenges. Modern SNARKs are built from IOPs following a two step process. First, the vector oracles are instantiated with some vector commitment scheme (e.g. by using Merkle trees, or by sending the entire vector itself as per Remark 2.1). This results in a public-coin interactive proof (IP). Then interaction is removed via the Fiat-Shamir heuristic, i.e., replacing verifier challenges with hashes of the transcript. The key observation in [36] is that a VCoED scheme need not homomorphically evaluate the *entire* SNARK prover (which includes hashing). Instead, the server only needs to homomorphically evaluate the underlying IOP prover circuit.

Vector commitments can be computed *outside* of the FHE context; for instance, one can construct a Merkle tree by treating each ciphertext as a leaf or a block of data to be committed. Similarly, the Fiat-Shamir verifier challenges can be generated externally by hashing the ciphertexts of the running transcript. The intuition is that hashing of these ciphertexts (instead of the plaintexts they encode) remains adequately binding. Thus, this retains the temporal ordering that Fiat-Shamir enforces on the IP execution without requiring expensive homomorphic evaluation of hash functions.

Furthermore, the above implies that the verifier's challenges can be treated as *scalar inputs* to the homomorphic evaluation of the the IOP prover's circuit. This is a significant improvement, as it allows some of the prover's circuit to be evaluated via cheap scalar-ciphertext operations, rather than ciphertext-ciphertext operations that are costly in runtime and noise budget. Furthermore, even though the IOP prover executes multiple rounds, each logically dependent on the previous ones, these rounds do not add up in terms of multiplicative depth, because the hashing of transcript

[10] Note that the extended version of [83] contains corrected benchmarks.

ciphertexts into Fiat-Shamir challenge scalars effectively resets their FHE multiplicative level.[11]

The powerful consequence is that we need only design a *public-coin IOP* with an FHE-friendly prover, as compilation to a blind SNARG adds negligible overhead.

Core Insight: GKR is the right tool. Our core insight is that the GKR IOP is uniquely well-suited for homomorphic evaluation. Most other IOP choices require a *polynomial commitment (PC) under FHE*, where the total degree of the polynomial is linear in the circuit size; such PCs force the server to sacrifice either constant multiplicative depth or quasi-linear runtime (from NTTs). Furthermore, mitigating these costs leads to sacrificing the SIMD parallelism of payload circuit execution, as discussed in Section 1.1.

In contrast, the GKR protocol's IOP prover consists of only two tasks: evaluating multilinear polynomials and computing sumcheck round polynomials.[12] Both of these tasks can be very FHE-friendly, *without* a transformation that incurs a super-logarithmic blowup to the size of the prover circuit to achieve constant multiplicative depth. Multilinear evaluation reduces to a simple inner product, and the sumcheck prover circuit is discussed below. Computing the whole GKR proof then reduces to performing multiple rounds of this form, and as discussed above, the total multiplicative depth needed to homomorphically evaluate all rounds equals that of a *single* round.

FHE-Friendly GKR Sumchecks. The classic, linear-time sumcheck algorithm [28, 73] reuses work from previous rounds. This dependency gives it an $\mathcal{O}(\mu)$ multiplicative depth (where μ is the number of rounds), which is exactly the kind of super-constant depth we are trying to avoid. However, a different, quasi-linear time sumcheck algorithm from [27] offers a crucial work-depth trade-off. In this algorithm, the prover computes each of the μ round polynomials from scratch, without reusing work. While this is less efficient in the plaintext setting (total work is quasi-linear, not linear), its circuit boasts an $O(1)$ multiplicative depth. The observation that *Spartan* sumcheck instances can be computed in low multiplicative depth was used in [81]. In our work, we optimize the prover circuit for *GKR* sumcheck instances (which are more complicated than those from Spartan), carefully tailoring them to be efficient for the blind prover, and attain a multiplicative depth only 2.5.

For a d-layer circuit, the GKR protocol invokes sumcheck d times to reduce a claim about the output layer to one about input layer.[13] The ith sumcheck is taken over a μ_i-variate polynomial for some $\mu_i > \log N$. In Section 4, we describe two variants of the GKR IOP, which we term *base* and *folded* IOPs. The folded IOP is conceptually simple; it completes all μ_i rounds of each sumcheck exactly as prescribed, and requires multiplicative depth 2.5. However, the base IOP is tailored to be maximally FHE-friendly, doing so in two key ways. First, it terminates each sumcheck $\log N$ rounds early, which saves the server some FHE operations. Second, it skips the final steps performed by the folded IOP prover, which can include evaluating a $\log N$-variate multilinear polynomial (which requires one additional level of multiplication). The consequence of these choices is that the base IOP prover achieves a multiplicative depth of only 1.5 at the cost of having the base IOP *verifier* receiving larger proofs and finishing the computations the prover did not complete.[14]

[11] The actual homomorphic evaluation of the prover rounds remains temporally ordered by the need to compute the Fiat-Shamir challenges in sequence.

[12] See Section 2.7 for details on the [64] sumcheck protocol.

[13] These sumchecks are made more efficient by exploiting the "data-parallel" circuit structure [73] present in payload circuits designed for BGV evaluation (see Section 3.1.3).

[14] One can think of the base IOP prover as spawning N threads to compute each round message, but rather than joining the results, it sends each thread's result directly to the base IOP verifier who locally performs aggregation. Aggregation often entails an evaluation of a multilinear extension of the results at a random evaluation point, though occassionally only requires computing a sum of N terms. The "folded" descriptor refers to how a folded prover "folds" the results of its N threads before presenting it to the verifier.

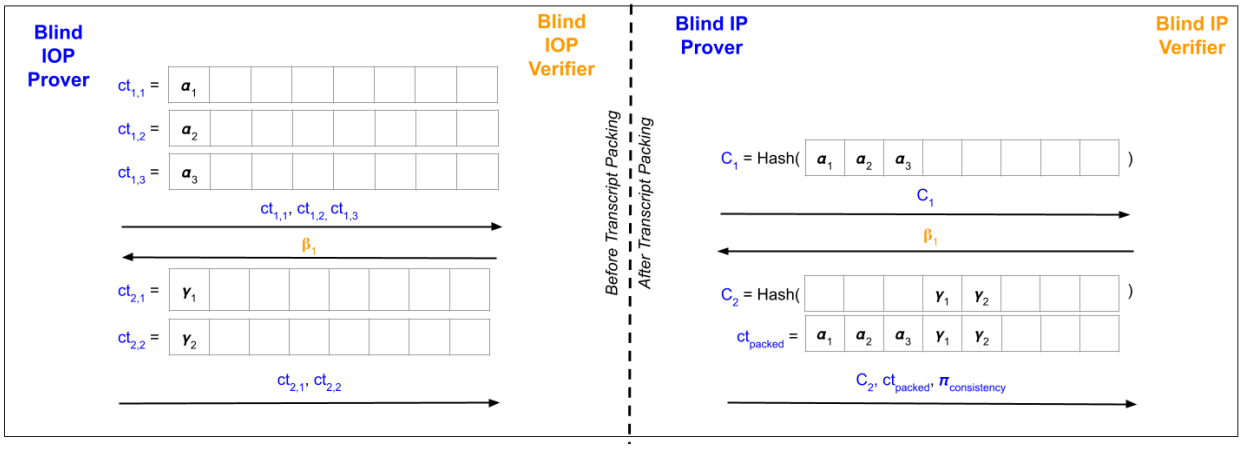

Figure 2: Transcript Packing compilation for a two-round protocol with SIMD capacity $N = 8$. The naive Blind IOP prover sends 5 ciphertexts (40 slots) to convey only 5 useful values. The compiled Blind IP prover aggregates these into a single packed ciphertext (8 slots), supplemented by hash digests and a SNARK proving consistency between the packed ciphertext and the committed preimages.

Challenge: Succinct Proof Sizes. In each round $i \in [r]$, our folded BIOP prover sends an oracle to a length m_i-vector of ciphertexts. When using a plaintext space of roughly 2^{16} targeting $\lambda = 100$ bits of security, this vector consists concretely of $m_i \approx 24$ ciphertexts. The folded verifier needs to learn m_i of the $m_i \cdot N$ communicated field elements - one from each ciphertext. The natural choice would be to compile our folded IOP with the trivial vector commitment scheme, as is typical in GKR-based SNARGs. However, we would then be sending m_i ciphertexts in every round, each of size $7N$ bytes.

For circuits with $d = 32$ layers, $n = 2^{20}$ gates, and $N = 2^{14}$ SIMD slots, there are $r \approx 1500$ total rounds, resulting in $252N$ kilobytes, or 4 GB. This is unacceptable, especially in the extreme case where the payload circuit uses only 1 active slot and we do not have multiple instances to batch.

Solution: Transcript Packing. We instead develop a new Blind IOP to Blind IP compiler. We outline the compiler (we consider only the folded Blind IOP for exposition sake), and a full treatment can be found in Section 5.2.

Our first step is to reduce space consumption in each round. For round $i \in [r]$, let m_i denote the number of field elements sent by the GKR prover. It would be inefficient for a blind prover to send m_i ciphertexts in round i, as it wastes $m_i \cdot (N-1)$ slots. With a lightweight "flattening" procedure (via rotation and addition), we consolidate the m_i ciphertexts into a *single* ciphertext. However, this approach still wastes $N - m_i$ slots *per round*.

Ideally, we would pack *all* round information into a handful of *packed* ciphertexts. However, it is not obvious *a priori* how to do so. Such ciphertexts would need to contain prover messages across *multiple rounds*, but the soundness analysis of IPs require that the verifier *observe* round i messages before sending challenges that are used to compute the round $i+1$ messages. Our solution is to succinctly commit to unpacked ciphertexts in each round, each using unique global slot indices. In the final round, the prover sends just enough "packed ciphertexts" to contain $m := \sum_{i=1}^{r} m_i$ slots of information, consistent with the committed unpacked ciphertexts, along with a SNARK proof of this consistency. Intuitively, these (relaxed) commitments bind the prover to underlying *plaintext* slots, thereby preserving IP soundness. As we discuss in Section 5.2, this "proof of packing" adds negligible overhead to proof size and is tractable for the prover. Application of transcript packing compilation from a blind IOP to a blind IP is depicted in Fig. 2.

When our folded Blind IOP is compiled into a Blind IP using this transcript packing, the

resulting VCoED scheme (after further Fiat-Shamir compilation into a Blind SNARG) is called $\mathsf{Laminate}_{\mathsf{fold}}$. For a payload circuit with $d = 32$ layers, $n = 2^{20}$ gates, $N = 2^{14}$ slots, and plaintext modulus $q = 2^{16} + 1$, $\mathsf{Laminate}_{\mathsf{fold}}$ achieves proof sizes of 270 kilobytes. When the payload circuit is evaluating $B = 2^{14}$ disjoint instances to fully utilize the SIMD capacity of the payload circuit, this implies a *per instance* proof size of only 16.5 *bytes*, in stark contrast to [81]'s per-instance proof size of 60 *megabytes*.

Limitations. We discuss two limitations of our framework. First, our blind proof framework is inherently designated verifier: only a holder of the secret key can verify a blind proof. This contrasts with "SNARK-over-FHE" approaches, where the server proves statements about ciphertexts that anyone can verify. However, as demonstrated in [36], designated-verifier blind proofs can be lifted to public verifiability via proof composition. Second, the "FHE-over-SNARK" paradigm exhibits a fundamental one-bit leakage if the adversary observes the verification result. While this is an inherent trade-off, the alternative SNARK-over-FHE paradigm faces its own security challenges: as discussed in Remark 2.10, our approach offers resistance against known IND-CPA-D key-recovery attacks.

1.5 Summary of Evaluation

1.5.1 Comparisons with Prior Works

The full VCoED scheme incurs the following costs. For simplicity, we focus on the performance of $\mathsf{Laminate}_{\mathsf{fold}}$ (unless otherwise noted), which is a version that focuses on *minimizing the proof size*, and compare all our variants in detail in Section 6 (which may provide better server runtime and lower multiplicative depth).

Homomorphically evaluating the blind SNARG. Following prior works [81, 36], we provide an estimation of the runtime of our construction using detailed operation counts and microbenchmarks. Concretely, for a circuit with 2^{20} gates as in [81, 36], the runtime[15] for evaluating the blind SNARG is $\sim$495$\times$ faster than [81] and $\sim$2012$\times$ faster than [36] when fully utilizing our batching capacity (evaluating 2^{14} independent instances). Given just 2^7 instances, we still outperform by $\sim$3.9$\times$ and $\sim$15.7$\times$ respectively. Moreover, these advantages increase as circuit size grows, since the servers in prior work require more than quasi-linear FHE operations to homomorphically evaluate the blind SNARG under FHE in constant multiplicative depth (due to NTT-based polynomial commitments, see Section 1.1).

Proof size. Laminate's proof size is only $\sim$0.27 MB regardless of the number of instances batched together. For a single instance, this is $> 215\times$ smaller than prior work (60 MB in [81]). For 2^{14} instances, it is $> 3,500,000\times$ smaller.

Homomorphic payload computation. Because Laminate allows parallel execution of independent instances using all $N = 2^{14}$ SIMD slots, it attains a corresponding performance advantage: $2^{14}\times$ more throughput in homomorphic evaluation of the payload circuits, compared to the single-slot payload evaluation in [81, 36].

Input ciphertext blowup. Should prior schemes tune parameters to make their server perform quasi-linear (needed for larger circuits), then they must introduce an overhead of $\Omega(\log n)$ multiplicative depth for evaluating the prover. This necessitates a noise budget increase (concretely: at least 200 extra bits), and thus larger input ciphertexts and correspondingly slower evaluation of the payload circuit. Laminate reduces this overhead to (depending on the variant) between 1.5 and 3.5 multiplicative levels (concretely, 50 to 120 bits of noise budget).

[15] Via operation counts (see Remark 4.1) and microbenchmarks (see Section 6) as prior works.

For example, OMR [58] uses ∼800 bits of noise budget for honest evaluation; thus the ciphertext size overhead is between ∼6.25% and ∼15% with Laminate, compared to > 25% with prior schemes. This advantage further increases as the circuit size grows, since the multiplicative depth of our quasi-linear server circuit is always *constant*, unlike prior works.

Witness reduction. Lastly, our construction does not incur the potentially huge costs of witness reduction, which are inherent (albeit not accounted for) in prior work, as discussed in Section 1.1.

1.5.2 Comparisons with Honest Payload Generation

We now compare our results to *honest payload generation*, where the circuit is homomorphically evaluated without any integrity proof. For a circuit with 2^{20} gates, evaluation time varies with the circuit's topology, specifically the distribution of gate types (multiplications vs. additions) and the multiplicative levels at which they occur. Across all scenarios we have considered, honest evaluation ranges from 0.2 hours to 7.0 hours.

Compared to this honest evaluation, Laminate incurs as little as ∼12× overhead when the circuit is multiplication-heavy and most operations occur near the input layer. Across all tested settings, the overhead remains below 370×, with the worst case arising when the circuit is dominated by additions concentrated near the output layer. This cost includes the additional payload generation overhead incurred by the extra multiplicative depth required by Laminate.

2 Preliminaries

2.1 (Leveled) Homomorphic Encryption

A (leveled) homomorphic encryption (HE) scheme provides four algorithms: $\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval}$. The Eval algorithm allows for public computation on encrypted data. Given a function f and ciphertexts $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}, m_i)$, Eval computes $\mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{evk}, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_w)$ such that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') = f(m_1, \ldots, m_w)$. We use leveled HE, which supports circuits up to a predetermined depth $\mathsf{MAXDEPTH}$ without a costly bootstrapping operation. We denote semantic security by IND-CPA.

Notation. We say that a ciphertext ct encrypts a plaintext x if $x = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ where sk is the client's decryption key, which will usually be implicit.

Let $\mathbb{F}_p$ be the (base) plaintext space for some finite field order p. For plaintexts $\vec{x} \in \mathbb{F}_p^k$, let $\mathsf{ct}[\vec{x}] = (\mathsf{ct}[\vec{x}_1], \ldots, \mathsf{ct}[\vec{x}_k])$ denote a vector of ciphertexts encrypting $\vec{x}$. For any two vectors of ciphertext $\vec{\mathsf{ct}}_1, \vec{\mathsf{ct}}_2$ of size k, we use $\vec{\mathsf{ct}}_1 \times \vec{\mathsf{ct}}_2$ to denote the homomorphic multiplication $\mathsf{Eval}(\mathsf{pp}, \mathsf{evk}, \times, \vec{\mathsf{ct}}_1[i], \vec{\mathsf{ct}}_2[i])_{i \in [k]}$. We similarly define $\vec{x} \times \vec{\mathsf{ct}}$ for plaintext vector $\vec{x}$ and ciphertext vector $\vec{\mathsf{ct}}$. Homomorphic additions $(+)$ are defined analogously.

We also consider an extended plaintext space $\mathbb{F}_{p^R}$ for some $R > 1$. The extended ciphertext (ect) encrypting an extended plaintext $x \in \mathbb{F}_{p^R}$ consists of R ciphertexts. For $\vec{x} \in \mathbb{F}_{p^R}^k$, let $\mathsf{ect}[\vec{x}] = (\mathsf{ect}[\vec{x}_1], \ldots, \mathsf{ect}[\vec{x}_k])$ denote a vector encrypting $\vec{x}$, where $\mathsf{ect}[\vec{x}_i]$ $(i \in [k])$ consists of R ciphertexts. The homomorphic operations $\times$ and $+$ are defined analogously; if any of their inputs is an extended plaintext in $\mathbb{F}_{p^R}$, or the encryption of such a plaintext, then the operation is done in $\mathbb{F}_{p^R}$; otherwise it is $\mathbb{F}_p$.

We use $\mathsf{depth}(\mathsf{ct}[\cdot])$ to denote the maximum depth of the circuit over which a ciphertext can be evaluated (and in particular $\mathsf{depth}(\mathsf{ct}[\cdot]) \leq \mathsf{MAXDEPTH}$ for all ciphertexts).

Leveled HE and FHE. Bootstrapping [41] can transform leveled HE (LHE) to FHE. In all existing FHE constructions, the input ciphertexts have some inherent initial noise, which accumulates with each level of ciphertext evaluation. Bootstrapping is a method that is used to reduce the

accumulated noise in ciphertexts back to some initial level. Unfortunately, all the existing methods of bootstrapping are very costly [50, 39, 62, 66, 61]. Thus, in practice, most applications only use the leveled version of HE, where the depth $\mathsf{MAXDEPTH}$ of the computation is bounded, thereby the focus of this work.

2.2 Indexed Languages and Proof Systems

Indexed Languages. We are concerned with indexed languages $\mathcal{L} = \{\mathcal{L}_i\}$, where an index i (e.g., a circuit) defines a specific language $\mathcal{L}_i$ (e.g., valid input-output pairs). We use preprocessing proof systems where i is preprocessed into a short encoding $e[i]$. A verifier can then use $e[i]$ to check proofs in time sublinear in the size of i.

Interactive Oracle Proofs. Our work builds on public-coin *Holographic Interactive Oracle Proofs (pc-hIOPs)* for indexed languages [6, 24]. In each round, the prover sends the verifier a *vector oracle* and the verifier replies with a uniformly random message. At the end of the protocol, the verifier can query the vector oracles sent by the prover and make a decision to accept or reject the proof.

IOP to SNARG compilation. These information-theoretic protocols can be compiled into *Succinct Non-interactive Arguments (SNARGs)* in two steps. First, by using a *Vector Commitment Scheme (VCS)* to instantiate the vector oracles sent by the IOP prover, we compile an IOP into an interactive proof. Then, after applying the *Fiat-Shamir transformation* in the Random Oracle Model (ROM) [33], the protocol is non-interactive.

Remark 2.1 (The Trivial VCS)**.** When the IOP prover sends vector oracles whose underlying vectors are singletons (or consist of a small handful of elements), a desirable choice is the "trivial" VCS where the commitment is the vector itself. In the trivial VCS, openings are a no-op. The verifier, having received the entire vector m_j as the "commitment," simply reads the entry at index q to verify the "opening."

2.3 Blind Indexed Language

Our goal is not just to prove statements about public data, but to prove statements about *encrypted* data. To do this, we introduce a homomorphic encryption (HE) scheme $\mathcal{E}$. We now formalize the problem we want to solve: proving membership in an indexed language, where both the index and the instance are "blinded" by encryption.

Definition 2.2 ([36])**.** For a given HE scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, and indexed language $\mathcal{L}$, we define the *indexed blind language* $\mathcal{E}[\mathcal{L}]$ as:

$$\mathcal{E}[\mathcal{L}] = \{(i'; x') = ((i, \mathsf{sk}); \mathsf{ct}[x]) : \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}[x]) = x \in \mathcal{L}_i\}$$

For a fixed blind index $i' = (i, \mathsf{sk})$, the *blind language* is $\mathcal{E}[\mathcal{L}_{i'}] = \{x' : (i'; x') \in \mathcal{E}[\mathcal{L}]\}$.

Definition 2.3. For a blind index $i' = (i, \mathsf{sk})$, we refer to i as the *public index* (which is known to all parties) and sk as the *secret index* (which is known only to the verifier).

Lemma 2.4 ([36])**.** *If $\mathcal{E}$ is an IND-CPA secure HE scheme, any PPT adversary has only negligible advantage in distinguishing the underlying $x \in \mathcal{L}_i$ given a corresponding $x' = \mathsf{ct}[x] \in \mathcal{E}[\mathcal{L}_{i'}]$.*

Remark 2.5. This is a slight simplification of the definition in [36]. Specifically, we allow the modify the index i' to include the secret key. Since we are building towards a designated-verifier VCoED scheme, it is okay for only the designated verifier to know the blind language $\mathcal{E}[\mathcal{L}_{i'}]$.

2.4 Public-Coin Blind Holographic Interactive Oracle Proofs

We first define the *interactive* version of a proof system for this blind language. A Blind hOIP (BhIOP) is conceptually a standard hIOP "lifted" into the encrypted domain using HE. The prover receives an evaluation key evk and homomorphically evaluates the prover's algorithm under encryption. The BhIOP verifier uses the secret key sk to simulate the hIOP verifier by decrypting only the queried oracle entries.

Definition 2.6 (Public-Coin Blind hIOP (BhIOP) [36])**.** For a given HE scheme $\mathcal{E}$ and public-coin hIOP π for $\mathcal{L}$, a public-coin blind hIOP $\mathcal{E}[\pi] = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ for the indexed blind language $\mathcal{E}[\mathcal{L}]$ includes the following PPT algorithms:

- $\mathcal{E}[\pi].\mathsf{Setup}(1^\lambda, i)$: For a public index i, this algorithm:
  - Generates $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda)$.
  - Generates the plaintext encoding $e[i] \leftarrow \pi.\mathsf{Ind}(i)$.
  - Outputs $(\mathsf{evk}, (\mathsf{sk}, e[i]))$. The blind index is now $i' = (i, \mathsf{sk})$.
- $\mathcal{E}[\pi].\mathsf{P}_{\mathsf{evk}}(i, x')$: The prover, given i, the encrypted instance $x' = \mathsf{ct}[x]$, and evk, interacts with the verifier. In round $j \in [\mu]$, it receives the "transcript so far" tr' (which contains encrypted messages) and computes the next round message by homomorphically evaluating the plaintext prover $\pi.\mathsf{P}_j$:
$$\mathsf{ct}[m_j] \leftarrow \mathcal{E}.\mathsf{Eval}_{\mathsf{evk}}(\pi.\mathsf{P}_j, (i, \mathsf{ct}[x], \mathsf{tr}'))$$
(Here, tr' is decrypted and re-encrypted as needed inside Eval.)
- $\mathcal{E}[\pi].\mathsf{V}_{\mathsf{sk}}(e[i], x')$: The verifier, given $e[i]$, $x' = \mathsf{ct}[x]$, and sk, interacts with the prover.
  - **Rounds $j = 1 \dots \mu$ (Message Rounds):** $\mathcal{E}[\pi].\mathsf{V}$ receives the ciphertext oracle $\mathsf{ct}[m_j]$ and stores it in its transcript tr'. It then returns a fresh random (scalar) challenge ρ_j.
  - **Round $\mu + 1$ (Decision Round):** $\mathcal{E}[\pi].\mathsf{V}$ first computes $x := \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(x')$. It then executes the algorithm of the plaintext verifier $\pi.V$, providing it with x. When the $\pi.V$ algorithm needs to make a query at a location q:
    - **To the index $e[i]$:** $\mathcal{E}[\pi].\mathsf{V}$ reads $e[i][q]$ from its input and returns it.
    - **To a prover oracle m_j:** $\mathcal{E}[\pi].\mathsf{V}$ retrieves its stored ciphertext oracle $\mathsf{ct}[m_j]$, computes $v := \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}[m_j][q])$, and returns the plaintext value v.

    $\mathcal{E}[\pi].\mathsf{V}$ then outputs the final accept/reject decision of $\pi.V$.

Remark 2.7. We again modify the original definition from [36] for simplicity. This definition is specific to the *public-coin* setting, which is the only setting of practical interest.

2.5 The Blind SNARG Compilation

We compile the public-coin BhIOP from Section 2.4 into a non-interactive blind SNARG using the compilation described in Section 2.2. The VCS chosen for this compilation must support committing to a vector of *ciphertexts* and allow the verifier to request opening proofs to specific positions of the committed vector. The trivial VCS (see Remark 2.1) is a valid choice.

Letting $[\![\rho]\!]$ denote a random oracle, the blind SNARG prover computes the jth verifier challenge as $\rho_j := [\![\rho]\!](\mathsf{tr})$, where the transcript tr contains all public data, including the index i, the encrypted instance x', and all ciphertext messages $\mathsf{ct}[m_k]$ sent so far.

The verifier (who holds sk) runs the non-interactive protocol, decrypts the necessary queries from the proof ciphertexts, and runs the plaintext verifier's logic to accept or reject.

Remark 2.8 (Designated Verifier blind SNARGs)**.** The blind SNARGs described in this manuscript are designated-verifier SNARGs, as the secret key sk is needed for verification. This is shared across all prior works that follow the FHE-over-SNARK paradigm [38, 4, 36, 81].

Remark 2.9 (One-bit leakage for FHE-over-SNARK)**.** As mentioned in prior works [4, 36, 81], there is a one-bit leakage of the payload in the FHE-over-SNARK paradigm. For example, an adversary could replace one bit of the payload with the value 1 and follow the rest of the protocol honestly. The blind proof would pass the check if the replaced bit is 1 and fail otherwise. With access to the verification result, the adversary learns this bit of payload. This is inherent for the FHE-over-SNARK paradigm, since proofs are generated with respect to underlying plaintexts. The SNARK-over-FHE avoids this since the proof attests that each intermediate *ciphertext* was properly generated. On the other hand, the difference in the statements being proven give an advantage to the FHE-over-SNARK paradigm in other settings, as discussed in Remark 2.10.

Remark 2.10 (IND-CPA-D of FHE)**.** While our scheme (or in general any SNARK-over-FHE and FHE-over-SNARK schemes) guarantees integrity for *correctness*, it does not *a priori* rule out a security leakage (e.g., payload leakage) if decrypted plaintexts are leaked (even if all the ciphertexts are guaranteed to be correctly formed, captured by the IND-CPA-D model [54, 55, 1, 8, 18] for FHE). In other words, *even though* the FHE-over-SNARK or SNARK-over-FHE paradigms guarantee that the evaluation process is correct, FHE schemes might still be insecure in the presence of a decryption oracle.

It is worth noting that the FHE-over-SNARK paradigm has an advantage over the SNARK-over-FHE approach in this setting. Specifically, for "exact FHE" schemes such as BGV, BFV, FHEW, and TFHE [15, 14, 32, 31, 25], to our knowledge, all known attacks on their IND-CPA-D security exploit the fact that accumulated errors and bootstrapping (or the accumulated noise) can alter the underlying plaintexts during decryption [22, 19, 63]. Such plaintext changes may occur *even when* the ciphertext evaluations are computed correctly, allowing these attacks to bypass the SNARK-over-FHE paradigm. Intuitively, FHE-over-SNARK can prevent this attack, since the decrypted SNARK proves a statement about the plaintexts directly. We leave a formal exploration of this observation as future work.

2.6 Multilinear Polynomials and Multilinear Extensions

Definition 2.11. A *multilinear polynomial* $f(X_1, \ldots, X_n) \in \mathbb{F}[X_1, \ldots, X_n]$ is linear in each variable.

Fact 2.12. *A multilinear polynomial on n variables can be uniquely defined by its evaluations over the Boolean Hypercube domain $\{0,1\}^n$. As a corollary, any function $h : \{0,1\}^n \to \mathbb{F}$ has a* unique multilinear extension (MLE) *$\tilde{h} : \mathbb{F}^n \to \mathbb{F}$ such that $\forall \vec{v} \in \{0,1\}^n$, $f(\vec{v}) = \tilde{h}(\vec{v})$.*

The equality multilinear polynomial. We define $\tilde{\mathsf{eq}}_n(X_1, \ldots, X_n, Y_1, \ldots, Y_n)$ as the multilinear extension of the function $\mathsf{eq} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\} \subseteq \mathbb{F}$ defined like so:

$$\mathsf{eq}(X_1, \ldots, X_n, Y_1, \ldots, Y_n) = \begin{cases} 1 \text{ if } X_i = Y_i \text{ for all } i \in [n], \\ 0 \text{ otherwise.} \end{cases}$$

The multilinear extension $\tilde{\mathsf{eq}}_n \in \mathbb{F}[X_1, \ldots, Y_n]$ is defined like so:

$$\tilde{\mathsf{eq}}_n[X_1, \ldots, X_n, Y_1, \ldots, Y_n] = \prod_{i=1}^{n} \left(X_i Y_i + (1 - X_i)\,(1 - Y_i)\right)$$

Note that this is multilinear and its Boolean hypercube evaluations agree with the function eq.

Arbitrary Multilinear Extensions. Fix an arbitrary list of values $(a_{\vec{v}})_{\vec{v}\in\{0,1\}^n}$ and let $h : \{0,1\}^n \to \mathbb{F}$ be the function such that $h(\vec{v}) = a_{\vec{v}} \in \mathbb{F}$ for all $v \in \{0,1\}^n$. We express the multilinear extension of h, denoted $\tilde{h}$, via Lagrange Interpolation.

$$\tilde{h}(X_1, X_1, \ldots, X_n) = \sum_{\vec{v}\in\{0,1\}^n} a_{\vec{v}} \cdot \tilde{\mathsf{eq}}_n(X_1, X_2, \ldots, X_n, v_1, v_2, \ldots, v_n) \tag{1}$$

Definition 2.13. We call $(a_{\vec{v}})_{\vec{v}\in\{0,1\}^n}$ the *Lagrange Coefficients* of $\tilde{h}$.

Evaluation in Extension Field. Let $\mathbb{E}$ be an extension field of $\mathbb{F}$. It is well defined to view $\tilde{h}$ as a multilinear polynomial defined over $\mathbb{E}$, and thus we can evaluate it at any point $\vec{\gamma} \in \mathbb{E}^n$. As seen in Equation (1), we can evaluate $\tilde{h}$ at $\vec{\gamma}$ by evaluating each Lagrange Basis polynomial $L_{\vec{v}}(X_1, \ldots, X_n) := \tilde{\mathsf{eq}}(X_1, \ldots, X_n, \vec{v})$ for $\vec{v} \in \{0,1\}^n$ at $\vec{X} = \vec{\gamma}$ and taking an inner product with the Lagrange coefficients.

2.7 Multivariate Sumcheck Protocol

We briefly recall the multivariate sumcheck protocol from [64] as used by modern proof systems. Let $\mathbb{F}$ be a field, and let $\mathbb{E}$ be an extension field of $\mathbb{F}$ that is cryptographically large, i.e. $|\mathbb{E}| \geq 2^\lambda$ for security parameter λ.

The prover has access to a multivariate polynomial $f(X_0, \ldots, X_{\mu-1}) \in \mathbb{F}[X_0, \ldots, X_{\mu-1}]^{\leq d}$ of maximium individual degree d. (Typically d is a small constant, like 3.) The verifier has oracle access to f. The prover wants to convince a distrustful verifier that

$$\sum_{x_0,\ldots,x_{\mu-1}\in\{0,1\}} f(x_0, \ldots, x_{\mu-1}) = s_0 \in \mathbb{F}.$$

in such a way that the verifier need only evaluate f one time (which requires 2^μ times without the prover, since the verifier needs to compute the entire procedure itself).

This takes place over μ rounds. In round $i \in \{0, \ldots, \mu-1\}$, the prover reduces a claim about the sum of $2^{\mu-i}$ values to a claim about the sum of $2^{\mu-i-1}$ values. The protocol proceeds like so. That is, the initial claim is that $\sum_{x_0,\ldots,x_{\mu-1}\in\{0,1\}} f(x_0, \ldots, x_{\mu-1}) = s_0$. Then in round 0, the prover sends a degree d univariate round polynomial $R_0(X) \in \mathbb{F}[X]^{\leq d}$ claiming that

$$R_0(X) = \sum_{x_1,\ldots,x_{\mu-1}\in\{0,1\}} f(X, x_1, \ldots, x_{\mu-1})$$

If the prover is honest, then $R_0(0) + R_0(1) = \sum_{x_0,\ldots,x_{\mu-1}\in\{0,1\}} f(x_0, \ldots, x_{\mu-1})$. So, the verifier checks that $R_0(0) + R_0(1) = s_0$. Next, the verifier samples a uniform random challenge $\gamma_0 \xleftarrow{\$} \mathbb{E}$ and evaluates $s_1 := R_0(\gamma_0)$. The next *round claim* is that $\sum_{x_1,\ldots,x_{\mu-1}\in\{0,1\}} f(\gamma_0, x_2, \ldots, x_n) = s_1$. This process repeats in round $1, \ldots, n-1$ until we are left with a final round claim $f(\gamma_0, \ldots, \gamma_{n-1}) = s_n$

Completeness Intuition If the initial claim is true and the prover follows the protocol, then all subsequent round claims will be true.

Soundness Intuition Suppose in round i the current claim is false. The dishonest prover has two choices. If it sends the "real" round polynomial $R_i(X)$ such that

$$R_i(X) = \sum_{x_{i+1},\ldots,x_{\mu-1}\in\{0,1\}} f(\gamma_0,\ldots,\gamma_{i-1},X,x_{i+1},\ldots,x_{\mu-1})$$

then $R_i(0) + R_i(1) \neq s_i$, and the verifier will reject. If it sends a fake round polynomial, then $R_i(X) \neq \sum_{x_{i+1},\ldots,x_{\mu-1}\in\{0,1\}} f(\gamma_0,\ldots,\gamma_{i-1},X,x_{i+1},\ldots,x_{\mu-1})$, so with probability at least $1-\frac{d}{|\mathbb{E}|}$, $R_i(\gamma_i) \neq \sum_{x_{i+1},\ldots,x_{\mu-1}\in\{0,1\}} f(\gamma_0,\ldots,\gamma_{i-1},\gamma_i,x_{i+1},\ldots,x_{\mu-1})$, in which case the next round claim will also be false. So, the dishonest prover has μ rounds to turn a false claim into a true claim, but in each round, the probability of doing so is at most $\frac{d}{|\mathbb{E}|}$. By Union Bound, a cheating prover fools the verifier with probability at most $\frac{d\mu}{\mathbb{E}}$.

Remark 2.14 (Simple optimization: omit one round polynomial evaluation)**.** Typically, the prover specifies a degree d round polynomial with $d+1$ evaluations (e.g. $R(0),\ldots,R(d)$ if the field has characteristic more than d). However, since the verifier is going to check $R(0)+R(1)=s$, the prover can save time and omit evaluating $R(1)$, as the verifier can simply deduce it. Thus, we will assume that the prover uses d evaluations (e.g. $R(0),R(2),\ldots,R(d)$) to specify degree d round polynomials.

3 Requisite hIOP and HE Schemes

We review the GKR protocol (specifically the refinement of [27] described in [74]) and the BGV/BFV FHE scheme.

3.1 GKR Protocol

Let Ckt be a depth d layered fan-in-two arithmetic circuit over field $\mathbb{F}_q$. By convention, we say L_0 is the list of *output* wires, L_d is the list of *input* wires, and L_i is the list of wires in layer i. The *layered requirement* states that every wire in L_i is the output of an ADD or MUL gate applied to two wires in L_{i+1}. The GKR protocol describes how a prover P convinces verifier V of membership in the language

$$L_{\mathsf{Ckt}} = \{(\mathsf{in},\mathsf{out}) \mid \mathsf{Ckt}(\mathsf{in}) = \mathsf{out}\}$$

with perfect completeness and negligible soundness error.

Formally, the GKR protocol can be described as a public-coin holographic Interactive (Oracle) Proof System $\mathsf{HOL} = (\mathbf{I},\mathbf{P},\mathbf{V})$. The indexer $\mathbf{I}$ processes Ckt in an offline phase as follows.

For each layer $j \in \{0,\ldots,d\}$, it computes and stores $s_j := \log_2(|L_j|)$.[16] Then, for each layer $j \in \{0,\ldots,d-1\}$, it computes gate indicator functions $\mathsf{add}_j, \mathsf{mul}_j : \{0,1\}^{s_j} \times \{0,1\}^{2s_{j+1}}$:

$$\mathsf{add}_j(g,x,y) = \begin{cases} 1 & \text{if } L_j[g] = L_{j+1}[x] + L_{j+1}[y], \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathsf{mul}_i(g,x,y) = \begin{cases} 1 & \text{if } L_j[g] = L_{j+1}[x] \cdot L_{j+1}[y], \\ 0 & \text{otherwise.} \end{cases}$$

Using this, indexer $\mathbf{I}$ creates $2d$ commitments to multilinear extensions $\tilde{\mathsf{add}}_j, \tilde{\mathsf{mul}}_j$ for all $j \in \{0,\ldots,d-1\}$. The index i stores the s_j's (log size of each layer) and the $\mathsf{add}_j, \mathsf{mul}_j$ indicator

[16] For ease of exposition, we will assume all layers have power-of-two many wires. If this is not the case, one can pad with dummy zero wires.

GKR Online Phase

1. **Initial Claim:** The protocol begins with the prover's initial claim that the circuit output matches the provided output: $L_0 \stackrel{?}{=} \mathsf{out}$.
2. **Output Randomization:** The verifier V samples a random challenge $\gamma \in (\mathbb{F}_{q^R})^{s_0}$ and sends it to the prover P. The claim is updated to be an evaluation of the multilinear extension:

$$\tilde{L}_0(\gamma) \stackrel{?}{=} \tilde{\mathsf{out}}(\gamma)$$

   By the Schwartz-Zippel lemma, this step incurs a soundness loss of at most $\frac{s_0}{|q^R|}$.
3. **Layer Reduction Loop:** For each layer $i = 0, \ldots, d-1$:
   - P and V engage in a $2s_{i+1}+1$ round "layer reduction" subroutine.
   - This subroutine reduces the claim about $\tilde{L}_i$ (from the previous step) to a new claim of the form $\tilde{L}_{i+1}(r_a) + \alpha\tilde{L}_{i+1}(r_b) \stackrel{?}{=} v$ for some r_a, r_b, α, v derived during the subroutine.
   - This reduction step incurs a soundness loss of $\frac{8s_{i+1}+1}{q^R}$.
4. **Final Check:** After the loop finishes (at $i = d-1$), the verifier holds a final claim about the input layer L_d:

$$\tilde{L}_d(r_a) + \alpha\tilde{L}_d(r_b) \stackrel{?}{=} v$$

   The verifier can check this claim directly using the known circuit inputs in.

Figure 3: GKR Online Phase

functions. The encoded index $e[i]$ stores the values $\{s_j\}_{0\le j\le d}$ and the multilinear commitments to $\tilde{\mathsf{add}}_j, \tilde{\mathsf{mul}}_j$ for $j \in \{0, \ldots, d-1\}$.

Remark 3.1. For the context of this paper, we will make a simplifying assumption that the verifier has access to $\mathsf{add}_j, \mathsf{mul}_j$, rather than commitments. We will justify this assumption further in Section 3.1.3.

After the offline phase, the prover and verifier engage in an interactive online phase, given in Figure 3 using the subroutines below.

3.1.1 The Layer Reduction Subroutine

Recall from the high-level overview that at the beginning of round i (for $i \in \{0, \ldots, d-1\}$), the verifier V holds a single claim about layer i. For $i > 0$, this claim is a batched evaluation:

$$\tilde{L}_i(\vec{r}_a) + \alpha\tilde{L}_i(\vec{r}_b) \stackrel{?}{=} v_i$$

where $\vec{r}_a, \vec{r}_b$ and α are challenges from the previous round's reduction. The layer reduction subroutine reduces this to a new, single claim about layer $i+1$.

This reduction exploits the following fundamental identity, which relates the evaluation of $\tilde{L}_i$ at any point $\vec{r}$ to a sum over all possible input wire pairs from layer $i+1$.

$$\tilde{L}_i(\vec{r}) = \sum_{\vec{x}\in\{0,1\}^{s_{i+1}}} \sum_{\vec{y}\in\{0,1\}^{s_{i+1}}} g^{(i)}(\vec{r}, \vec{x}, \vec{y}) \tag{2}$$

$$\begin{aligned}\text{where } g^{(i)}(\vec{r}, \vec{x}, \vec{y}) := {} & \tilde{\mathsf{add}}_i(\vec{r}, \vec{x}, \vec{y}) \left(\tilde{L}_{i+1}(\vec{x}) + \tilde{L}_{i+1}(\vec{y})\right) \\ & + \tilde{\mathsf{mul}}_i(\vec{r}, \vec{x}, \vec{y}) \left(\tilde{L}_{i+1}(\vec{x}) \cdot \tilde{L}_{i+1}(\vec{y})\right)\end{aligned}$$

Applying this identity to the verifier's batched claim, we get:

$$\begin{aligned} v_i &\stackrel{?}{=} \tilde{L}_i(\vec{r}_a) + \alpha \tilde{L}_i(\vec{r}_b) \\ &= \sum_{\vec{x},\vec{y}} g^{(i)}(\vec{r}_a, \vec{x}, \vec{y}) + \alpha \sum_{\vec{x},\vec{y}} g^{(i)}(\vec{r}_b, \vec{x}, \vec{y}) \\ &= \sum_{\vec{x}\in\{0,1\}^{s_{i+1}}} \sum_{\vec{y}\in\{0,1\}^{s_{i+1}}} \left(g^{(i)}(\vec{r}_a, \vec{x}, \vec{y}) + \alpha g^{(i)}(\vec{r}_b, \vec{x}, \vec{y}) \right) \end{aligned}$$

Define the batched polynomial for the sumcheck as:

$$g^{(i)}_{\text{batch}}(\vec{x}, \vec{y}) := g^{(i)}(\vec{r}_a, \vec{x}, \vec{y}) + \alpha g^{(i)}(\vec{r}_b, \vec{x}, \vec{y}) \tag{3}$$

The claim v_i is now equivalent to $\sum_{\vec{x},\vec{y}} g^{(i)}_{\text{batch}}(\vec{x}, \vec{y}) \stackrel{?}{=} v_i$. The prover and verifier use the sumcheck protocol to verify this, as can be seen in Figure 4.

Remark 3.2. The first layer reduction ($i = 0$) is a special case. The initial claim is $\tilde{L}_0(\vec{\gamma}) \stackrel{?}{=} v_0$ (where $v_0 = \tilde{\mathsf{out}}(\vec{\gamma})$). This can be seen as the general form $\tilde{L}_0(\vec{r}_a) + \alpha \tilde{L}_0(\vec{r}_b) \stackrel{?}{=} v_0$ by setting $\vec{r}_a = \vec{\gamma}$, $\alpha = 0$, and $\vec{r}_b$ to an arbitrary vector (e.g., $\vec{0}$). The verifier's check in Step 2 simplifies accordingly, as $v_a = \tilde{\mathsf{add}}_0(\vec{\gamma}, \vec{\rho}, \vec{\phi})$ and $v_m = \tilde{\mathsf{mul}}_0(\vec{\gamma}, \vec{\rho}, \vec{\phi})$.

Theorem 3.3 ([27, 74])**.** *The prover of the GKR refinement described in [27, 74] can be run in quasi-linear time with respect to the size of the circuit. Furthermore, the proof size and verifier time are linear in the number of GKR rounds.*

Remark 3.4. Libra [78] describes a GKR variant that can be run in *linear* time, but the approach is not FHE-friendly.

3.1.2 Round Complexity and Proof Size

The GKR protocol as described above is an r-round holographic IOP that is information-theoretically succinct, where $r = \sum_{i=1}^{d}(2s_i+1)$. Instantiating the vector oracles with a trivial vector commitment scheme, we view GKR as a holographic IP. In each round, the prover sends at most three extension field elements. We give a convenient upper bound on GKR proof size, that depends only on the total number of gates n and the depth of the circuit d.

Claim 3.5. *For a circuit of depth d and n gates, the number of GKR rounds is upper bounded by*

$$d\left(2\log\left(\frac{n}{d}\right) + 1\right)$$

Proof. The round complexity is defined as $r = \sum_{i=1}^{d}(2s_i + 1)$. Substituting $s_i = \log S_i$, we rewrite this sum as:

$$r = 2\sum_{i=1}^{d} \log S_i + d.$$

By definition, we know that $\sum_{i=0}^{d} S_i = n$, which implies $\sum_{i=1}^{d} S_i \leq n$. Since the logarithm function is concave, by Jensen's inequality, the term $\sum_{i=1}^{d} \log S_i$ is maximized when all S_i are equal (i.e., $S_i = n/d$). Therefore:

$$\sum_{i=1}^{d} \log S_i \leq d \cdot \log\left(\frac{n}{d}\right).$$

GKR Layer Reduction ($i \to i+1$)

1. **Sumcheck:** P and V engage in a $2s_{i+1}$-round sumcheck protocol to prove the claim:

$$\sum_{\vec{x}\in\{0,1\}^{s_{i+1}}} \sum_{\vec{y}\in\{0,1\}^{s_{i+1}}} g^{(i)}_{\text{batch}}(\vec{x},\vec{y}) \stackrel{?}{=} v_i$$

   At the end of the sumcheck, V has sampled random challenges $\vec{\rho},\vec{\phi} \in (\mathbb{E})^{s_{i+1}}$ and P has provided a final claimed value v_{sum}. The original claim is reduced to a new claim:

$$g^{(i)}_{\text{batch}}(\vec{\rho},\vec{\phi}) \stackrel{?}{=} v_{\mathsf{sum}}$$

2. **Evaluate and Send:** The verifier will check this new claim. By factoring $g^{(i)}_{\text{batch}}$, it becomes:

$$\begin{aligned} v_{\mathsf{sum}} \stackrel{?}{=} &\left(\tilde{\mathsf{add}}_i(\vec{r}_a,\vec{\rho},\vec{\phi}) + \alpha\tilde{\mathsf{add}}_i(\vec{r}_b,\vec{\rho},\vec{\phi})\right)\cdot\left(\tilde{L}_{i+1}(\vec{\rho}) + \tilde{L}_{i+1}(\vec{\phi})\right) \\ &+\left(\tilde{\mathsf{mul}}_i(\vec{r}_a,\vec{\rho},\vec{\phi}) + \alpha\tilde{\mathsf{mul}}_i(\vec{r}_b,\vec{\rho},\vec{\phi})\right)\cdot\left(\tilde{L}_{i+1}(\vec{\rho})\cdot\tilde{L}_{i+1}(\vec{\phi})\right)\end{aligned}$$

   To verify this, the parties proceed as follows:
   - V computes the batched gate evaluations *locally* (using the assumption in Remark 3.1):

$$v_a \leftarrow \tilde{\mathsf{add}}_i(\vec{r}_a,\vec{\rho},\vec{\phi}) + \alpha\tilde{\mathsf{add}}_i(\vec{r}_b,\vec{\rho},\vec{\phi}); \qquad v_m \leftarrow \tilde{\mathsf{mul}}_i(\vec{r}_a,\vec{\rho},\vec{\phi}) + \alpha\tilde{\mathsf{mul}}_i(\vec{r}_b,\vec{\rho},\vec{\phi})$$

   - P computes and sends the two values V does not know:

$$v_0 \leftarrow \tilde{L}_{i+1}(\vec{\rho}); \qquad v_1 \leftarrow \tilde{L}_{i+1}(\vec{\phi})$$

3. **Verifier's Local Check:** V checks that the values provided by P are consistent with the sumcheck result.

$$V \text{ checks: } v_{\mathsf{sum}} \stackrel{?}{=} v_a\,(v_0+v_1) + v_m\,(v_0\cdot v_1)$$

   If this check fails, V rejects. Otherwise, V is now convinced of the $i \to i+1$ reduction *if and only if* P's two claims from the previous step are true.
4. **Batching:** To reduce to one claim, V samples and sends batching challenge $\alpha' \in \mathbb{E}$ to P.
5. **Resulting Claim:** The two claims are combined into a single, new claim about layer $i+1$:

$$\tilde{L}_{i+1}(\vec{\rho}) + \alpha'\tilde{L}_{i+1}(\vec{\phi}) \stackrel{?}{=} v_0 + \alpha' v_1$$

   This new claim becomes the input for the next layer reduction (for layer $i+1 \to i+2$).

Figure 4: GKR Layer Reduction ($i \to i+1$)

Substituting this back into the expression for r:

$$r \le 2\left(d\cdot\log\left(\frac{n}{d}\right)\right) + d = d\left(2\log\left(\frac{n}{d}\right)+1\right).$$

□

Lemma 3.6. *For a circuit of depth d and n gates, the proof size of the GKR hIP is upper bounded by $3d\left(2\log\left(\frac{n}{d}\right)\right)+2d$ extension field elements.*

Proof. In the proof of Claim 3.5, we showed that the GKR protocol has at most $r = d\left(2\log\left(\frac{n}{d}\right)\right)$ sumcheck rounds and d other rounds. In each sumcheck round, the prover sends a degree 3 univariate sumcheck round polynomial. Typically, this would require the sumcheck prover to send four evaluations to uniquely specify this round polynomial. However, the simple optimization described

in Remark 2.14 allows the omission of one evaluation. In the d non sumcheck rounds, the prover sends two multilinear evaluations. This requires two extension field elements.

The claimed upper bound on total proof size follows. □

3.1.3 Leveraging Circuit Structure

When the circuit possesses structural properties—common in real-world applications—the GKR protocol achieves significantly higher efficiency.

Data-Parallel Circuits. As observed by [73], circuits with regular wiring patterns admit a more efficient layer reduction subroutine. For these *data-parallel circuits*, the ith layer reduction step can be optimized from a sparse sumcheck over $2s_{i+1}$ variables to a dense sumcheck over s_i variables.

MLE-Structured Functions. Standard multilinear evaluation requires an inner product between Lagrange coefficients and basis polynomial evaluations, as shown in Equation (1). However, certain functions possess structure that permits more efficient evaluation of their multilinear extensions. In [71], these are termed *MLE-structured* functions. For example, consider the function $\mathsf{SOME} : \{0,1\}^n \to \{0,1\} \subseteq \mathbb{F}$ defined by

$$\mathsf{SOME}(X_1, \ldots, X_n) = \begin{cases} 0 & \text{if } X_1 = \ldots = X_n = 0, \\ 1 & \text{otherwise.} \end{cases}$$

Its multilinear extension is $\tilde{\mathsf{SOME}}(X_1, \ldots, X_n) = 1 - \prod_{j=1}^{n}(1 - X_j)$, which can be evaluated in $O(n)$ field operations rather than $O(2^n)$.

MLE-Structured Gate Indicators. In Remark 3.1, we assumed the verifier had oracle access to multilinear extensions of the gate indicator functions $\mathsf{add}_j, \mathsf{mul}_j$. For circuits where these indicators are MLE-structured, we can remove this assumption. The indexer $\mathbf{I}$ simply provides the verifier with a succinct arithmetic circuit describing the evaluation of the indicator's MLE; the verifier then runs this locally whenever an evaluation is required.

Unstructured Circuits via Commitments. In the general case where gate indicator functions are *not* MLE-structured, we must justify the assumption in Remark 3.1 differently. The indexer instead provides the verifier with binding multilinear commitments to the gate indicators. During layer reduction, when the verifier queries $\tilde{\mathsf{add}}_i$ or $\tilde{\mathsf{mul}}_i$, the prover supplies the claimed evaluation. As a post-processing step, the prover provides opening proofs (with respect to the binding commitments) to validate all evaluation claims made during the protocol.

Field-Agnostic Sparse Commitments. To handle the general case efficiently, we require a multilinear commitment scheme where prover costs scale with the sparsity (number of non-zero entries) of the polynomial (otherwise, the prover would incur an $\Omega(n^2)$ cost). Spark [70, 71] provides a sumcheck-based compiler that transforms a dense multilinear commitment scheme into a sparse one, preserving the desired prover efficiency. Instantiating Spark with a field-agnostic PCS, such as BaseFold [79], satisfies these requirements.

3.1.4 Algorithmically Dealing with Gate Indicator Functions

A naive implementation of the layer reduction sumcheck appears to require $\Omega((S_{i+1})^2)$ work, as the sum is over $2^{2s_{i+1}} = (S_{i+1})^2$ terms. Indeed, if any layer L_{i+1} is *fat* (i.e., has $S_{i+1} = \Omega(n)$ wires), this implies an $\Omega(n^2)$ cost, which is far from the (quasi)-linear time claim.

The key to an efficient prover is that the gate indicator functions add_i and mul_i are **sparse**. By definition, there are at most S_i gates in layer i, so the functions add_i and mul_i can have at most

S_i non-zero entries in total. This sparsity in the Lagrange-coefficient domain is inherited by their multilinear extensions, $\tilde{\mathsf{add}}_i$ and $\tilde{\mathsf{mul}}_i$.

The GKR sumcheck prover's main task is to compute the round polynomials $R_j(X)$. This, in turn, requires evaluating the $g^{(i)}_{\text{batch}}$ polynomial, which depends on $\tilde{\mathsf{add}}_i$ and $\tilde{\mathsf{mul}}_i$. A standard technique (used in [27] and beyond) allows us to compute the evaluations of these sparse MLEs at each round j of the sumcheck.

Specific to GKR, a consequence of the gate indicators' sparsity is the following lemma, whose proof we defer to Appendix A.

Lemma 3.7. *Fix a layer reduction $i \in \{0, \ldots, d-1\}$. The prover can compute (sparse) tables $\hat{A}_j$ (and $\hat{M}_j$) at the beginning of the jth round of sumcheck such that for layer challenge $\vec{r} \in \mathbb{F}_{q^R}^{s_i}$, sumcheck challenges $\vec{\gamma} = (\gamma_0, \ldots, \gamma_{j-1}) \in \mathbb{F}_{q^R}^{j}$, $X \in \{0, 2, 3\}$, and $w \in \{0,1\}^{2s_{i+1}-j-1}$*

$$\hat{A}_j[X][w] = \tilde{\mathsf{add}}_i(\vec{r}, \vec{\gamma}, X, \vec{w})$$
$$\hat{M}_j[X][w] = \tilde{\mathsf{mul}}_i(\vec{r}, \vec{\gamma}, X, \vec{w})$$

Moreover, the prover requires $O(S_i \cdot (s_i + s_{i+1}))$ $\mathbb{F}_q$-ops in total across all sumcheck rounds.

3.1.5 Other Remarks

It is natural to wonder why we present the refinement of [27] as described in [74], which has a quasi-linear time prover, rather than presenting [78], which admits a linear-time GKR prover. We do so because the approach taken in Libra is not FHE-friendly.

The reader may also wonder why we restrict ourselves to layered circuits, when a GKR variant described in [80] supports general arithmetic circuits. Our reason is two-fold. First, the construction of [80] is more complicated. Second, when applied to a circuit that is already layered, the prover of [80] suffers an additional logarithmic factor slowdown.[17] Most real-world circuits are naturally layered, and there is a generic transformation to layer a general circuit. Although in the worst case, this transformation can blowup the total number of wires by a factor of circuit depth d, in practice this rarely happens lest the circuit is contrived. Thus, we make a judgement call to not discuss GKR for general arithmetic circuits here, and encourage the reader to review [80] should they need.

3.2 BGV/BFV FHE Protocol

The BGV [15] and BFV [14, 32] line of works are popular choices of leveled homomorphic encryption schemes that are both compatible with the Laminate protocols we will present in Section 5. While we overview the BGV construction below, we remark that BFV is very similar.

Given a polynomial y that lives in the ring $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N + 1)$, the BGV scheme encrypts y 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. We assume that t satisfies that $t \equiv 1 \mod 2N$, where N is a power of two, which is typical in practice [7, 68].

Plaintext encoding. To encrypt a plaintext $\vec{m} = (m_1, \ldots, m_N) \in \mathbb{Z}_t^N$, BGV creates polynomial $m(X) = \sum_{i \in [N]} m_i X^{i-1}$, and then *encodes* it by constructing *another* polynomial $y(X) = \sum_{i \in [N]} y_i X^{i-1}$ where $m_i = y(\eta_j)$, for some predetermined points η_j. Such encoding can be done using an Inverse Number Theoretic Transformation (INTT).

[17] In practice, we would not use the [80] protocol as described in its full generality for an already layered circuit.

Encryption and decryption. The BGV ciphertext encrypting $\vec{m}$ under $\mathsf{sk} \leftarrow$ dist has the format $\mathsf{ct} = (a, b) \in \mathcal{R}_Q^2$, satisfying $b - a \cdot \mathsf{sk} = y + t \cdot e$ where $\lfloor Q/t \rfloor \cdot y \in \mathcal{R}_Q$ and y is the polynomial encoded in the manner above, and e is some small error term sampled from some Gaussian distribution over $\mathcal{R}_Q$.

Symmetric key encryption can be done by sampling a random $a \in \mathcal{R}_Q$ and constructing $b \in \mathcal{R}_Q$ accordingly using sk. Public key encryption can also be achieved easily but it is not relevant to our paper so we refer the reader to [14, 32, 49] for details.

Decryption is calculated via $y' \leftarrow \lceil (t/Q) \cdot (b - a \cdot \mathsf{sk}) \rfloor \in \mathcal{R}_t$ (note that $(b - a \cdot \mathsf{sk})$ is done over $\mathcal{R}_Q$), followed by a decoding process (which is also a linear transformation). Henceforth, we assume $\mathsf{BGV.Dec}$ outputs plaintext $\in \mathbb{Z}_t^N$, which is the decoded form, for simplicity. Similarly, we assume $\mathsf{BGV.Enc}$ contains the encoding process.

BGV operations. BGV essentially supports addition, multiplication, and rotation, satisfying the following property:

- (Addition) $\mathsf{BGV.Dec}(\mathsf{ct}_1 + \mathsf{ct}_2) = \mathsf{BGV.Dec}(\mathsf{ct}_1) + \mathsf{BGV.Dec}(\mathsf{ct}_2)$
- (Scalar multiplication) $c \times \mathsf{BGV.Dec}(\mathsf{ct}) = \mathsf{BGV.Dec}(c \times \mathsf{ct})$ for $c \in \mathbb{Z}_t$
- (Plaintext multiplication) $\vec{c} \times \mathsf{BGV.Dec}(\mathsf{ct}) = \mathsf{BGV.Dec}(\vec{c} \times \mathsf{ct})$ for $\vec{c} \in \mathbb{Z}_t^N$
- (Ciphertext 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) $\mathsf{BGV.Dec}(\mathsf{rot}(\mathsf{ct}, j))[i] = \mathsf{BGV.Dec}(\mathsf{ct})[i + j \pmod N]$ for all $i, j \in [N]$.[18]

We assume all necessary evaluation keys are correctly and implicitly taken. All operations are operated over the entire plaintext vector $m \in \mathbb{Z}_t^N$, element-wise. Thus, all messages need to be evaluated using the same operation by default: i.e., the additions and multiplications are simply done element-wise. This is also known as the Single Instruction Multiple Data (SIMD) [72] property of BGV.

Since we use all of these operations as blackboxes, we omit the details of their realization and refer the reader to [15, 14, 32, 49].

Multiplicative depth. One important aspect of BFV/BGV is the multiplicative depth. To evaluate a circuit with k levels of multiplications, the runtime is essentially $\Omega(k)$ per operation, unless bootstrapping is used (which is often too expensive to use in practice). The main reason is that each operation enlarges the input RLWE ciphertext noise by a certain amount. When the noise crosses a certain threshold $(\frac{Q}{2t})$, the correctness guarantee is broken. Thus, when generating the initial (fresh) ciphertexts, one needs to account for the final noise, such that the "noise budget" is enough. Concretely, this means setting the ciphertext modulus Q accordingly. We elaborate below how much of the noise budget each operation consumes.

Noise Growth and Depth. We characterize the cost of homomorphic operations based on their impact on the noise budget. k of Homomorphic addition incurs a noise growth factor of approximately $\sqrt{k}$; in practice, this is negligible compared to multiplication, so we consider additions to be free.

In contrast, multiplication consumes a significant noise budget. We distinguish between two types:

[18] In fact, for BGV and BFV, the plaintext is divided into two $\vec{m}_1, \vec{m}_2 \in \mathbb{Z}_t^{N/2}$ vectors and the rotations are performed within these two vectors independently (i.e., let $\vec{m}'_b$ be the rotated vector, for $b \in [1, 2]$; then $\vec{m}'_b[i] = \vec{m}'_b[i + j]$ for $i, j \in [N/2]$). These two vectors can be swapped in their positions as well. For simplicity, we ignore this since it does not affect our scheme.

- **Scalar Multiplication:** Multiplying a ciphertext by a plaintext scalar increases noise by a factor of t (i.e., k consecutive scalar multiplications introduce t^k noise). In this work, we allocate this a cost of 0.5 levels of depth.

- **Homomorphic Multiplication:** Multiplying two ciphertexts increases noise by a factor of roughly $t \cdot N$ (i.e., k consecutive scalar multiplications introduce $(t \cdot N)^k$ noise). We allocate this a cost of 1 level of depth.

Lastly, for rotations, similar to additions, most works in FHE simply ignore it as its noise consumption is additive instead of multiplicative. Specifically, for k rotations, the noise is $k \cdot \sqrt{N} \cdot \ell$ for some tunable parameter ℓ (see, e.g., [49] for details). Concretely, in the default implementation of the SEAL library [68], the default parameters consume ≥ 10 bits of noise budget for 14 consecutive rotations, which is $< 1/3$ of the noise budget consumption as a single multiplication. Thus, we do not directly count it towards multiplicative depth, but we will count it towards the noise budget estimation in our evaluation.

The runtime of these operations is linear in the circuit depth due to the following reason: Operations are performed over a large ciphertext modulus Q. To implement this efficiently, the BGV implementations normally employ the Residue Number System (RNS) to decompose Q into d small primes $q_1, \ldots, q_d$, such that $\prod q_i \approx Q$. Each q_i is chosen to fit within a standard machine word ($\lessapprox 60$ bits). Since the required size of Q (and thus the number of RNS limbs d) scales linearly with the multiplicative depth of the circuit, the total runtime scales linearly with the depth. For large t (e.g., 50 bits), a single multiplication level consumes the noise budget provided by one limb. For smaller t (e.g., 17 bits), it can take 2 multiplications to consume the same noise budget. For simplicity, one may assume d to be equal to the number of multiplications needed unless otherwise specified.

4 GKR-inspired Blind Holographic IOPs

4.1 Overview and Prerequisites

In this section, we present two Blind Holographic IOPs (BhIOPs) based on variants of the GKR protocol. These protocols verify computations with SIMD-structure exhibited by payload circuits evaluated under BGV or BFV.

The Base vs. Folded paradigm. We distinguish between two strategies for the prover, which trade off homomorphic noise growth against communication bandwidth:

- **Base hIOP:** The prover minimizes noise consumption by avoiding homomorphic rotations and multiplications where possible. Instead of aggregating sumcheck polynomials across SIMD slots homomorphically, the prover sends distinct polynomials for every slot. The verifier aggregates these plaintext polynomials locally. This results in lower prover noise but higher communication ($N\times$).

- **Folded hIOP:** The prover prioritizes succinctness. It uses homomorphic rotations (via a FOLD operator) to aggregate sumcheck polynomials across slots into a single slot. This minimizes communication to standard GKR levels but consumes additional noise budget (approx. 1 level + $\log N$ rotations).

Recall from Section 2.4 that a BhIOP is fully specified by its two components:

1. The HE Scheme $\mathcal{E}$: We choose the BGV/BFV FHE scheme.

2. The hIOP π: We choose a suitable variant of the GKR protocol.

Therefore, it suffices to describe two hIOPs, which we call the *base* and *folded* hIOPs for reasons discussed later. We will first develop some prerequisite machinery, then describe our hIOPs for a simplified special case (for exposition sake), and finally present the general case. These hIOPs induce BhIOPs as per the transformation discussed in Section 2.4. We present a table of BGV operation counts for both of the BhIOP provers whose full derivations are detailed in Appendix C.

Remark 4.1. Total operation counts that are presented in this section and derived in Appendix C represent *conservative upper bounds*. For example, our derivation assumes that for every layer i, the sparsity of addition gates and multiplication gates can each *independently* approach the total layer size S_i (i.e., we bound $|\mathsf{add}_i| \leq S_i$ and $|\mathsf{mul}_i| \leq S_i$). Furthermore, our simplification using the global maximum width $s_{\max} := \max\{\log(S_i)\}$ introduces additional slack for circuits with varying layer widths. Thus, the actual computational cost for any specific circuit execution will be within (and likely lower than) the reported upper bounds.

4.1.1 Class of Supported Payload Circuits

We support payload circuits with three gate types: SIMD addition, SIMD multiplication, and rotation. For exposition sake, we make the assumption that every layer consists of either only SIMD arithmetic or only rotation gates.[19]

4.1.2 Data Layout and Notation

The payload circuit is represented in d layers, with L_0 denoting the output layer, and L_d denoting the input layer. Each layer has S_i gates. The supported gate types are Input, SIMD Addition, SIMD Multiplication, and Rotation. We assume that S_i is a power of two and $s_i := \log_2(S_i)$. We use $T_i = S_i \cdot N$ to denote the number of wires in the global $\mathbb{F}_q$-arithmetic circuit and $t_i := \log_2(T_i) = s_i + \log N$. As shorthand, $n := \sum_i S_i$ denotes the total number of gates in the payload circuit. When obvious from context, we drop subscripts on S_i, T_i, s_i, t_i.

After homomorphic payload evaluation, the server has n ciphertexts that arose as intermediate assignments when calculating the output. In a particular layer whose assignment has S ciphertexts, the assignment has T wire values. We index the wires with length t bitstrings according to the following convention:

- The most significant s bits describe the *ciphertext index*.
- The least significant $\log N$ bits describe the *slot index*
- The $t = s + \log N$ bits together describe the overall *index*.

This data layout is depicted below in Fig. 5.

We use the notation $\mathsf{mask}[w]$ to denote an encrypted value, which mathematically corresponds to a particular slot of a particular ciphertext. For typographical reasons, we henceforth use the convenient notation that

$$\mathsf{ct}[x] \equiv \left(\mathsf{mask}[x(0^{\log N})], \cdots, \mathsf{mask}[x(1^{\log N})]\right),$$

[19] In practice, there is a dedicated dummy 0 wire in every layer that helps with the transformation of a general circuit into one with this structure (see Section 3.1.5 for a discussion). When the circuit is not naturally layered, i.e. a value from layer i needs to be used as an input to layer $j < i - 1$, the transformation uses forward gates. Forward gates can be implemented either as addition with 0 or a zero rotation.

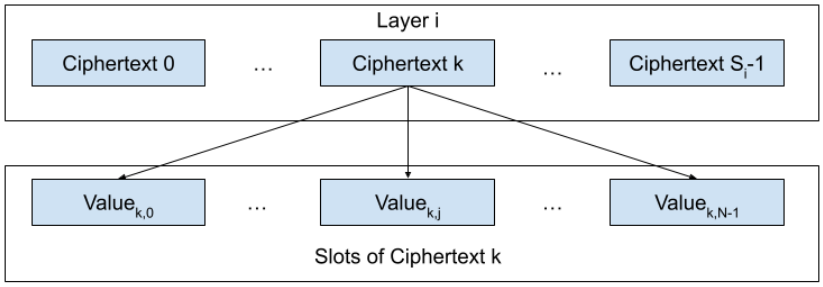

Figure 5: Data layout for the ith layer of a payload circuit. The first s_i bits determine which ciphertext ct holds the value, while the last $\log N$ bits determine the SIMD slot.

where $\mathsf{ct}[x]$ denotes a single ciphertext and $x : \{0,1\}^{\log N} \to \mathbb{F}_q$ is a function with inputs on the Boolean hypercube. For plaintexts, we adopt a similar convention:

$$\mathsf{pt}[y] \equiv \left(y(0^{\log N}), \cdots, y(1^{\log N})\right).$$

It will be convenient to define the notion of *extended* ciphertexts and plaintexts that store an N-tuple of $\mathbb{F}_{q^R}$ extension field elements. We use the respective notation ect and ept. Internally, a ect (or ept) is represented by R ct (or pt) limbs. Similarly, we use the notation emask to denote the encryption of an $\mathbb{F}_{q^R}$ extension field value. We will use the following equivalences

$$\begin{aligned}\mathsf{ect}[x] &\equiv \left(\mathsf{emask}[x(0^{\log N})], \ldots, \mathsf{emask}[x(1^{\log N})]\right),\\ \mathsf{ept}[y] &\equiv \left(y(0^{\log N}), \ldots, y(1^{\log N})\right),\end{aligned}$$

where $x, y : \{0,1\}^{\log N} \to \mathbb{F}_{q^R}$ are functions.

4.1.3 The folding operator

The FOLD operator acts on a list of N values and returns one value that stores the sum. For typographical convenience, we overload notation and sometimes invoke FOLD on an input ciphertext (or a plaintext) which has the same output type. The post-condition is that the returned ciphertext (or plaintext) stores the sum of the N input slots in its **first slot**. We allow FOLD to also take as input a ect (or ept), which runs R parallel copies of FOLD on the R ct (or pt) limbs of its input. Following the notation introduced in Section 4.1.2, it will sometimes be convenient to think of FOLD as taking as input a ct (or ect) and returning a mask (or emask).

Remark 4.2. The fold operator FOLD can be implemented in BFV/BGV using $\log N$ iterations of rotation and addition. As discussed in Section 3.2, rotations and additions are typically not counted toward the multiplicative depth; however, they do consume noise budget. In our setting, the total noise consumption is modest (less than 10 bits; see Section 6 for details). For generality, we introduce θ_{fold} to denote the multiplicative-depth–equivalent noise budget required to evaluate FOLD. Nevertheless, for the reasons above, we do not count θ_{fold} toward the multiplicative depth of Laminate.

4.2 Warmup: BhIOPs for Rotation-Free Payload Circuits

Recall from Section 4.1.1 that we assume some minimal structure on our payload circuits. First, we assume that they have d layers and that every wire in layer i is the result of some operation on wires in layer $i+1$. Second, we assume that every layer i consists solely of computation gates (SIMD arithmetic) or solely of rotation gates.

For exposition, we describe our two hIOPs (base and folded) in the simplified setting where all layers are *compute-only*. Equivalently, we describe the hIOPs for payload circuits without rotation gates.

Remark 4.3. For exposition, a reader may elect to proceed with the pedagogical assumption that $N = 1$ (and in particular $\log N = 0$). In this setting, the base and folded hIOPs are equivalent.

Notation Recap We let T_i denote the number of wires in the ith layer of a circuit Ckt, and let $S_i := \frac{T_i}{N}$ denote the number of ciphertexts required to represent the ith layer. As usual, we let $s_i := \log_2(S_i)$ and $t_i := \log_2(T_i) = s_i + \log N$. We let $n := \sum_{0 \le i \le d} S_i$ be the total number of gates.

4.2.1 Description of Rotation-Free GKR hIOPs

Our hIOPs will mirror the high level structure of GKR outlined in Fig. 3. For a d-layer payload circuit, we will invoke a layer reduction sub-protocol d times. Each invocation will reduce a set of evaluation claims about $\tilde{L}_i$ to a set of evaluation claims about $\tilde{L}_{i+1}$. Since our payload circuit is promised to have SIMD-structure, we can use an efficient bespoke layer reduction sub-protocol.[20]

Layer $i \to i+1$ reduction for Compute-Only Layer i. Fix a compute layer $i \in \{0, \ldots, d-1\}$. We are promised that the layer $(i+1) \to i$ computation includes only SIMD-respecting addition and multiplication operations. We therefore restrict the domain of indicator functions $\mathsf{add}_i, \mathsf{mul}_i$ to take labels that indicate *ciphertext indices* rather than wire indices (which would include a ciphertext index and a slot index). See Section 4.1.2.

In other words, $\mathsf{add}_i, \mathsf{mul}_i : \{0,1\}^{s_i} \times \{0,1\}^{s_{i+1}} \times \{0,1\}^{s_{i+1}} \to \{0,1\}$ denote indicator functions which specify whether a particular ciphertext in layer i is the result of a SIMD-addition (or SIMD-multiplication) of two particular ciphertexts in layer $i+1$.

Multilinear Evaluation for $\tilde{L}_i$. Exploiting this, evaluation of the ith layer MLE is given by the following equation:

$$\tilde{L}_i(\vec{\gamma}, \vec{\psi}) = \sum_{\vec{x},\vec{y} \in \{0,1\}^{s_{i+1}}} \sum_{\vec{\ell} \in \{0,1\}^{\log N}} \tilde{\mathsf{eq}}(\vec{\psi}, \vec{\ell}) \cdot \Bigg(\tilde{\mathsf{add}}_i(\vec{\gamma}_z, \vec{x}, \vec{y}) \cdot \left(\tilde{L}_{i+1}(\vec{x}, \vec{\ell}) + \tilde{L}_{i+1}(\vec{y}, \vec{\ell}) \right) + \tilde{\mathsf{mul}}_i(\vec{\gamma}_z, \vec{x}, \vec{y}) \cdot \left(\tilde{L}_{i+1}(\vec{x}, \vec{\ell}) \cdot \tilde{L}_{i+1}(\vec{y}, \vec{\ell}) \right) \Bigg) \tag{4}$$

Folded hIOP. In the folded hIOP, the prover and verifier engage in a standard sumcheck protocol for all $2s_{i+1} + \log N$ variables. This process fully reduces the claim about layer i to a claim about layer $i+1$ evaluated at specific challenges $\vec{r}_x, \vec{r}_y \in \mathbb{F}_{q^R}^{s_{i+1}}$ and $\vec{r}_l \in \mathbb{F}_{q^R}^{\log N}$. The complete protocol is specified in Figure 6.

Base hIOP. The base protocol modifies the standard approach in two ways to reduce prover noise at the cost of verifier work. First, the sumcheck is terminated early (after $2s_{i+1}$ rounds), leaving the sum over the SIMD slots ($\log N$ variables) to be handled directly by the verifier. Second, the prover

[20] These are an instance of "Data-Parallel" circuits discussed in Section 3.1.3.

Folded Compute Layer Reduction

1. **Full Sumcheck:** P and V engage in a sumcheck protocol for $2s_{i+1} + \log N$ rounds. At the end, V has challenges $\vec{r}_x, \vec{r}_y \in \mathbb{F}_{q^R}^{s_{i+1}}$ and $\vec{r}_l \in \mathbb{F}_{q^R}^{\log N}$, and a final claim $c \in \mathbb{F}_{q^R}$:

$$\begin{aligned} c \stackrel{?}{=} & \tilde{\mathsf{add}}_i(\vec{\gamma}_z, \vec{r}_x, \vec{r}_y) \cdot \left(\tilde{L}_{i+1}(\vec{r}_x, \vec{r}_l) + \tilde{L}_{i+1}(\vec{r}_y, \vec{r}_l)\right) \\ & + \tilde{\mathsf{mul}}_i(\vec{\gamma}_z, \vec{r}_x, \vec{r}_y) \cdot \left(\tilde{L}_{i+1}(\vec{r}_x, \vec{r}_l) \cdot \tilde{L}_{i+1}(\vec{r}_y, \vec{r}_l)\right) \end{aligned} \tag{5}$$

2. **Evaluation Claims:** P sends the claimed evaluations of the next layer:

$$v_x \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_x, \vec{r}_l), \quad v_y \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_y, \vec{r}_l)$$

3. **Consistency Check:** V checks if c is consistent with v_x, v_y using Eq. (5).

4. **Batching:** V samples a random challenge $\alpha \in \mathbb{F}_{q^R}$. The reduction is complete, and the new claim for layer $i+1$ is:

$$v_x + \alpha v_y \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_x, \vec{r}_l) + \alpha \tilde{L}_{i+1}(\vec{r}_y, \vec{r}_l) \tag{6}$$

Figure 6: Full specification of a Compute-Only Layer $i \to i+1$ reduction for the folded hIOP.

is lazy: instead of sending aggregated round polynomials, it sends separate round polynomials for each SIMD slot, which the verifier must aggregate. The detailed protocol is specified in Figure 7.

4.2.2 Operation Counts

Theorem 4.4. *Suppose plaintext field order q and cryptographic extension degree R are such that there exists a multiplicative depth 1 $\mathbb{F}_q$-circuit to compute $\mathbb{F}_{q^R}$ multiplication. Then the Base BhIOP prover for rotation-free circuits takes as input the intermediate ciphertexts arising from payload computation, requiring 1.5 levels of additional noise budget, and has homomorphic operation count upper bounded by Table 8.*

We remark that some challenges arise when designing a sumcheck prover circuit that achieves this depth lower bound. The naive low-depth sumcheck protocol for GKR would require at least 3 or 3.5 multiplicative levels. We defer a proof of this theorem to Appendix C.1. It is also unclear *a priori* how to handle extension field multiplications in an efficient manner. We discuss this in Remark 6.1.

Theorem 4.5. *Suppose plaintext field order q and cryptographic extension degree R are such that there exists a multiplicative depth 1 $\mathbb{F}_q$-circuit to compute $\mathbb{F}_{q^R}$ multiplication. Then the Folded BhIOP prover for rotation-free circuits takes as input the intermediate ciphertexts arising from payload computation, requiring $2.5 + \theta_{fold}$ levels of additional noise budget, and has homomorphic operation count upper bounded by Table 9.*

A proof of operation counts presented in Table 9 is given in Appendix C.2.

4.3 General Case: BhIOPs for Layered Payload Circuits

We now generalize the BhIOPs to support payload circuits that use rotation gates. Recall that we assume some structure on the payload circuit Section 4.1.1, i.e. that it is layered and each layer consists of only compute gates or only rotation gates. For such a payload circuit, we adopt the

Base Compute Layer Reduction

1. **Sumcheck (Early Termination):** P and V engage in sumcheck for $2s_{i+1}$ rounds. In every round j, P sends a vector of round polynomials $\mathsf{ept}[R_j]$ (i.e., N polynomials $\{R_{j,\vec{\ell}}(X)\}_{\vec{\ell}\in\{0,1\}^{\log N}}$). Fixing a specific slot index $\vec{\ell} \in \{0,1\}^{\log N}$, the round polynomial $R_{j,\vec{\ell}}(X)$ is equivalent to the jth round polynomial of the following sumcheck instance:

$$\tilde{L}_i(\vec{\gamma},\vec{\ell}) = \sum_{\vec{x},\vec{y}\in\{0,1\}^{s_{i+1}}} \tilde{\mathsf{add}}_i(\vec{\gamma}_z,\vec{x},\vec{y}) \cdot \left(\tilde{L}_{i+1}(\vec{x},\vec{\ell}) + \tilde{L}_{i+1}(\vec{y},\vec{\ell})\right) + \tilde{\mathsf{mul}}_i(\vec{\gamma}_z,\vec{x},\vec{y}) \cdot \left(\tilde{L}_{i+1}(\vec{x},\vec{\ell}) \cdot \tilde{L}_{i+1}(\vec{y},\vec{\ell})\right)$$

   V aggregates these to recover the standard round polynomial $R_j(X)$ via multilinear evaluation:

$$R_j(X) := \tilde{R}_j(X,\vec{\psi}) = \sum_{\vec{\ell}\in\{0,1\}^{\log N}} \tilde{\mathsf{eq}}(\vec{\psi},\vec{\ell}) \cdot R_{j,\vec{\ell}}(X)$$

2. **Final Sumcheck Claim:** After $2s_{i+1}$ rounds, with challenges $\vec{r}_x,\vec{r}_y \in \mathbb{F}_{q^R}^{s_{i+1}}$, V holds a claim $c \in \mathbb{F}_{q^R}$ that still sums over the slot indices $\vec{\ell}$:

$$c \stackrel{?}{=} \sum_{\vec{\ell}\in\{0,1\}^{\log N}} \tilde{\mathsf{eq}}(\vec{\psi},\vec{\ell}) \cdot \left(\tilde{\mathsf{add}}_i(\vec{\gamma}_z,\vec{r}_x,\vec{r}_y) \cdot \left(\tilde{L}_{i+1}(\vec{r}_x,\vec{\ell}) + \tilde{L}_{i+1}(\vec{r}_y,\vec{\ell})\right) + \tilde{\mathsf{mul}}_i(\vec{\gamma}_z,\vec{r}_x,\vec{r}_y) \cdot \left(\tilde{L}_{i+1}(\vec{r}_x,\vec{\ell}) \cdot \tilde{L}_{i+1}(\vec{r}_y,\vec{\ell})\right)\right) \tag{7}$$

3. **Vector Evaluation Claims:** P sends $2N$ evaluation claims. Using the plaintext vector notation from Section 4.1.2, these are sent as $\mathsf{ept}[w_x]$ and $\mathsf{ept}[w_y]$ such that for all $\vec{\ell} \in \{0,1\}^{\log N}$:

$$w_{x,\vec{\ell}} \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_x,\vec{\ell}), \quad w_{y,\vec{\ell}} \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_y,\vec{\ell})$$

4. **Consistency Check:** V uses the vectors $\mathsf{ept}[w_x], \mathsf{ept}[w_y]$ to evaluate the RHS of Eq. (7) and checks if it equals c.

5. **Sampling and Reduction:** V samples slot challenges $\vec{r}_l \sim \mathbb{F}_{q^R}^{\log N}$. V evaluates the multilinear extensions of the claim vectors to get single points:

$$v_x \leftarrow \tilde{w}_x(\vec{r}_l), \quad v_y \leftarrow \tilde{w}_y(\vec{r}_l)$$

6. **Batching:** V samples $\alpha \in \mathbb{F}_{q^R}$. The new claim for layer $i+1$ matches the form in Eq. (6):

$$v_x + \alpha v_y \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_x,\vec{r}_l) + \alpha \tilde{L}_{i+1}(\vec{r}_y,\vec{r}_l)$$

Figure 7: Full specification of a Compute-Only Layer $i \to i+1$ reduction for the base hIOP.

following notation. We let c be the number of compute-only layers (in these layers, there are no rotation gates) and f be the number of forward-only layers (in these layers, there are no SIMD arithmetic gates).[21] We let $d = c + f$ denote the total number of layers.

As before, we follow the high-level structure shown in Fig. 3, invoking a layer reduction subroutine d times. However, we use a different layer reduction subroutine depending on whether the current layer is compute-only or forward-only.

[21] We elect to call such a circuit a (c, f)-layered circuit.

Operation	Level	Total Count Bound
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$1.0 \to 0.0$	**3n**
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$1.5 \to 1.0$	$s_{\max} \cdot 8n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$0.5 \to 0.0$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	1.0	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	0.0	$s_{\max} \cdot 8n$
Total FHE Depth		**1.5 Levels**

Table 8: Total FHE Cost for the full (Warmup) Base BhIOP Prover on n-gate d-layer payload circuits without rotation gates.

Operation	Level	Total Count Bound
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$2.0 + \theta_{\text{fold}} \to 1.0 + \theta_{\text{fold}}$	**3n**
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$2.5 + \theta_{\text{fold}} \to 1.5 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$1.5 + \theta_{\text{fold}} \to 1 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$2.0 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$1.0 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n + 2d \log N$
$\mathsf{ect} \cdot \mathsf{ept} \to \mathsf{ect}$	$1.0 + \theta_{\text{fold}}$	$d(6s_{\max} + 4 \log N)$
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$2.0 + \theta_{\text{fold}}$	$2d \log N$
FOLD	θ_{fold}	$dR(6s_{\max} + 4 \log N)$
Total FHE Depth		**2.5 + θ_{fold} Levels**

Table 9: Total FHE Cost for the full (Warmup) Folded BhIOP Prover on n-gate d-layer payload circuits without rotation gates.

Notation Recap We let T_i denote the number of wires in the ith layer of Ckt, and let $S_i := \frac{T_i}{N}$ denote the number of ciphertexts required to represent the ith layer. As usual, we let $s_i := \log_2(S_i)$ and $t_i := \log_2(T_i) = s_i + \log N$. We let $n_c := \sum_{i \in \mathcal{C}} S_i$ be the total number of compute gates, $n_f = \sum_{i \in \mathcal{F}} S_i$ be the total number of forward gates, and define $n := S_d + n_c + n_f$ as the total number of gates.

4.3.1 (c, f)-layered GKR hIOP Description

Fix a (c, f)-layered payload circuit Ckt. The GKR protocol performs d layer reductions, reducing a claim about L_0 (the output layer) to L_d (the input layer). We partition the set $\{0, \ldots, d-1\}$ into sets $\mathcal{C}$ and $\mathcal{F}$ indicating whether a particular layer is compute-only or forward-only respectively.

In Section 4.2, we described how to handle *compute-only* layer reductions, so it suffices to describe a layer-reduction protocol for when $i \in \mathcal{F}$.

Layer $i \to i+1$ reduction for Forward-Only Layer i. Fix layer $i \in \mathcal{F}$. Let $\mathsf{fwd}_i : \{0,1\}^{t_i} \times \{0,1\}^{t_{i+1}} \to \{0,1\}$ be the indicator function defined as follows:

$$\mathsf{fwd}_i(\vec{z}, \vec{y}) = \begin{cases} 1 & \text{if } L_i[z] := L_{i+1}(y), \\ 0 & \text{otherwise.} \end{cases}$$

As per our usual notation, $\tilde{\mathsf{fwd}}_i[Z_0, \ldots, Z_{t_i - 1}, Y_0, \ldots, Y_{t_{i+1}-1}]$ denotes the multilinear extension of the function fwd_i. For $(\vec{\gamma}, \vec{\psi}) \in \mathbb{F}_{q^R}^{s_i + \log N}$, evaluation of the ith layer MLE can be written as the following sum with respect to layer $i+1$:

$$\tilde{L}_i(\vec{\gamma}, \vec{\psi}) = \sum_{\vec{x} \in \{0,1\}^{s_{i+1}}} \sum_{\vec{\ell} \in \{0,1\}^{\log N}} \tilde{\mathsf{fwd}}_i(\vec{\gamma}, \vec{\psi}, \vec{x}, \vec{\ell}) \cdot \tilde{L}_{i+1}(\vec{x}, \vec{\ell}) \tag{8}$$

Folded hIOP. In the folded hIOP, the prover and verifier engage in all $t_{i+1} = s_{i+1} + \log N$ sumcheck rounds. This process fully reduces the claim about layer i to a single evaluation claim about layer $i+1$. The complete protocol is specified in Figure 10.

Folded Forward Layer Reduction

1. **Full Sumcheck:** P and V engage in a sumcheck protocol for t_{i+1} rounds. At the end, V has challenges $\vec{r}_x \in \mathbb{F}_{q^R}^{s_{i+1}}$ and $\vec{r}_l \in \mathbb{F}_{q^R}^{\log N}$, and a final claim $c \in \mathbb{F}_{q^R}$:

$$c \stackrel{?}{=} \tilde{\mathsf{fwd}}_i(\vec{\gamma}, \vec{\psi}, \vec{r}_x, \vec{r}_l) \cdot \tilde{L}_{i+1}(\vec{r}_x, \vec{r}_l) \tag{9}$$

2. **Evaluation Claim:** P sends the claimed evaluation of the next layer:

$$v_x \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_x, \vec{r}_l)$$

3. **Consistency Check:** V checks if c is consistent with v_x using Eq. (9).
4. **Reduction:** The reduction is complete. The verifier now waits to be convinced of the claim $v_x \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_x, \vec{r}_l)$ in the next step.

Figure 10: Full specification of a Forward-Only Layer $i \to i+1$ reduction for the folded hIOP.

Base hIOP. In the base hIOP, the prover and verifier engage in only s_{i+1} rounds of sumcheck, leaving the sum over $\log N$ slot bits for the verifier to handle. Additionally, the prover is lazy and sends per-slot round polynomials. The detailed protocol is specified in Figure 11.

Remark 4.6. An interesting design choice could be to replace the forward-only layer with a "linear combination only" layer. With a drop-in replacement for the fwd indicator function, the GKR protocol could support proving that every wire in L_i is a particular linear combination of wires in L_{i+1}. We do not present it this way for two reasons: ease of exposition and ease of comparison to prior work. However, we strongly encourage practitioners who have a particular application (and payload circuit) in mind to consider this tunable lever.

In particular, this observation directly implies that for every circuit with multiplicative depth d_{mul}, there is an equivalent (c, f)-layered circuit with $c = d_{\mathsf{mul}}$ and $f = d_{\mathsf{mul}} + 1$. For payload circuits intended to be evaluated under FHE *without bootstrapping*, we expect d_{mul} to be bounded above by some reasonable constant, and not growing with circuit size n.

4.3.2 Operation Counts

This section introduces the operation counts of our schemes via the following theorems, with proofs of these counts provided in Appendix C.3 and Appendix C.4.

Theorem 4.7. *Suppose plaintext field order q and cryptographic extension degree R are such that there exists a multiplicative depth 1 $\mathbb{F}_q$-circuit to compute $\mathbb{F}_{q^R}$ multiplication. Then the Base BhIOP prover for (c, f)-payload circuits on n gates takes as input the intermediate ciphertexts arising from payload computation, requiring 1.5 levels of additional noise budget, and has homomorphic operation count upper bounded by Table 12.*

Theorem 4.8. *Suppose plaintext field order q and cryptographic extension degree R are such that there exists a multiplicative depth 1 $\mathbb{F}_q$-circuit to compute $\mathbb{F}_{q^R}$ multiplication. Then the Folded BhIOP prover for (c, f)-payload circuits on n gates takes as input the intermediate ciphertexts arising from payload computation, requiring $2.5 + \theta_{fold}$ levels of additional noise budget, and has homomorphic operation count upper bounded by Table 13.*

Base Forward Layer Reduction

1. **Sumcheck (Early Termination):** P and V engage in sumcheck for s_{i+1} rounds. In every round j, P sends a vector of round polynomials $\mathsf{ept}[R_j]$ (i.e., N polynomials $\{R_{j,\vec{\ell}}(X)\}_{\vec{\ell}\in\{0,1\}^{\log N}}$). Fixing a specific slot index $\vec{\ell} \in \{0,1\}^{\log N}$, the round polynomial $R_{j,\vec{\ell}}(X)$ is equivalent to the jth round polynomial of the following sumcheck instance:

$$\tilde{L}_i(\vec{\gamma}, \vec{\ell}) = \sum_{\vec{x}\in\{0,1\}^{s_{i+1}}} \tilde{\mathsf{fwd}}_i(\vec{\gamma}, \vec{\psi}, \vec{x}, \vec{\ell}) \cdot \tilde{L}_{i+1}(\vec{x}, \vec{\ell})$$

   To learn $R_j(X)$, V aggregates the slot polynomials (note that unlike the compute layer, there is no $\tilde{\mathsf{eq}}$ term here):

$$R_j(X) = \sum_{\vec{\ell}\in\{0,1\}^{\log N}} R_{j,\vec{\ell}}(X)$$

2. **Final Sumcheck Claim:** After s_{i+1} rounds, with challenges $\vec{r}_x \in \mathbb{F}_{q^R}^{s_{i+1}}$, V holds a claim $c \in \mathbb{F}_{q^R}$ that still sums over the slot indices $\vec{\ell}$:

$$c \stackrel{?}{=} \sum_{\vec{\ell}\in\{0,1\}^{\log N}} \tilde{\mathsf{fwd}}_i(\vec{\gamma}, \vec{\psi}, \vec{r}_x, \vec{\ell}) \cdot \tilde{L}_{i+1}(\vec{r}_x, \vec{\ell}) \tag{10}$$

3. **Vector Evaluation Claim:** P sends a vector of N scalar evaluation claims $\mathsf{ept}[w_x]$ such that for all $\vec{\ell} \in \{0,1\}^{\log N}$:

$$w_{x,\vec{\ell}} \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_x, \vec{\ell})$$

4. **Consistency Check:** V uses the vector $\mathsf{ept}[w_x]$ to evaluate the RHS of Eq. (10) and checks if it equals c.

5. **Sampling and Reduction:** V samples slot challenges $\vec{r}_l \sim \mathbb{F}_{q^R}^{\log N}$. V evaluates the multilinear extension of the claim vector to get a single point:

$$v_x \leftarrow \tilde{w}_x(\vec{r}_l)$$

   The verifier now waits to be convinced of the claim $v_x \stackrel{?}{=} \tilde{L}_{i+1}(\vec{r}_x, \vec{r}_l)$.

Figure 11: Full specification of a Forward-Only Layer $i \to i+1$ reduction for the base hIOP.

4.4 Summary: Base vs Folded BhIOP variants

We presented two BhIOPs: one *base* and one *folded* variant. The base variant has the following property. In each BhIOP round rd, the base prover sends a list of m_{rd} ciphertexts to the verifier. The base verifier decrypts these ciphertext and uses all $m_{\mathsf{rd}} \cdot N$ of the decrypted $\mathbb{F}_q$ values to make a verification decision. On the other hand, the folded verifier decrypts received ciphertexts and uses only m_{rd} of the $m_{\mathsf{rd}} \cdot N$ decrypted $\mathbb{F}_q$ values to make a verification decision.[22] As a visual aid, the difference between these protocols for common sumcheck rounds[23] is depicted below in Fig. 14.
A priori, the purpose of the folded protocol is unclear. To list a few drawbacks:

- For d-layer payload circuits, the folded BhIOP has $d \log N$ additional rounds. This happens when the base BhIOP prover terminates each layer reduction sumcheck $\log N$ rounds early.

[22] The folded BhIOP prover has done additional processing so that the folded BhIOP verifier need only care about the value stored in the first slot of each decrypted plaintext.

[23] Recall that the base prover skips some rounds of sumcheck entirely.

Operation	Level	Total Count Bound
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$1.0 \to 0.0$	$3n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$1.5 \to 1.0$	$s_{\max} \cdot 8n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$0.5 \to 0.0$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	1.0	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	0.0	$s_{\max} \cdot 10n$
$\mathsf{ept} \cdot \mathsf{ct} \to \mathsf{ect}$	1.0	$s_{\max} \cdot 2n_f$
Total FHE Depth		**1.5 Levels**

Table 12: Total FHE Cost for the full base BhIOP Prover on n-gate d-layer payload circuits with c compute-only and f forward-only layers. Here n_f denotes the number of gates on forward layers.

Operation	Level	Total Count Bound
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$2.0 + \theta_{\text{fold}}$	$3n + 2f \log N$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$2.5 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$1.5 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$2.0 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$1.0 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n + 2d \log N$
$\mathsf{ect} \cdot \mathsf{ept} \to \mathsf{ect}$	$1.0 + \theta_{\text{fold}}$	$d(6s_{\max} + 4 \log N)$
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$2.0 + \theta_{\text{fold}}$	$2d \log N$
$\mathsf{ept} \cdot \mathsf{ct}$	$1.0 + \theta_{\text{fold}}$	$2(ns_{\max} + 2f \log N)$
$\mathsf{ect} + \mathsf{ect}$	θ_{fold}	$2(ns_{\max} + f \log N)$
FOLD	θ_{fold}	$dR(6s_{\max} + 4 \log N)$
Total FHE Depth		**2.5 + θ_{fold} Levels**

Table 13: Total FHE Cost for the full folded BhIOP Prover on n-gate d-layer payload circuits with c compute-only and f forward-only layers.

- A direct consequence is that the folded BhIOP prover computes and sends more ciphertexts, and the folded BhIOP verifier needs (one slot of) information from all of the decrypted plaintexts.
- The folded BhIOP prover consumes more noise than its base counterpart, specifically one additional level of multiplication and $\log(N)$ rotations (which, as discussed in Section 3.2, is concretely < 10 bits of noise).

Despite the drawback of the folded prover doing more work, when BhIOPs are compiled to Blind SNARGs in a particular way, the resulting proof size and verifier time of the folded protocol are much less than their base counterparts. We discuss this in Section 5.

4.5 Active Slots and Batching

In the standard FHE setting, applications typically aim to fully exploit the SIMD parallel capabilities of the BFV/BGV schemes. However, specific application logic may naturally require a width of only $j < N$ slots to implement. We refer to such logic as a *j-slot payload circuit*.

To maximize throughput, it is common practice to evaluate $B := \lfloor N/j \rfloor$ disjoint instances of such a payload circuit simultaneously within a single set of ciphertexts. We define the number of *active slots* k as the total number of slots containing meaningful data across all batched instances:

$$k := B \cdot j \leq N$$

The remaining $N - k$ slots are considered padding (zeros).

The BhIOPs presented in this section assume that $k = N$ for simplicity and exposition. In general, all sums over slot indices can be optimized to have k terms rather than N. In particular, if $k \ll N$, we can optimize the round complexity and FHE operations performed in the folded BhIOP.

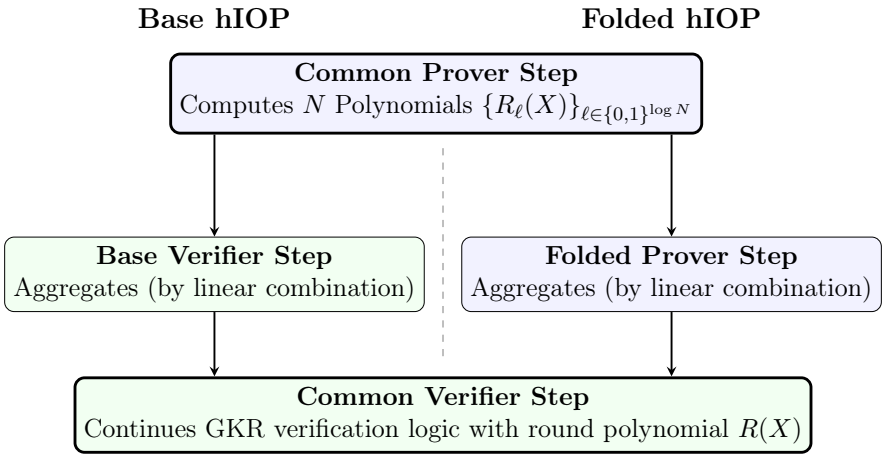

Figure 14: Comparison of protocol flow. Top: Both variants begin with the Prover computing raw polynomials. Middle: In **Base** (left), the Verifier aggregates; in **Folded** (right), the Prover aggregates. Bottom: Both converge on the same verification logic.

Prover Cost. A key property of our BhIOP provers is that they operate on full BGV ciphertexts. Moreover, if the payload circuit has k active slots, so too does the BhIOP prover circuit. This encourages maximizing B to process more instances at the same total cost.

5 Achieving Efficient VCoED

In Section 4, we defined two public-coin[24] BhIOP protocols (base and folded). In this section, we describe how to use these to build an efficient end-to-end VCoED protocol.

5.1 Sub-optimality of Previous Blind SNARG Compilation

The main ingredient needed for VCoED is a Blind holographic SNARG, which is obtained by describing a Blind holographic Interactive Proof (BhIP) and applying Fiat-Shamir to remove interaction. We could easily derive BhIPs from our BhIOPs using a trivial commitment scheme (see Remark 2.1), although this is especially wasteful for our folded BhIOP variant.

In BhIPs, FHE ciphertexts sent in each round contribute significantly to overall proof size. Consider instantiating the folded BhIOP with the trivial vector commitment scheme. In each round rd, the BhIOP verifier would receive m_{rd} ciphertexts. This allows the BhIP prover to communicate up to $m_{\mathsf{rd}} \cdot N$ many $\mathbb{F}_q$ elements to the BhIP verifier, but in our setting, only m_{rd} (those that reside in the first slot) have any influence on the final verification decision.

Let $m = \sum_{\mathsf{rd}} m_{\mathsf{rd}}$ denote the total number of ciphertexts the folded BhIOP prover sends across all rounds, each of which encodes useful information only in slot one. Ideally, the BhIOP prover could instead send $\lceil \frac{m}{N} \rceil$ "packed ciphertexts", however this would break temporal ordering required for IP soundness.

5.2 Transcript Packing: New BIOP to BIP Compiler

In this subsection, we introduce a **transcript packing** technique to compress the overall proof size. The key observation is that when compiling a blind IOP into an IP, it suffices to send *commitments*

[24]It should be assumed henceforth that all discussed protocols are public-coin.

to the blinded (i.e. FHE encrypted) messages in each round. The prover can then homomorphically derive a few "packed" ciphertexts at the end that encrypts the combined list of messages from all rounds of the IOP, along with a outer SNARK proof that this packed ciphertext is consistent with the commitments sent in each round.

Theorem 5.1. *Let $H : \{0,1\}^* \to \{0,1\}^{\mathrm{poly}(\lambda)}$ be a hash function modeled as a random oracle. Let $\mathcal{E}[\pi] = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ be an r-round public-coin BhIOP defined over BGV scheme $\mathcal{E}$. Let $m := \sum_{\mathsf{rd}=1}^{r} m_{\mathsf{rd}}$ denote the total number of raw ciphertexts sent in $\mathcal{E}[\pi]$ across all r rounds. Fix parameter $k \in [N]$ such that the verification decision of $\mathcal{E}[\pi].V$ is unaffected by changes to the contents of slots $\{k+1, \ldots, N\}$ in any of these m ciphertexts. Let $z := m \cdot k$ be the total number of useful $\mathbb{F}_q$ elements. Let $Z := \lceil z/N \rceil$ denote the number of N-tuples required to represent z elements. Let $F := \sum_{i=1}^{r} \lceil m_{\mathsf{rd}} \cdot k \rceil$.*

Then, the compiler described in Fig. 15 creates an r-round public-coin BhIP $\mathcal{E}'[\pi'] = (\mathsf{Setup}', \mathsf{P}', \mathsf{V}')$ with the following properties:

- *If $\mathcal{E}[\pi]$ is complete, then $\mathcal{E}'[\pi']$ is complete.*
- *If $\mathcal{E}[\pi]$ is sound when $\mathcal{E}[\pi].\mathsf{P}$ is given secret key sk, then $\mathcal{E}'[\pi']$ is sound in the ROM.*
- *Homomorphically computing all $\mathcal{E}'[\pi'].\mathsf{P}'$ messages requires one additional level compared to computing $\mathcal{E}[\pi].\mathsf{P}$ messages. Specifically, $\mathcal{E}'[\pi'].\mathsf{P}'$ performs fewer than $m+F$ rotations at level 1, m additions at level 1, F plaintext-ciphertext multiplications at level 1, and F ciphertext-ciphertext additions at level 0.*
- *The communication of $\mathcal{E}'[\pi'].\mathsf{P}'$ messages across all rounds consists of r hash digests, Z ciphertexts, and one SNARK proof.*

5.2.1 Compiler Description

To minimize the complexity of the outer SNARK, we employ a two-stage packing strategy. In round $i \in [r]$, the underlying IOP prover P_i produces m_i "raw" ciphertexts, where each contains valid data only in the first k slots. Committing to these m_i sparse ciphertexts directly would induce a large cost for the SNARK (which must verify the hash decomposition).

Instead, P'_i first performs *intra-round flattening*: it packs the m_i sparse ciphertexts into $f_i := \lceil (m_i \cdot k)/N \rceil$ dense ciphertexts using rotations and summations. Then, P'_i performs *inter-round alignment*: it homomorphically rotates these f_i flattened ciphertexts so that their contents align with non-overlapping positions in the final global packed vector. P'_i commits to these f_i aligned ciphertexts. This significantly reduces the number of inputs to the hash function from m_i to f_i. We formally describe the protocol $\mathcal{E}'[\pi']$ in Fig. 15.

5.2.2 Proof of Theorem 5.1

Completeness, efficiency, and communication complexity follow directly from the construction. We establish soundness via a reduction in the Random Oracle Model (ROM).

The Reduction. Let $\mathcal{P}^*$ be an adversary that breaks the compiled protocol soundness with non-negligible probability. We construct a reduction $\mathcal{B}$ that breaks the soundness of the underlying BhIOP. $\mathcal{B}$ simulates the Random Oracle H (maintaining a query list Q_H) and the public-coin verifier V' as follows:

Commit-and-Pack Compiler

Setup: Let N be the ciphertext slot count. Let m_i be the number of raw messages in round i, each of length k. The prover will pack these into $L = \lceil (\sum m_i \cdot k)/N \rceil$ final ciphertexts.

1. **For each round $i = 1$ to r:**
   - P'_{i} runs P_{i} to obtain m_i raw ciphertexts: $\mathsf{ct}_{\text{raw}}^{(i,1)}, \ldots, \mathsf{ct}_{\text{raw}}^{(i,m_i)}$.
   - **Intra-round Flattening:** P'_{i} packs these m_i sparse ciphertexts into $f_i = \lceil (m_i \cdot k)/N \rceil$ dense ciphertexts $\mathsf{ct}_{\text{flat}}^{(i,\cdot)}$. This step requires homomorphically rotating the raw ciphertexts to pack their useful slots $(0 \ldots k-1)$ into adjacent blocks within the flattened ciphertexts.
   - **Inter-round Alignment:** For each flattened ciphertext $u \in [f_i]$, P'_{i} computes a global offset $\rho_{i,u}$ to align it with the global transcript vector.

$$\mathsf{ct}_{\text{rot}}^{(i,u)} \leftarrow \mathsf{Rot}(\mathsf{ct}_{\text{flat}}^{(i,u)}, \rho_{i,u})$$

   - P'_{i} sends a hash commitment to the *aligned* set:

$$C_i := H\left(\{\mathsf{ct}_{\text{rot}}^{(i,u)}\}_{u=1}^{f_i}\right)$$

   - The verifier V' observes C_i and (if $i < r$) sends challenges c_i.
2. **Finalization (Round r):**
   - P' computes L final packed ciphertexts. Each $\mathsf{ct}_{\text{final},\ell}$ is a masked sum of the relevant subset S_ℓ of the rotated witness ciphertexts:

$$\mathsf{ct}_{\text{final},\ell} = \sum_{(i,u)\in S_\ell} \left(\mathsf{ct}_{\text{rot}}^{(i,u)} \cdot \mathsf{pt}_{\text{mask}}^{(i,u)}\right) \tag{11}$$

   - P' sends $\{\mathsf{ct}_{\text{final},\ell}\}_{\ell=1}^{L}$ and a SNARK proof π_{outer} attesting that the final ciphertexts are consistent with the commitments $\{C_i\}_{i=1}^{r}$ via Equation (11).
3. **Final Verifier Check:**
   - V' verifies π_{outer} against public inputs.
   - V' decrypts the L final ciphertexts to obtain the global transcript vector $\vec{v} \in \mathbb{F}_q^{L \cdot N}$.
   - **Input Reconstruction:** To simulate the inner verifier V_r, V' must reconstruct the original m_i ciphertexts for each round. For every round i and raw message index $j \in [m_i]$:
     (i) V' identifies the segment in $\vec{v}$ corresponding to the jth message of round i.
     (ii) V' extracts these values and rotates them back to slots $0, \ldots, k-1$.
     (iii) V' encodes these values into a trivial ciphertext $\mathsf{ct}_{\text{rec}}^{(i,j)}$.
   - V' executes the decision algorithm of V_r using these reconstructed ciphertexts.

Figure 15: Commit-and-Pack Compiler. We use two-stage packing to minimize the SNARK circuit size.

1. **Interaction:** For each round $i = 1 \ldots r$:
   - $\mathcal{B}$ receives commitment C_i from $\mathcal{P}^*$.
   - $\mathcal{B}$ samples a random challenge $c_i \leftarrow \mathbb{F}_q$ and sends it to $\mathcal{P}^*$.
2. **Finalization:** $\mathcal{P}^*$ outputs the final packed ciphertexts and a SNARK π_{outer}.
3. **Extraction:** $\mathcal{B}$ runs the SNARK extractor $\mathcal{E}$ on π_{outer} to obtain the witness $\mathbb{W} = \{\vec{w}_1, \ldots, \vec{w}_r\}$, where $\vec{w}_i$ represents the aligned ciphertexts for round i. If extraction fails, $\mathcal{B}$ aborts.
4. **Reconstruction:** $\mathcal{B}$ decrypts the final packed ciphertexts to get vector $\vec{v}$. $\mathcal{B}$ runs the deterministic reconstruction algorithm $\mathsf{Reconstruct}(\vec{v})$ to obtain the raw ciphertexts $\mathcal{T}_{\text{msg}} = (\{\tilde{\mathsf{ct}}_{\text{raw}}^{(1,\cdot)}\}, \ldots, \{\tilde{\mathsf{ct}}_{\text{raw}}^{(r,\cdot)}\})$.
5. **Transcript Output:** $\mathcal{B}$ outputs the transcript $\mathcal{T}$ formed by interleaving $\mathcal{T}_{\text{msg}}$ with the challenges $c_1, \ldots, c_r$.

Analysis. We assume $\mathcal{P}^*$ makes at most $Q = \text{poly}(\lambda)$ queries to H. If V' accepts the proof, then by the soundness of the SNARK, extraction succeeds with overwhelming probability. We now analyze the validity of the extracted transcript $\mathcal{T}$ against the underlying BhIOP.

We define two failure events in the Random Oracle Model:

- $\mathsf{E}_{\mathsf{guess}}$: $\mathcal{P}^*$ outputs a commitment C_i without having previously queried H on a preimage $\vec{u}_i$ such that $H(\vec{u}_i) = C_i$. The probability of this is $\mathsf{negl}(\lambda)$.
- $\mathsf{E}_{\mathsf{coll}}$: $\mathcal{P}^*$ finds two distinct inputs $x \neq y$ such that $H(x) = H(y)$. The probability of this is bounded by $Q^2/|\text{Image}(H)| \leq \mathsf{negl}(\lambda)$.

Assuming neither failure event occurs:

1. **Binding:** Since $\mathsf{E}_{\mathsf{guess}}$ did not occur, for every round i, there existed a query $\vec{u}_i$ in Q_H such that $H(\vec{u}_i) = C_i$ *before* $\mathcal{P}^*$ received c_i. Since $\mathsf{E}_{\mathsf{coll}}$ did not occur, the extracted witness $\vec{w}_i$ must be equal to this query $\vec{u}_i$ (otherwise $\vec{w}_i$ and $\vec{u}_i$ would constitute a collision). Thus, the witness $\vec{w}_i$ is effectively fixed before c_i is revealed.
2. **Independence:** The reconstructed raw ciphertexts $\mathcal{T}_{\text{msg}}$ are deterministically derived from the witnesses $\{\vec{w}_i\}$. Since $\vec{w}_i$ was fixed before round i (and thus before $c_i, \ldots, c_r$), the messages in $\mathcal{T}_{\text{msg}}$ are independent of the subsequent challenges.
3. **Validity:** If V' accepts, the SNARK ensures the consistency of the final ciphertexts with the commitments. Consequently, the reconstructed transcript $\mathcal{T}$ satisfies the decision predicate of the inner verifier V_r.

Since $\mathcal{T}$ is an accepting transcript where challenges are random and independent of prior messages, it constitutes a valid forgery against the BhIOP. The success probability of $\mathcal{P}^*$ is therefore bounded by:

$$\mathbf{Pr}[\text{Success}] \leq \epsilon_{\text{BhIOP}} + \epsilon_{\text{SNARK}} + \mathsf{negl}(\lambda)$$

5.2.3 Cost Analysis of Computing Outer SNARK π_{outer}

The R1CS Arithmetization. Let Q denote the final BGV ciphertext modulus. For simplicity, we assume that Q is a prime. Our R1CS will be defined natively over the scalar field $\mathbb{F}_Q$, so as to avoid expensive non-native field arithmetic.

- **Public inputs:**
  - The r commitment hashes $C_1, \ldots, C_r$.
  - The L final packed ciphertexts $\{\mathsf{ct}_{\text{final},\ell}\}_{\ell=1}^{L}$. Since the packing operation involves only plaintext-ciphertext multiplication and addition, these ciphertexts remain degree-1 (consisting of 2 polynomials: c_0, c_1). Thus, this input contributes $2NL$ field elements.
- **Witness:**
  - The $f = \sum f_i$ flattened ciphertexts $\{\mathsf{ct}_{\text{rot}}^{(i,u)}\}_{i,u}$. These are also degree-1 ciphertexts, contributing $2Nf$ field elements.

Constraint 1: Hashing (Commitment Verification). The circuit must verify that the witness elements correspond to the public commitments C_i. Let C_{hash} be the (amortized) number of R1CS constraints required to hash a single $\mathbb{F}_Q$ element. The total number of field elements to be hashed is $2Nf$. Thus, the hashing cost is:

$$\mathcal{N}_{\text{hash}} = 2Nf \cdot C_{\text{hash}} \tag{12}$$

Constraint 2: Linear Packing Consistency. The circuit must verify the relationship in Eq. (11):

$$\mathsf{ct}_{\text{final},\ell} = \sum_{(i,u)\in S_\ell} \left(\mathsf{ct}_{\text{rot}}^{(i,u)} \cdot \mathsf{pt}_{\text{mask}}^{(i,u)}\right)$$

In this equation, the masks $\mathsf{pt}_{\text{mask}}^{(i,u)}$ are public constants. We emphasize that this operation is a **plaintext-ciphertext** multiplication, so no relinearization is required; the result remains a pair of polynomials (c_0, c_1). Furthermore, we assume the modulus Q is sufficient to accommodate the noise growth from this operation.

In R1CS, this equation represents a fixed linear combination of witness variables. Enforcing that a linear combination of witnesses equals a public output requires **1 constraint per output element**. Since there are L output ciphertexts, each consisting of $2N$ elements, the cost is:

$$\mathcal{N}_{\text{linear}} = 2NL \tag{13}$$

Total Complexity. Summing these components, the total constraint count is:

$$\begin{aligned}
\mathcal{N}_{\text{total}} &= \mathcal{N}_{\text{hash}} + \mathcal{N}_{\text{linear}} && (14)\\
&= 2Nf \cdot C_{\text{hash}} + 2NL && (15)\\
&\leq 2Nf \cdot (C_{\text{hash}} + 1) && (16)
\end{aligned}$$

Prover Runtime Estimates. We are free to instantiate the outer SNARK with any field-agnostic scheme. For concreteness, we select:

- The Spartan [70] field-agnostic PIOP with the BaseFold [79] field-agnostic PCS.
- Wrapped in a Groth16 proof [44].

This combination yields $O(1)$ proof size and verification time. It remains to verify that the prover runtime is acceptable.

Cost of the outer Groth16 Proof. The Groth16 prover proves the Spartan verification circuit. As this is a succinct instance significantly smaller than the primary circuit, it is not the bottleneck; we therefore ignore this cost.

Figure 7 of [70] indicates that the single-threaded Spartan PIOP can prove 2^{20} R1CS constraints in approximately 36 seconds. However, our R1CS instance is highly uniform; as the author notes in [69], the Spartan prover is roughly 10× faster for uniform circuits. Consequently, we estimate the single-threaded Spartan PIOP can prove 2^{20} constraints in 3.6 seconds.

The Spartan prover must also commit to a witness vector z of length 2^{20}. According to Figure 5 of [79], committing to a 2^{20}-length vector of 64-bit field elements takes less than 0.4 seconds. Summing these components $(3.6+0.4)$ and assuming quasi-linear asymptotics for both Spartan and BaseFold, we extrapolate the time to compute π_{outer} for a circuit of size $\mathcal{N}$ as:

$$4 \cdot \frac{\mathcal{N}}{2^{20}} \cdot \frac{\log(\mathcal{N})}{20} \text{ seconds.}$$

Concrete Circuits. Assume a plaintext modulus of $q = 2^{16} + 1$ and extension degree $R = 8$ (providing ≈ 100 bits of security). We select a final ciphertext modulus $Q \leq 2^{32}$, which fits within 4 bytes. Using an arithmetization-friendly hash function like Poseidon [30] at security level $\lambda = 128$,

the cost amortizes to approximately two constraints per byte of input. Since elements of $\mathbb{F}_Q$ are 4 bytes, we set $C_{\text{hash}} \leq 8$.

Suppose our payload circuit has $n = 2^{20}$ gates, $d = 32$ layers, and SIMD-batching parameter $N = 2^{14}$. Following an argument similar to Claim 3.5, our folded prover requires at most:

$$\begin{aligned} r &= d(2\log(n/d) + \log N + 1) \\ &= 32(30 + 14 + 1) \\ &= 1440 \end{aligned}$$

rounds. Moreover, since each round contains $3R \ll N$ slots of useful data, we have $f = r$. Substituting these values into our bound for total constraints $\mathcal{N}_{\text{total}}$:

$$\mathcal{N}_{\text{total}} \leq 2Nf \cdot (C_{\text{hash}} + 1) \leq 2^{15} \cdot 1440 \cdot 9 \leq 2^{15} \cdot 2^{14} = 2^{29}.$$

Finally, we estimate the prover runtime:

$$\begin{aligned} 4 \cdot \frac{2^{29}}{2^{20}} \cdot \frac{29}{20} &= 4 \cdot 2^9 \cdot 1.45 \\ &\leq 3600 \text{ seconds.} \end{aligned}$$

This is a negligible fraction of the total VCoED server runtime (see Table 20). We conclude that the inclusion of an outer SNARK impacts neither proof size nor server runtime meaningfully.

5.3 End to End VCoED from Blind Holographic SNARGs

Let Ckt be a payload circuit with multiplicative depth θ_{Ckt}. Let $\mathcal{E}[\pi]$ be a Blind SNARG whose prover requires θ_π levels of noise budget to homomorphically compute the proof. We define the total required depth as $\theta := \theta_{\mathsf{Ckt}} + \theta_\pi$. In Fig. 16, we describe the end-to-end protocol where the client initializes the HE scheme with a modulus chain of length at least θ to accommodate both the payload evaluation and the subsequent proof generation.

5.4 Family of VCoED Protocols

As discussed in Section 5.3, an end-to-end VCoED protocol is fully determined by the choice of Blind Holographic SNARG, which is itself constructed from a public-coin BhIOP and a choice of compiler (trivial or transcript packing). We present three concrete VCoED instantiations derived from the BhIOPs in Section 4. We provide a high-level comparison of these three protocols against prior works in Table 1.

5.4.1 Efficiency Parameters and Notation

We analyze the server efficiency and proof size of the three Laminate protocols with respect to a (c, f)-layered payload circuit over the plaintext field $\mathbb{F}_q$ (as defined in Section 4.3) with $k \leq N$ active slots (as defined in Section 4.5).

Following earlier conventions, we let n_c denote the number of compute gates, n_f denote the number of rotation gates, $d = c + f$ be the number of layers, and $n = n_c + n_f + S_d$ be the total number of gates. Furthermore, we let $s_{\max}$ be the logarithm of the maximum layer width ($\max\{\log S_i\}$) and R be the extension degree of $\mathbb{F}_{q^R}$ required for soundness.

The operation counts and proof sizes presented below use the following parameters:

End-to-End VCoED from Blind Holographic SNARG

1. **Setup:** The client runs $\mathcal{E}[\pi].\mathsf{Setup}(1^\lambda, \mathsf{Ckt})$. This generates the HE keys $(\mathsf{sk}, \mathsf{evk})$ and the prover/verifier keys $(\mathsf{ipk}', \mathsf{ivk}')$. The client keeps ivk' (which contains sk) and sends ipk' (which contains evk) to the server (Prover).
2. **Client Encryption:** The client prepares input ciphertexts with θ levels of multiplicative depth noise budget.
3. **Server Computation:** The server homomorphically evaluates the payload circuit:
$$\{\mathsf{ct}[\mathsf{out}]\} \leftarrow \mathcal{E}.\mathsf{Eval}_{\mathsf{evk}}(\mathsf{Ckt}, \{\mathsf{ct}[\mathsf{in}]\})$$
4. **Server Witness Cleanup:** To minimize the cost of the subsequent proof generation, the server "cleans up" all intermediate ciphertexts (the witness) generated during payload evaluation. The server performs modulus switching on each intermediate ciphertext so that it has *exactly* the minimum θ_π levels of noise budget required for the proof. This reduces the bit-width of the witness ciphertexts before they enter the SNARG prover.
5. **Server Proof Generation:** Let $\mathcal{W}$ denote the set of cleaned-up intermediate ciphertexts. The server computes the Blind SNARG proof using the instance $\mathbf{x} = (\{\mathsf{ct}[\mathsf{in}]\}, \{\mathsf{ct}[\mathsf{out}]\})$ and witness $\mathcal{W}$:
$$\pi' \leftarrow \mathcal{E}[\pi].\mathsf{P}^{[\![\rho]\!]}(\mathsf{ipk}', \mathbf{x}, \mathcal{W})$$
6. **Server Response:** The server sends results $\{\mathsf{ct}[\mathsf{out}]\}$ and blind proof π' to the client.
7. **Client Verification:** The client uses its verifier key ivk' (which includes sk) to verify the proof.
$$\{\mathsf{acc}/\mathsf{rej}\} \leftarrow \mathcal{E}[\pi].V^{[\![\rho]\!]}(\mathsf{ivk}', (\{\mathsf{ct}[\mathsf{in}]\}, \{\mathsf{ct}[\mathsf{out}]\}), \pi')$$

Figure 16: End-to-End VCoED from Blind Holographic SNARG

- r**: The number of BhIOP rounds.**
  - Base BhIOP: $r \le d(2\log(n/d) + 1)$.[25]
  - Folded BhIOP: $r \le d(2\log(n/d) + \log k + 1)$.
- m**: The total number of "raw" ciphertexts in the BhIOP transcript.**
  - Base BhIOP: $m \le 6dR\log(n/d) + 2dR$.
  - Folded BhIOP: $m \le 6dR\log(n/d) + 2dR(\log k + 1)$.
- F**: The number of "flattened" ciphertexts (before inter-round packing).**
  - Base BhIOP: $F \le \sum_{\mathsf{rd}=1}^{r} \lceil m_{\mathsf{rd}} \cdot k/N \rceil$.[26]
  - Folded BhIOP: $F \le \sum_{\mathsf{rd}=1}^{r} \lceil m_{\mathsf{rd}}/N \rceil$.
- Z**: The final number of "packed" ciphertexts sent to the verifier (when using transcript packing technique).**
  - Base BhIOP: $Z \le \lceil m \cdot k/N \rceil$.
  - Folded BhIOP: $Z = \lceil m/N \rceil$.

[25] This is the bound from Claim 3.5 assuming $f = 0$. Compute layers require more rounds than forward layers.
[26] For every $\mathsf{rd} \in [r]$, we have $m_{\mathsf{rd}} \le 3R$.

Operation	Level	Total Count Bound
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$1.0 \to 0.0$	$3n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$1.5 \to 1.0$	$s_{\max} \cdot 8n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$0.5 \to 0.0$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	1.0	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	0.0	$s_{\max} \cdot 10n$
$\mathsf{ept} \cdot \mathsf{ct} \to \mathsf{ect}$	1.0	$s_{\max} \cdot 2n_f$
Total FHE Depth		**1.5 Levels**

Table 17: Total Additional FHE Cost for the full $\mathsf{Laminate}_{\mathsf{base}}$ Server after Payload Evaluation. Parameters are described in Section 5.4.1. We assume that extension field multiplication can be implemented as a multiplicative depth 1 arithmetic circuit over the base field, which is an assumption discussed in Remark 6.1.

5.4.2 $\mathsf{Laminate}_{\mathsf{base}}$: Base BhIOP with Trivial Compiler

This scheme instantiates the Base BhIOP (see Table 12) using the trivial commitment scheme. It requires 1.5 levels of multiplicative depth overhead. Since the trivial compiler performs no packing, the proof consists of m ciphertexts.

- **Prover Efficiency:** The server's cost is dominated by the BGV operations performed by the base BhIOP prover, which is independent of the active slot count k.
- **Use Case:** This scheme is optimal for high-throughput applications where the payload is fully batched ($k \approx N$) and bandwidth is not the bottleneck. In this regime, packing offers little compression benefit.

Proof Size: m ciphertexts.

Total server FHE operations are presented in Table 17.

5.4.3 $\mathsf{Laminate}_{\mathsf{fold}}$: Folded BhIOP with Transcript Packing

This scheme instantiates the Folded BhIOP (see Table 13) using the Transcript Packing compiler. It incurs the highest depth overhead, requiring approximately $3.5 + \theta_{\text{fold}}$ levels ($2.5 + \theta_{\text{fold}}$ from the BhIOP plus 1 from the compiler). Here, the active slots of the BhIOP transcript are packed into $Z = \lceil m/N \rceil$ ciphertexts (since the folded prover outputs data only in slot 1).

- **Prover Efficiency:** The server's cost is dominated by the BGV operations performed by the Folded and Packed BhIP Prover, which is independent of the active slot count k. The server has an additional cost of SNARK proving F plaintext-ciphertext multiplications and F ciphertext-ciphertext additions.
- **Use Case:** This protocol offers the smallest possible proof size ($Z \ll m$). It is ideal for bandwidth-constrained environments where the additional depth is acceptable.

Proof Size: $\lceil m/N \rceil$ ciphertexts + 1 SNARK Proof + r hash outputs.

Total server FHE operations are presented in Table 18.

Operation	Level	Total Count Bound
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$3.0 + \theta_{\text{fold}}$	$3n + 2f \log N$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$3.5 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$2.5 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$3.0 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$2.0 + \theta_{\text{fold}}$	$s_{\max} \cdot 8n + 2d \log N$
$\mathsf{ect} \cdot \mathsf{ept} \to \mathsf{ect}$	$2.0 + \theta_{\text{fold}}$	$d(6s_{\max} + 4 \log N)$
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$3.0 + \theta_{\text{fold}}$	$2d \log N$
$\mathsf{ept} \cdot \mathsf{ct}$	$2.0 + \theta_{\text{fold}}$	$2(ns_{\max} + 2f \log N)$
$\mathsf{ect} + \mathsf{ect}$	$1 + \theta_{\text{fold}}$	$2(ns_{\max} + f \log N)$
FOLD	$1 + \theta_{\text{fold}}$	$dR(6s_{\max} + 4 \log N)$
rotation	1	$m + F$
$\mathsf{ct} + \mathsf{ct}$	1	m
$\mathsf{pt} \cdot \mathsf{ct}$	1	F
$\mathsf{ct} + \mathsf{ct}$	0	F
Total FHE Depth		**3.5 + θ_{fold} Levels**

Table 18: Total Additional FHE Cost for the full $\mathsf{Laminate}_{\mathsf{fold}}$ Server after Payload Evaluation. Parameters are described in Section 5.4.1. We assume that extension field multiplication can be implemented as a multiplicative depth 1 arithmetic circuit over the base field, which is an assumption discussed in Remark 6.1.

5.4.4 $\mathsf{Laminate}_{\mathsf{pack}}$: Base BhIOP with Transcript Packing

This scheme instantiates the Base BhIOP using the Transcript Packing compiler, requiring 2.5 levels of multiplicative depth (1.5 from BhIOP plus 1 from compiler). Here, the active slots are packed into $Z = \lceil (m \cdot k)/N \rceil$ ciphertexts.

Remark 5.2 (Batching and Sparsity)**.** This variant is specifically optimized for the sparse setting where the active slot count is $k \ll N$ (e.g., a single instance of a narrow circuit). In such a case, the standard $\mathsf{Laminate}_{\mathsf{base}}$ prover would transmit ciphertexts that are mostly padding. Transcript packing allows the server to condense the execution trace into the few slots that matter, reducing communication.

However, if the application chooses to *batch* instances such that $k \to N$, this scheme converges to $\mathsf{Laminate}_{\mathsf{base}}$. As ciphertexts become dense with useful data, the packing compiler offers no compression benefit ($Z \to m$) while still incurring the computational overhead of the packing circuit.

Proof Size: $\lceil mk/N \rceil$ ciphertexts + 1 SNARK Proof + r hash outputs.

Total server FHE operations are presented below in Table 19.

Remark 5.3 (Verifiable CKKS)**.** Another important line of work in FHE with SIMD support is the CKKS homomorphic encryption scheme [23]. The main distinction from BGV/BFV is that CKKS supports fixed-point (i.e., approximate real-number) arithmetic rather than exact finite-field operations. Consequently, our current method does not immediately apply, since the GKR protocol is inherently defined over finite fields.

However, a very recent work [9] introduces a method for performing sumcheck over approximate computations. Because sumcheck is the core algebraic component underlying GKR, this result provides an intriguing first step toward extending GKR to the approximate setting. As the authors note, additional analysis is required to fully instantiate a GKR protocol over approximate arithmetic, but we view this development as a promising direction. An approximate GKR protocol may be combined with the techniques of this paper to attain *verifiable CKKS*. We leave this exploration to future work.

Operation	Level	Total Count Bound
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$2.0 \to 1.0$	$3n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$2.5 \to 2.0$	$s_{\max} \cdot 8n$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$1.5 \to 1.0$	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	2.0	$s_{\max} \cdot 8n$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	1.0	$s_{\max} \cdot 10n$
$\mathsf{ept} \cdot \mathsf{ct} \to \mathsf{ect}$	2.0	$s_{\max} \cdot 2n_f$
rotation	1	$m + Z$
$\mathsf{ct} + \mathsf{ct}$	1	m
$\mathsf{pt} \cdot \mathsf{ct}$	1	F
$\mathsf{ct} + \mathsf{ct}$	0	F
Total FHE Depth		**2.5 Levels**

Table 19: Total Additional FHE Cost for the full $\mathsf{Laminate}_{\mathsf{pack}}$ Server after Payload Evaluation. Parameters are described in Section 5.4.1. We assume that extension field multiplication can be implemented as a multiplicative depth 1 arithmetic circuit over the base field, which is an assumption discussed in Remark 6.1.

6 Evaluation

This section discusses implementation considerations and how we compare to prior works.

6.1 Implementation Considerations

Low-depth extension field multiplications. Recall that BGV/BFV uses $\mathbb{F}_p$ as its plaintext field, but for GKR evaluation, we need to work over $\mathbb{F}_{p^R}$ for some $R > 1$ (chosen so that $p^R \geq 2^\lambda$ for a security parameter λ). Fortunately, this extension is straightforward. Define $\mathbb{F}_{p^R} := \mathbb{F}_p[X]/f(X)$ where $f(X)$ is an irreducible polynomial of degree R. Then each element $x \in \mathbb{F}_{p^R}$ is represented as a vector in $\mathbb{F}_p^R$. To multiply $x, y \in \mathbb{F}_{p^R}$, we first compute $z \leftarrow x \otimes y \in \mathbb{F}_p^{2R-1}$ where $\otimes$ denotes polynomial multiplication (from two degree-$(R-1)$ polynomials to a degree-$(2R-2)$ polynomial).

Note that this polynomial must then be reduced modulo $f(X)$. More formally, for $i \geq R$ we use the relation $X^i \equiv X^{i-R} \cdot (-f(X))$ to reduce its degree. However, computing this naively requires $i - R + 1$ multiplicative levels: since $X^R \equiv -f(X)$, multiplying any coefficient $c_R \cdot X^R$ becomes $c_R \cdot (-f(X))$, which costs one multiplicative level. Then, reducing $c_{R+1} \cdot X^{R+1}$ first similarly requires multiplying $-X \cdot c_{R+1} \cdot f(X)$, again costing a multiplicative level. However, now, the resulting polynomial still contains a degree-R term that requires an additional reduction. Thus, modding X^i naively takes $i - R + 1$ multiplication levels, if the modulo operation is done in such a recursive way.

To avoid this depth blowup, we precompute $f_i(X) = X^i \bmod f(X)$ for all $i \geq R$ and store the results. Now, each reduction requires only a single multiplication, independent of i. Moreover, by choosing $f(X)$ so that all coefficients are ± 1, the reduction step involves only additions and subtractions, with no multiplications at all. Such a polynomial does not necessarily exist for arbitrary (p, R), but for the parameters we use (see Section 6), we identified such a choice. In this case, each extension-field multiplication costs exactly one multiplication level, matching the cost of a base-field multiplication.

Remark 6.1. For many prime powers p and required extension degrees R, there are sparse polynomials of degree R with low norm coefficients (e.g. $-1, 0, 1$) that are irreducible over $\mathbb{F}_p$.[27] Even when

[27] A "folklore" conjecture stipulates that we can find irreducible polynomials of any degree that have at most 5 nonzero coefficients, though it does not assert the coefficients are low norm. See Section 3.4.3 of [67].

	1 instance		128 instance		16384 instance		Noise budget overhead	Multiplicative Depth
	Total proving runtime (hours)	Total proof size (MB)	Total proving runtime (hours)	Total proof size (MB)	Total proving runtime (hours)	Total proof size (MB)		
$\mathsf{Laminate}_{\mathsf{base}}$	37.2	2604.00	37.2	2604	**37.2**	**2604**	50	1.5
$\mathsf{Laminate}_{\mathsf{pack}}$	**56.8**	**0.27**	**184.8**	**27.5**	16440.8	2604	80	2.5
$\mathsf{Laminate}_{\mathsf{fold}}$	75.7	0.27	**75.7**	**0.27**	**75.7**	**0.27**	120	3.5
Phalanx [81]	$\mathsf{W}(1)+2.3$	58.10	$\mathsf{W}(128)+294.4$	7437	$\mathsf{W}(16384)+37683.2$	951910	360	6
Blind Fractal [36]	$\mathsf{W}(1)+9.3$	116	$\mathsf{W}(128)+1190.4$	116.0	$\mathsf{W}(16384)+152371.2$	116.0	250	12

Table 20: Comparisons of concrete costs with prior works for the parameters in Section 6.2. $\mathsf{W}(k)$ stands for the runtime of witness reduction with respect to k disjoint instances, which is overlooked by prior works as discussed in Section 6. Noise budget overhead refers to the bits of the noise budget that must be added to the original FHE payload evaluation in order to compute the proof. It assumes witness reduction introduces no levels, though this may not be true for prior works. Laminate is separated into the three variants: a base scheme (Table 17), a packed scheme (Table 19), and a folded plus packed scheme (Table 18). The preferred version of our construction for various numbers of instances is highlighted in bold. Runtimes are all single-threaded.

such a polynomial cannot be found, $\mathbb{F}_{p^R}$ field multiplication can be computed by a $\mathbb{F}_p$-arithmetic circuit with multiplicative depth 1.5 (since we only require an addition level of scalar multiplication).

Delayed Inverse NTT and Relinearization. In BGV/BFV, a ciphertext multiplication between ct_1 and ct_2 (each containing two ring elements) consists of four steps: (1) applying the Number Theoretic Transform (NTT) to both ciphertexts, (2) performing tensor products to obtain a ciphertext with three ring elements, (3) applying the Inverse NTT, and (4) relinearizing to reduce back to two ring elements. When evaluating $\sum_i \mathsf{ct}_{1,i} \times \mathsf{ct}_{2,i}$, we can keep all the ciphertexts in NTT form in advance (such that step (1) is not necessary), and then we only need to perform step (2) for each pair to compute $\mathsf{ct}'_i \leftarrow \mathsf{ct}_{1,i} \times \mathsf{ct}_{2,i}$, and then accumulate $\mathsf{ct}' \leftarrow \sum_i \mathsf{ct}'_i$. Lastly, we apply steps (3) and (4) once to ct'. This reduces both runtime and noise growth.

6.2 Methodology

We estimate the server runtime in our scheme $\mathsf{Laminate}_{\mathsf{base}}, \mathsf{Laminate}_{\mathsf{pack}}, \mathsf{Laminate}_{\mathsf{fold}}$ (cf. Table 1) with the operation counts in Section 5.4, using the same methodology as the closest prior works, Phalanx [81] and Blind Fractal [36]. Specifically, we rely on operation counts and microbenchmarks, and we adopt similar parameter choices to ensure a fair comparison.

We use N disjoint instances of 1-slot payload circuits. (Section 4.5). Thus the resulting payload circuit uses all N SIMD slots (one per instance) and containing only SIMD arithmetic (not utilizing our rotation gates), with $n = 2^{20}$ gates and $d \leq 32$ layers. We assume each layer i contains the same number of gates $S_i = n/d$.[28] The resulting operation counts are shown in Section 5.4.

Microbenchmarks of BGV. We use the same VM instance type as [81] (Google Compute Cloud `N4` with 16GB RAM) and the same ring dimension $N = 2^{14}$. We perform microbenchmarks on the BGV scheme implemented in SEAL [68], as shown in Table 22. We choose BGV instead of BFV (as in [81]) because our evaluation involves substantially more ciphertext multiplications. However, these multiplications are aggregated, allowing us to use *delayed relinearization*. At a high level, relinearization is the most expensive component of ciphertext multiplication, but since it can be deferred until after aggregation (see Section 6.1), the amortized cost is significantly smaller. We therefore benchmark the primitive operations listed below. We use a ciphertext modulus of $\sim 2^{50}$ (such that HEXL acceleration can be best utilized [10]) and plaintext modulus $2^{16}+1$ with extension degree $R = 8$ (to achieve at least 100 bits of security in the GKR). For $x > 1$ RNS limbs, the runtime

[28] As shown in Lemma 3.6, assuming that each layer has the same number of wires upper bounds the resulting proof size for a circuit with n gates and d layers.

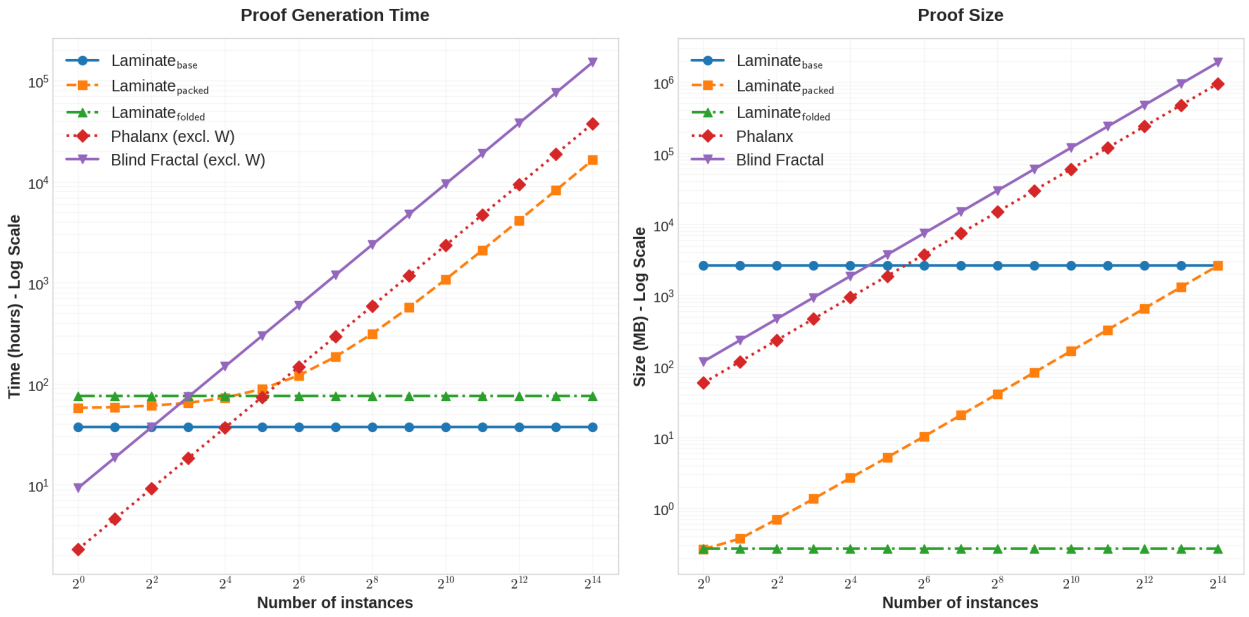

Figure 21: Proof generation time and proof size with respect to the number of instances for Laminate (all three variants as in Table 20) compared to prior works Phalanx [81] and Blind Fractal [36] *excluding* their runtime for witness reduction, i.e., W in Table 20. Note that these efficiency metrics overlook some of our additional advantages, such as our smaller noise budget overhead and its implications for the payload evaluation. We discuss this in more detail in Section 6.3.

Add	Scalar-by-ciphertext multiplication	Ciphertext-by-ciphertext multiplication (w/o relinearization)	Relinearization or rotation
0.022ms	0.027ms	0.064ms	1.331ms

Table 22: Microbenchmarks using SEAL library for our $N = 2^{14}$ and 1 RNS limb with ∼50 bit.

is multiplied by x. Table 20 is based on 2 RNS limbs for Laminate$_{base}$ and Laminate$_{pack}$ and three RNS limbs for Laminate$_{fold}$ (each with ≤ 50 bits, as discussed in Section 3.2).

More specifically, one level of scalar by ciphertext multiplication takes ∼17 bits of noise budget; one level of plaintext by ciphertext and ciphertext by ciphertext multiplication takes ∼30 bits of noise budget; and one FOLD operation, which takes $\log(N)$ rotations, takes < 10 bits of noise budget[29] (tested using the SEAL library [68]). By using this (and leaving 17 bits for the plaintext space), we obtain the amount of noise budget needed for each version of Laminate, and we transfer this to the number of RNS limbs by having each limb be at most 50 bits.

6.3 Comparisons

Performance. Using these microbenchmarks and delayed relinearization, we estimate our server runtime and proof size in Table 20 and Fig. 21. For Laminate$_{pack}$ and Laminate$_{fold}$, we additionally account for the SNARK runtime to compress the proof (see Section 5.2.3 for details).

For a varying number of instances, different variants of Laminate may be preferred. For a single instance, our proof size can be as small as 0.27MB, given 56.8 hours of proof generation for

[29]Note that this is the default noise consumption of rotations by SEAL [68]. However, rotation or key switching in general is highly flexible in terms of noise budget consumption and one may use a different way to realize key switching or rotation that takes a smaller or larger amount of noise budget in the underlying application. We use the default setting in [68] to demonstrate that rotation in most cases does not take much noise budget.

$\mathsf{Laminate}_{\mathsf{pack}}$. For a larger number of instances, $\mathsf{Laminate}_{\mathsf{fold}}$ takes over and becomes strictly better than $\mathsf{Laminate}_{\mathsf{pack}}$. However, $\mathsf{Laminate}_{\mathsf{base}}$ may be preferred if the server runtime is the main concern.

Compared to [81], $\mathsf{Laminate}_{\mathsf{pack}}$ is $\sim$25$\times$ slower for a *single* instance, but the proof size is 215$\times$ smaller. However, we would like to note that in the setting where the server *does not* contain any private input, Phalanx [81] proof contains 504 ciphertexts, which consist of roughly $504 \cdot N \approx 2^{23}$ field elements, which is larger than the number of gates in the delegated computation 2^{20}. Thus, in such a setting, the Phalanx client is noticeably slower than a client who does not use delegation at all, and computes the payload locally.

On the other hand, once the underlying BGV scheme leverages SIMD batching, our advantages become more significant. With 128 instances, our runtime becomes $> 3.8\times$ faster. With 16384 instances (full SIMD utilization), our runtime is $> 497\times$ faster. In general, as shown in Fig. 21, for ≥ 17 instances, $\mathsf{Laminate}_{\mathsf{base}}$ achieves better runtime than prior works and for ≥ 33 instances, $\mathsf{Laminate}_{\mathsf{fold}}$ achieves better runtime than prior works. $\mathsf{Laminate}_{\mathsf{fold}}$ constantly maintains $> 215\times$ better proof size, and becomes as high as 3807640$\times$ smaller (more than 6 orders of magnitude). These comparisons are shown in Fig. 21.

The difference in proof sizes is even larger when compared to [36]. Since [36] does not report proof sizes, and following [81], we omit that comparison. However, we expect their proof size to be of a similar order of magnitude as [81].

Limitations of prior evaluation. While the comparison above already illustrates the superior performance of our construction, we highlight several further advantages. (1) We can match the base field directly, avoiding additional overhead in the underlying application. (2) We can generate the proof directly from the intermediate ciphertexts produced during FHE evaluation, without any reformatting. (3) Our depth overhead is significantly smaller. The first advantage is difficult to quantify, and we elaborate on the latter two below.

Overlooked costs for proof generation (witness reduction). Prior works require a preprocessing step to convert intermediate FHE values into the witness format their proof systems expect.

In [81], if the FHE evaluation uses k SIMD slots per ciphertext, then matching the Phalanx input format requires $k \cdot n$ plaintext multiplications and $\log(k) \cdot k \cdot n$ rotations to produce $k \cdot n$ ciphertexts, each containing the same value replicated in all slots. With our parameters and $k = 128$ for $N = 16384$, the cost exceeds 10 hours. For $k = 1$, this preprocessing disappears, but such a configuration is unrealistic for BFV/BGV-based workloads.

Similarly, [36] requires $k \cdot n$ plaintext multiplications to compress the witness vector of length n into n/N ciphertexts, which again costs hours.

Once these overlooked costs are included, our performance advantage becomes even larger.

Additional gain from low-depth overhead. A further benefit of our construction is its near-minimal multiplicative depth overhead. Concretely, our variants require only 50, 80, or 120 bits of noise budget (and asymptotically a constant multiplicative depth of 1.5, 2.5, or 3.5 respectively), corresponding to 1, 2, or 3 RNS limbs; see Table 20. In contrast, prior works require either sacrificing a quasi-linear server circuit or introducing $\omega(1)$ additional levels asymptotically, and concretely for the parameters chosen in prior works, at least four additional levels concretely and ≥ 250 bits of noise budget, corresponding to ≥ 5 RNS limbs.

Note that multiplicative depth does not directly translate into noise budget, as it depends on the underlying parameter choices. For example, [36] adopts specialized parameters for GBGV rather than standard BGV, which reduce the noise consumption per level and therefore incur a smaller noise budget cost than [81]. In terms of concrete performance, the noise budget—implemented via RNS limbs, as discussed in Section 3.2—is the primary cost driver. Accordingly, we focus on noise

Circuit Type	8 Levels			20 Levels		
	Honest Eval (hrs)	VCoED Payload	VCoED Proving	Honest Eval (hrs)	VCoED Payload	VCoED Proving
Gates distributed uniformly over all multiplicative depths						
Half Adds/Mults	0.93	2.02×	81.4×	2.17	1.54×	34.9×
90% Mults	1.65	2.02×	45.9×	3.85	1.54×	19.7×
90% Adds	0.21	2.02×	360.5×	0.49	1.54×	154.5×
90% of gates at the highest multiplicative depth						
Half Adds/Mults	1.57	1.45×	48.2×	3.92	1.19×	19.3×
90% Mults	2.78	1.45×	27.2×	6.96	1.19×	10.9×
90% Adds	0.35	1.45×	216.3×	0.88	1.19×	86.0×

Table 23: Server evaluation times with $\mathsf{Laminate}_{\mathsf{fold}}$. The top section shows circuits with gates distributed uniformly over layers; the bottom section shows circuits where 90% of the gates are at the input layer. "Honest Eval" denotes the honest evaluation time of the payload without integrity. "VCoED Payload" denotes the VCoED evaluation of the payload relative to honest execution. "VCoED Proving" denotes the VCoED evaluation of the prover relative to honest payload execution.

budget in our comparisons below.

For a circuit requiring 20 RNS limbs (see Section 3.2), our proof requires only one to three extra limbs, while prior works require at least four. Thus, the underlying FHE computation incurs significantly greater overhead when using prior works. If each level has roughly the same number of homomorphic operations, then the payload evaluation overhead is $\sim 1.15, 1.20, 1.26\times$ for $\mathsf{Laminate}_{\mathsf{base}}, \mathsf{Laminate}_{\mathsf{pack}}, \mathsf{Laminate}_{\mathsf{fold}}$ respectively, while it is $\sim 1.36\times$ for blind fractal [36] and $\sim 1.41\times$ for Phalanx [81].

This depth overhead also increases ciphertext sizes and evaluation-key sizes. For example, suppose a circuit computation requires a 1200-bit ciphertext modulus. Under our scheme, the communication overhead from increasing the noise budget to $1250, 1280, 1320$ bits is $\sim 4.2\%, 6.7\%, 10\%$ respectively for our three variants. For prior works, the overhead is $\sim 20.9\%, 30\%$ respectively. For smaller-depth circuits, such as using 600 bits, the overhead doubles. For applications with large inputs, this difference is significant: for 10GB of ciphertext input or evaluation keys, prior works introduce 1–2GB of extra communication compared to our scheme for 1200-bit noise budget and doubles for 600-bit noise budget.

More generally, the communication overhead is approximately L/X, where L is the proof-system noise-budget overhead and X is the budget required by the original circuit. The computation overhead is roughly $\sum_{i\in[d]} C_i \cdot L/(X \cdot (d-i)/d)$, where C_i is the number of multiplicative operations at depth i in a circuit of depth d.

6.4 Overhead Compared to Honest Evaluation

Lastly, we discuss the overhead of our construction compared to payload evaluation via FHE *without* integrity, i.e., assuming a honest-but-curious server.

Parameters. We first discuss the parameters we compare against. Again, we use $n = 2^{20}$ gates (of either ciphertext additions or multiplications).

We start with the base case. Since $N = 2^{14}$ is used as ring dimension, the maximum noise budget is ~ 440 bits to guarantee 128-bit computational security [68]. Excluding the 120 bits needed to generate the proof and 17 bits of plaintext, the noise budget remaining is ~ 300, which is sufficient

for ≈ 8 multiplicative depth, so we set the number of RNS limbs to 8. We then assume that all these gates are uniformly distributed across each multiplicative depth, and half of them are additions and half of them are multiplications (for simplicity, assuming only ciphertext multiplications).

Then, we can extend the parameters to two additional settings: (1) gates are mostly distributed with the highest multiplicative depth, instead of uniformly distributed over all the depths[30]; and (2) most gates (e.g., 90%) of the gates are multiplications or are additions against half additions and half multiplications.

Lastly, we also explore the case where there are 20 multiplicative depth instead of 8. This is possible if we use $N = 2^{15}$ instead of 2^{14}, and since the change of ring dimension affects the payload computation and our proof generation with the same ratio, the overhead ratio remains the same. Thus, we still use $N = 2^{14}$ to perform the comparisons for simplicity. These parameters are summarized in Table 23.

Proof generation overhead. With these parameters, we show in Table 23 the payload generation runtime (using our microbenchmarks in Section 6.2) and the proof generation overhead of $\mathsf{Laminate}_{\mathsf{fold}}$ compared to the proof generation time. As shown, our proof generation creates 10×—360× over overhead compared to the payload generation. The scenario where our proof generation has the lowest overhead (∼10×) is when most gates are multiplications and when most gates are calculated with the highest depth. The scenario where our proof generation has the input overhead (∼360×) is when the most gates are additions and the gates are uniformly distributed. This is as expected since our proof generation runtime is essentially independent of these scenarios.

Payload generation overhead from additional multiplicative depth. As mentioned above, there is one additional overhead: since the proof requires additional multiplicative depth, the payload generation induces some additional overhead. Each operation has an overhead of $\frac{i+x}{i}$ at level i, for x being the number of extra RNS limbs for the VCoED proof generation. For $\mathsf{Laminate}_{\mathsf{fold}}$, this overhead is at most 102% (when all the gates are uniformly distributed), while prior works can be as high as 203% ([81]), even ignoring the fact that the additional noise budget requirement from prior work would further enlarge the ring dimension to guarantee sufficient computational security. Nonetheless, this overhead is relatively small compared to the proof generation overhead. However, recall that this still overlooks the potential overhead from prior works on witness reduction discussed before.

Acknowledgments

Zeyu Liu is supported by Yale CADMY.

[30] There is another scenario where most gates have the lowest multiplicative depth. The payload generation time of this case is close to when the multiplicative depth is only 1, so we omit this case.

References

[1] A. Alexandru, A. A. Badawi, D. Micciancio, and Y. Polyakov. Application-aware approximate homomorphic encryption: Configuring FHE for practical use. Cryptology ePrint Archive, Paper 2024/203, 2024.

[2] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: lightweight sublinear arguments without a trusted setup. *Des. Codes Cryptogr.*, 91(11):3379–3424, 2023.

[3] S. Angel, H. Chen, K. Laine, and S. T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018.

[4] D. F. Aranha, A. Costache, A. Guimarães, and E. Soria-Vazquez. HELIOPOLIS: verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. *Asiacrypt 2024*, 2024.

[5] S. Atapoor, K. Baghery, H. V. L. Pereira, and J. Spiessens. Verifiable FHE via lattice-based snarks. *IACR Commun. Cryptol.*, (1), 2024.

[6] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*. ACM, 1991.

[7] A. A. Badawi, A. Alexandru, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, C. Pascoe, Y. Polyakov, I. Quah, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[8] O. Bernard, M. Joye, N. P. Smart, and M. Walter. Drifting towards better error probabilities in fully homomorphic encryption schemes. In S. Fehr and P.-A. Fouque, editors, *EUROCRYPT 2025, Part VIII*, volume 15608 of *LNCS*, pages 181–211. Springer, Cham, May 2025.

[9] D. Bitan, Z. DeStefano, S. Goldwasser, Y. Ishai, Y. T. Kalai, and J. Thaler. Sum-check protocol for approximate computations. Cryptology ePrint Archive, Paper 2025/2152, 2025.

[10] F. Boemer, S. Kim, G. Seifu, F. D.M. de Souza, and V. Gopal. Intel hexl: Accelerating homomorphic encryption with intel avx512-ifma52. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '21, page 57–62, New York, NY, USA, 2021. Association for Computing Machinery.

[11] A. Bois, I. Cascudo, D. Fiore, and D. Kim. Flexible and efficient verifiable computation on encrypted data. In J. A. Garay, editor, *Public-Key Cryptography - PKC 2021*.

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

[13] J. Bootle, A. Chiesa, and J. Groth. Linear-time arguments with sublinear verification from tensor codes. In R. Pass and K. Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 19–46. Springer, 2020.

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

[15] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[16] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In R. Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, Oct. 2011.

[17] B. Bünz, B. Fisch, and A. Szepieniec. Transparent snarks from DARK compilers. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020.

[18] G. Cao, S. Chatel, and C. Knabenhans. HE-based on-the-fly MPC, revisited: Universal composability, approximate and imperfect computation, circuit privacy. Cryptology ePrint Archive, Paper 2025/1845, 2025.

[19] M. Checri, R. Sirdey, A. Boudguiga, and J.-P. Bultel. On the practical cpad security of "exact" and threshold fhe schemes and libraries. In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part III*, page 3–33, Berlin, Heidelberg, 2024. Springer-Verlag.

[20] H. Chen, Z. Huang, K. Laine, and P. Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *ACM CCS 2018*. ACM Press, Oct. 2018.

[21] H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. In *ACM CCS 2017*. ACM Press, Oct. / Nov. 2017.

[22] J. H. Cheon, H. Choe, A. Passelègue, D. Stehlé, and E. Suvanto. Attacks against the IND-CPA-d security of exact FHE schemes. CCS 2024.

[23] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[24] A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology - EUROCRYPT 2020*. Springer, 2020.

[25] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33. Springer, 2016.

[26] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *ACM CCS 2021*. ACM Press, Nov. 2021.

[27] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *Innovations in Theoretical Computer Science 2012*. ACM.

[28] G. Cormode, J. Thaler, and K. Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endow.*, (1), 2011.

[29] A. Dalvi, A. Jain, S. Moradiya, R. Nirmal, J. Sanghavi, and I. Siddavatam. Securing neural networks using homomorphic encryption. In *2021 International Conference on Intelligent Technologies (CONIT)*, pages 1–7, 2021.

[30] C. Dobraunig, L. Grassi, A. Guinet, and D. Kuijsters. Ciminion: Symmetric encryption based on Toffoli-gates over large finite fields. In A. Canteaut and F. Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021*, volume 12697 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2021.

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

[32] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://ia.cr/2012/144.

[33] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO '86*. Springer-Verlag, 1987.

[34] D. Fiore, A. Nitulescu, and D. Pointcheval. Boosting verifiable computation on encrypted data. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *Public-Key Cryptography - PKC 2020*.

[35] B. Fisch, A. Lazzaretti, Z. Liu, and C. Papamanthou. Thorpir: Single server pir via homomorphic thorp shuffles. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2024.

[36] M. Gama, E. H. Beni, J. Kang, J. Spiessens, and F. Vercauteren. Blind zkSNARKs for private proof delegation and verifiable computation over encrypted data. *IACR Cryptol. ePrint Arch.*, 2024.

[37] C. Ganesh, A. Nitulescu, and E. Soria-Vazquez. Rinocchio: Snarks for ring arithmetic. *J. Cryptol.*, (4), 2023.

[38] S. Garg, A. Goel, and M. Wang. How to prove statements obliviously? In *Advances in Cryptology - CRYPTO 2024*. Springer, 2024.

[39] R. Geelen and F. Vercauteren. Bootstrapping for BGV and BFV revisited. Cryptology ePrint Archive, Paper 2022/1363, 2022.

[40] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO 2010*. Springer, 2010.

[41] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*. ACM, 2009.

[42] S. Goldwasser, G. N. Rothblum, and Y. T. Kalai. Delegating computation: Interactive proofs for muggles. *Electron. Colloquium Comput. Complex.*, TR17-108, 2017.

[43] A. Golovnev, J. Lee, S. T. V. Setty, J. Thaler, and R. S. Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In H. Handschuh and A. Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226. Springer, 2023.

[44] J. Groth. On the size of pairing-based non-interactive arguments. In M. Fischlin and J. Coron, editors, *Advances in Cryptology - EUROCRYPT 2016*.

[45] A. Gruen. Some improvements for the PIOP for zerocheck. *IACR Cryptol. ePrint Arch.*, page 108, 2024.

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

[47] W. jie Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2021.

[48] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security 2018*. USENIX Association, Aug. 2018.

[49] A. Kim, Y. Polyakov, and V. Zucca. Revisiting homomorphic encryption schemes for finite fields. In *ASIACRYPT 2021*. Springer, 2021.

[50] J. Kim, J. Seo, and Y. Song. Simpler and faster BFV bootstrapping for arbitrary plaintext modulus from CKKS. Cryptology ePrint Archive, Paper 2024/109, 2024.

[51] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.

[52] J.-W. Lee, E. Lee, Y.-S. Kim, and J.-S. No. Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In *Advances in Cryptology – ASIACRYPT 2023*, pages 36–68. Springer Nature Singapore, 2023.

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

[54] B. Li and D. Micciancio. On the security of homomorphic encryption on approximate numbers. In A. Canteaut and F.-X. Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 648–677. Springer, Cham, Oct. 2021.

[55] B. Li, D. Micciancio, M. Raykova, and M. Schultz. Hintless single-server private information retrieval. In L. Reyzin and D. Stebila, editors, *CRYPTO 2024, Part IX*, volume 14928 of *LNCS*, pages 183–217. Springer, Cham, Aug. 2024.

[56] F. Liu, H. Liang, T. Zhang, Y. Hu, X. Xie, H. Tan, and Y. Yu. Hasteboots: Proving FHE bootstrapping in seconds. *IACR Cryptol. ePrint Arch.*, 2025.

[57] Z. Liu, K. Sotiraki, E. Tromer, and Y. Wang. Snake-eye resistant PKE from LWE for oblivious message retrieval and robust encryption. In S. Fehr and P.-A. Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 126–156. Springer, Cham, May 2025.

[58] Z. Liu and E. Tromer. Oblivious message retrieval. In *CRYPTO 2022, Part I*, Aug. 2022.

[59] Z. Liu, E. Tromer, and Y. Wang. Group oblivious message retrieval. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4367–4385, 2024.

[60] Z. Liu, E. Tromer, and Y. Wang. PerfOMR: Oblivious message retrieval with reduced communication and computation. Usenix Security'24, 2024.

[61] Z. Liu and Y. Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{\mathrm{O}}(1)$ polynomial multiplications. In J. Guo and R. Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 101–132. Springer, Singapore, Dec. 2023.

[62] Z. Liu and Y. Wang. Relaxed functional bootstrapping: A new perspective on BGV/BFV bootstrapping. In K.-M. Chung and Y. Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 208–240. Springer, Singapore, Dec. 2024.

[63] Z. Liu, Y. Wang, and B. Fisch. IND-CPA-d of relaxed functional bootstrapping: A new attack, a general fix, and a stronger model. ACM CCS 2025, 2025.

[64] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, (4), 1992.

[65] S. J. Menon and D. J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2022.

[66] D. Micciancio and J. Sorrell. Ring packing and amortized FHEW bootstrapping. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl, July 2018.

[67] G. L. Mullen and D. Panario. *Handbook of Finite Fields*. Chapman & Hall/CRC, 1st edition, 2013.

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

[69] S. Setty. Spartan, 2025. Accessed: 2025-12-05.

[70] S. T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Advances in Cryptology - CRYPTO 2020*. Springer, 2020.

[71] S. T. V. Setty, J. Thaler, and R. S. Wahby. Unlocking the lookup singularity with lasso. In M. Joye and G. Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 180–209. Springer, 2024.

[72] N. P. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *DCC*, 71(1):57–81, 2014.

[73] J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology - CRYPTO 2013*. Springer, 2013.

[74] J. Thaler. A note on the gkr protocol. Technical report, Georgetown University, 2015.

[75] L. T. Thibault and M. Walter. Towards verifiable FHE in practice: Proving correct execution of tfhe's bootstrapping using plonky2. *IACR Cryptol. ePrint Arch.*, 2024.

[76] A. Viand, C. Knabenhans, and A. Hithnawi. Verifiable fully homomorphic encryption. *WAHC'24*, 2023.

[77] Y. Wang and F. Zhang. Qelect: Lattice-based single secret leader election made practical. Usenix Security 2025, 2025.

[78] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764. Springer, 2019.

[79] H. Zeilberger, B. Chen, and B. Fisch. Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In *Advances in Cryptology - CRYPTO 2024*. Springer, 2024.

[80] J. Zhang, W. Wang, Y. Zhang, and Y. Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. *IACR Cryptol. ePrint Arch.*, 2020.

[81] X. Zhang, R. Wang, Z. Liu, B. Xiang, Y. Deng, B. Fisch, and X. Lu. Phalanx: An FHE-friendly SNARK for verifiable computation on encrypted data. CCS'25, 2025.

[82] M. Zhou, W.-K. Lin, Y. Tselekounis, and E. Shi. Optimal single-server private information retrieval. In C. Hazay and M. Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 395–425. Springer, Cham, Apr. 2023.

[83] Z. Zhou, Y. Li, Y. Wang, Z. Yang, B. Zhang, C. Hong, T. Wei, and W. Chen. ZHE: Efficient zero-knowledge proofs for HE evaluations. In *2025 IEEE SP*, 2025. Extended version at https://eprint.iacr.org/2025/770.

A Proof of Lemma 3.7

In this section, we give a full proof of the following lemma.

Lemma A.1. *Fix a layer reduction $i \in \{0, \ldots, d-1\}$. The prover can compute (sparse) tables $\hat{A}_j$ (and $\hat{M}_j$) at the beginning of the jth round of sumcheck such that for layer challenge $\vec{r} \in \mathbb{F}_{q^R}^{s_i}$, sumcheck challenges $\vec{\gamma} = (\gamma_0, \ldots, \gamma_{j-1}) \in \mathbb{F}_{q^R}^{j}$, $X \in \{0, 2, 3\}$, and $w \in \{0,1\}^{2s_{i+1}-j-1}$*

$$\hat{A}_j[X][w] = \tilde{\mathsf{add}}_i(\vec{r}, \vec{\gamma}, X, \vec{w})$$
$$\hat{M}_j[X][w] = \tilde{\mathsf{mul}}_i(\vec{r}, \vec{\gamma}, X, \vec{w})$$

Moreover, the prover requires $O(S_i \cdot (s_i + s_{i+1}))$ $\mathbb{F}_q$-ops in total across all sumcheck rounds.

We start with some machinery. Fix a layer $i \in \{0, \ldots, d-1\}$. Suppose for simplicity, that the statement we wish to prove is the following claim about layer i.

$$v_i = \tilde{L}_i(r) = \sum_{\vec{x} \in \{0,1\}^{s_{i+1}}} \sum_{y \in \{0,1\}^{s_{i+1}}} g^{(i)}(\vec{r}, \vec{x}, \vec{y})$$

for some evaluation point $r \in \mathbb{F}_{q^R}^{s_i}$ and evaluation claim $v_i \in \mathbb{F}_{q^R}$.

Remark A.2. Indeed, as we presented our protocol, we will actually be proving a claim about $\tilde{L}_i(r_a) + \alpha \tilde{L}_i(r_b)$, but we ignore this for ease of readability.

At each round $j \in \{0, \ldots, 2s_{i+1} - 1\}$ of the sumcheck, given challenges $\vec{\gamma} = (\gamma_0, \ldots, \gamma_{j-1})$, the prover needs to efficiently access evaluations of the gate functions. We define (sparse) tables A_j and M_j that store these values:

$$A_j[w] = \tilde{\mathsf{add}}_i(\vec{r}, \vec{\gamma}, \vec{w})$$
$$M_j[w] = \tilde{\mathsf{mul}}_i(\vec{r}, \vec{\gamma}, \vec{w})$$

for all $w \in \{0,1\}^{2s_{i+1}-j}$, where $\vec{r} \in \mathbb{F}_{q^R}^{s_i}$ is the layer i challenge. Because $\tilde{\mathsf{add}}_i$ is sparse, these tables will also be sparse, containing at most S_i non-zero entries. The prover computes them efficiently in two phases:

Initial Table Computation (A_0, M_0). The prover first computes the initial tables A_0 and M_0. We show this for A_0. From the definition of an MLE (c.f. Section 2.6):

$$\begin{aligned} A_0[x][y] &= \tilde{\mathsf{add}}_i(\vec{r}, \vec{x}, \vec{y}) \\ &= \sum_{z \in \{0,1\}^{s_i}} \mathsf{add}_i(z, x, y) \cdot \tilde{\mathsf{eq}}(\vec{z}, \vec{r}) \\ &= \sum_{(z,x,y) \in \mathsf{add}_i^{-1}(1)} \tilde{\mathsf{eq}}(\vec{z}, \vec{r}) \end{aligned}$$

The prover only needs to iterate over the (at most) S_i non-zero entries of add_i. Each $\tilde{\mathsf{eq}}(\vec{z}, \vec{r})$ evaluation (an inner product) takes $O(s_i)$ $\mathbb{F}_q$-ops. Therefore, computing the sparse tables A_0 and M_0 takes $O(s_i \cdot S_i)$ $\mathbb{F}_q$-ops.

Updating the Tables ($A_j \to A_{j+1}$). Given the tables A_j, M_j from round j and a new challenge γ_j, the prover computes A_{j+1}, M_{j+1}. This is a simple linear combination:

$$\begin{aligned} A_{j+1}[w] &= \tilde{\mathsf{add}}_i(\vec{r}, \gamma_0, \ldots, \gamma_j, \vec{w}) \\ &= (1-\gamma_j) \cdot \tilde{\mathsf{add}}_i(\vec{r}, \gamma_0, \ldots, \gamma_{j-1}, 0, \vec{w}) + \gamma_j \cdot \tilde{\mathsf{add}}_i(\vec{r}, \gamma_0, \ldots, \gamma_{j-1}, 1, \vec{w}) \\ &= (1-\gamma_j) \cdot A_j[0||w] + \gamma_j \cdot A_j[1||w] \end{aligned}$$

The prover only iterates over the w's for which $A_j[0||w]$ or $A_j[1||w]$ is non-zero. Since A_j has at most S_i non-zero entries, this update step takes only $O(S_i)$ $\mathbb{F}_q$-ops.

Finally, we are ready to prove the main lemma.

Proof. This follows from the two phases described above.

1. **Initial computation:** Computing A_0 and M_0 requires iterating over S_i non-zero entries and performing an $O(s_i)$ operation for each. The total cost is $O(S_i \cdot s_i)$.
2. **Per-round computation:** For each of the $j \in \{0, \ldots, 2s_{i+1} - 1\}$ rounds, the prover updates the tables (e.g., $A_j \to A_{j+1}$) and computes the necessary round polynomial evaluations (e.g., $\hat{A}_j[X][\vec{w}]$).
   - The update $A_j \to A_{j+1}$ requires $O(S_i)$ ops, as it's a linear combination over at most S_i non-zero entries.
   - Computing the round polynomial evaluations (e.g., for $X \in \{0, 2, 3\}$) also involves a linear extrapolation from $A_j[0][\vec{w}]$ and $A_j[1][\vec{w}]$, which again takes $O(S_i)$ ops for all w.

   The total cost across all $2s_{i+1}$ sumcheck rounds is $O(s_{i+1} \cdot S_i)$.

Combining both phases, the total work is $O(S_i \cdot s_i + S_i \cdot s_{i+1}) = O(S_i \cdot (s_i + s_{i+1}))$ $\mathbb{F}_q$-ops. □

B Gruen Section 3 Optimization

We first need to describe the optimization from Section 3 [45] whose most important consequence is to reduce the degree of the univariate round polynomials the prover must send.

Setting Suppose the multivariate polynomial $f(X_0, \ldots, X_{n-1})$ in our sumcheck instance takes the form $f(X_0, \ldots, X_{n-1}) = g(X_0, \ldots, X_{n-1}) \cdot \tilde{\mathsf{eq}}(X_0, \ldots, X_{n-1}, \gamma_0, \ldots, \gamma_{n-1})$ where g has maximum individual degree $d - 1$.

Regular Sumcheck Protocol In round $i \in \{0, \ldots, n-1\}$ of the regular sumcheck protocol, the current claim the prover needs to show is of the form:

$$s_i = \sum_{\vec{x} \in \{0,1\}^{n-i}} f(\vec{r}, \vec{x}),$$

where $\vec{r} = (r_0, \ldots, r_{i-1})$ are the i verifier challenges received thus far. To do so, the honest prover sends

$$\begin{aligned} R_i(X) &= \sum_{\vec{x} \in \{0,1\}^{n-i-1}} f(\vec{r}, X, \vec{x}) \\ &= \sum_{\vec{x} \in \{0,1\}^{n-i-1}} \tilde{\mathsf{eq}}(\vec{r}, X, \vec{x}, \vec{\gamma}) \cdot g(\vec{r}, X, \vec{x}) \\ &= \tilde{\mathsf{eq}}(\vec{r}, \gamma_0, \ldots, \gamma_{i-1}) \cdot \tilde{\mathsf{eq}}(X, \gamma_i) \cdot \sum_{\vec{x} \in \{0,1\}^{n-i-1}} \tilde{\mathsf{eq}}(\vec{x}, \gamma_{i+1}, \ldots, \gamma_{n-1}) \cdot g(\vec{r}, X, \vec{x}), \end{aligned}$$

Finally, the verifier checks whether $R_i(0) + R_i(1) \stackrel{?}{=} s_i$. If this check passes, the verifier samples a fresh challenge r_i, evaluates $s_{i+1} := R_i(r_i)$, and believes the original claim is true if the prover can convince her of the new, simpler, reduced claim:

$$s_{i+1} = \sum_{\vec{x} \in \{0,1\}^{n-i-1}} f(\vec{r}, r_i, \vec{x}),$$

Gruen Section 3 Refinement Gruen presents a refinement of the sumcheck protocol for these structured instances. Instead, in round i, the claim the prover needs to show is of the form:

$$s_i' = \sum_{\vec{x} \in \{0,1\}^{n-i}} \tilde{\mathsf{eq}}(\vec{x}, \gamma_i, \ldots, \gamma_{n-1}) \cdot g(\vec{r}, \vec{x})$$

To do so, the *refined* round i honest prover sends:

$$R_i'(X) = \sum_{\vec{x} \in \{0,1\}^{n-i-1}} \tilde{\mathsf{eq}}(\vec{x}, \gamma_{i+1}, \ldots, \gamma_{n-1}) \cdot g(\vec{r}, X, \vec{x})$$

With this modification, the verifier now checks

$$R_{i-1}'(r_{i-1}) \stackrel{?}{=} (1 - \gamma_i) R_i(0) + \gamma_i R_i(1).$$

She then samples a new challenge r_i, evaluates $s_{i+1}' := R_i'(r_i)$, and interprets the newly reduced claim as:

$$s_{i+1}' \stackrel{?}{=} \sum_{\vec{x} \in \{0,1\}^{n-i-1}} \tilde{\mathsf{eq}}(\vec{x}, \gamma_{i+1}, \ldots, \gamma_{n-1}) \cdot g(\vec{r}, r_i, \vec{x})$$

Reducing Round Polynomial Degree. In each round i, the regular sumcheck prover sends degree d univariate polynomial $R_i(X)$, whereas the refined sumcheck prover sends degree $d-1$ univariate polynomial $R'(X)$. Combined with the simple optimization observed in Remark 2.14, this means that the refined sumcheck prover need only compute the evaluation of $R'(X)$ at $d-1$ evaluation points. This will be useful in the aggregation phase of our bespoke layer reduction subroutine.

C BhIOP Operation Counts

C.1 Base BhIOP Prover FHE Operations for Rotation-Free Payload Circuits

In this subsection, we derive the operation counts presented in Table 8.

C.1.1 Overview and Notation

Quick Recap. The (blind) base protocol (for a d-layer rotation-free payload circuit) consists of d invocations of the *compute-only* layer reduction subroutine, where in the i-th invocation, the prover reduces a claim about wire values in Layer i to a claim about wire values in Layer $i+1$. The i-th layer reduction subroutine takes as advice the (encrypted) wire values for the $(i+1)$-th layer, denoted $\mathsf{ct}[L_{i+1}(\vec{b})]$ for $\vec{b} \in \{0,1\}^{s_{i+1}}$, where $S_{i+1} = 2^{s_{i+1}}$ denotes the number of ciphertexts in that layer. Layer reduction consists of two steps:

1. A $2s_{i+1}$-round invocation of the sumcheck protocol.
2. A final round where the prover sends two claimed (encrypted) evaluations of the multilinear extension $\tilde{L}_{i+1}$.

The final round requires computing $2N$ MLE evaluations: $\mathsf{ect}[\tilde{L}_{i+1}(\vec{\rho})], \mathsf{ect}[\tilde{L}_{i+1}(\vec{\phi})]$. Since $\vec{\rho}, \vec{\phi} \in \mathbb{F}_{q^R}^{s_{i+1}}$, each result encapsulates N extension field element, so their encryption can be represented with 2 ects. Concretely, $\mathsf{ect}[\tilde{L}_{i+1}(\vec{\rho})]$ is computed as the inner product $\sum_{\vec{b}\in\{0,1\}^{s_{i+1}}} \mathsf{ct}[L_{i+1}(\vec{b})] \cdot \tilde{\mathsf{eq}}(\vec{b}, \vec{\rho})$. This requires S_{i+1} operations of $\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$, which costs 0.5 levels. Thus, the final round is not the depth (nor operation count) bottleneck. The more interesting case to analyze is the sumcheck. The central question is:

How many FHE operations are needed to homomorphically compute sumcheck round polynomials?

Notation for Homomorphic Operations. We define our cost model based on BGV/BFV leveling, where Level 0 is the final level with insufficient noise budget to support another multiplication.

- **Ciphertext Types:**
  - ct: A ciphertext encrypting N base field elements $m \in \mathbb{F}_q^N$.
  - ect: A tuple of R ciphertexts, $(\mathsf{ct}_1, \ldots, \mathsf{ct}_R)$, encrypting N big field elements $m \in (\mathbb{F}_{q^R})^N$.
- **Operations and Costs:**
  - **Addition** (Cost: 0 levels):
    - $\mathsf{ct} + \mathsf{ct} \to \mathsf{ct}$. Output Lvl $= \min(\mathrm{Lvl}_A, \mathrm{Lvl}_B)$.
    - $\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$. Output Lvl $= \min(\mathrm{Lvl}_A, \mathrm{Lvl}_B)$.
  - **CT-Scalar Multiplication** (Cost: 0.5 levels):
    - $\mathsf{ct} \cdot \mathbb{F}_q \to \mathsf{ct}$. Output Lvl $=$ InputLvl $- 0.5$.
    - $\mathsf{ect} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$. Output Lvl $=$ InputLvl $- 0.5$.
    - *(Promotion)* $\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$. Output Lvl $=$ InputLvl $- 0.5$.
  - **CT-CT Multiplication** (Cost: 1.0 level):
    - $\mathsf{ct} \cdot \mathsf{ct} \to \mathsf{ct}$. Output Lvl $= \min(\mathrm{Lvl}_A, \mathrm{Lvl}_B) - 1.0$.
    - $\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$. Output Lvl $= \min(\mathrm{Lvl}_A, \mathrm{Lvl}_B) - 1.0$.

$*$ $\mathsf{ct} \cdot \mathsf{ect} \to \mathsf{ect}$. Output Lvl $= \min(\text{Lvl}_A, \text{Lvl}_B) - 1.0$.

All input wire ciphertexts $\mathsf{ct}[L_{i+1}(\vec{b})]$ are at **Level 1.5** (or lower when we can afford).

Remark C.1. Because neither the payload circuit nor the base BhIOP prover requires rotations, the presentation in this (and only this) subsection is best understood when $N = 1$. For typographical convenience, we intentionally conflate a ciphertext $\mathsf{ct}[x]$ and an encrypted $\mathbb{F}_q$ value $\mathsf{mask}[x]$.

C.1.2 Computing the Layer Reduction Sumcheck Round Polynomial under FHE

Fix a layer reduction instance $i \in \{0, \ldots, d-1\}$. We use the simplifying assumption that our sumcheck is for a single evaluation $\mathsf{ct}[\tilde{L}_i(\vec{r})]$.

$$\mathsf{ct}[v_i] \stackrel{?}{=} \mathsf{ct}[\tilde{L}_i(\vec{r})] = \sum_{\vec{x} \in \{0,1\}^{s_{i+1}}} \sum_{\vec{y} \in \{0,1\}^{s_{i+1}}} g^{(i)}(\vec{r}, \vec{x}, \vec{y})$$

$$\begin{aligned} \text{where } g^{(i)}(\vec{r}, \vec{x}, \vec{y}) &:= \tilde{\mathsf{add}}_i(\vec{r}, \vec{x}, \vec{y}) \left(\mathsf{ct}[L_{i+1}(\vec{x})] + \mathsf{ct}[L_{i+1}(\vec{y})]\right) \\ &\quad + \tilde{\mathsf{mul}}_i(\vec{r}, \vec{x}, \vec{y}) \left(\mathsf{ct}[L_{i+1}(\vec{x})] \cdot \mathsf{ct}[L_{i+1}(\vec{y})]\right) \end{aligned}$$

Before we dive in, we state a lemma that will be used extensively in the remainder of this section.

Lemma C.2. *Let $\tilde{L}[X_0, \ldots, X_{s-1}]$ be an s-variate MLE over $\mathbb{F}_q$ with 2^s encrypted Lagrange coefficients $\mathsf{ct}[L(\vec{b})]$ at **Level l + 0.5**. Let $j < s$ and $\vec{\gamma} = (\gamma_0, \ldots, \gamma_j) \in \mathbb{F}_{q^R}$ be some partial evaluation point. Let $\vec{x} \in \{0,1\}^{s-j-1}$ be some Boolean hypercube point. Let $c \in \mathbb{F}_{q^R}$ be some constant.*

*We can compute encrypted evaluations $\mathsf{ect}[\tilde{L}(\vec{\gamma}, X, \vec{x})]$ for all $X \in \{0, 2, 3\}$ using $5 \cdot 2^j$ Promotions ($\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$) ops (Lvl l + 0.5 $\to$ **Lvl l**) and $5 \cdot 2^j$ CT-CT Add ($\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$) ops (at Lvl l). The resulting ciphertext ect is at **Level l**.*

Proof. For each $X \in \{0, 2, 3\}$, the following equation holds.

$$\begin{aligned} & c \cdot \mathsf{ect}[L(\vec{\gamma}, X, \vec{x})] \\ &= (1 - X)c \cdot \mathsf{ect}[L(\vec{\gamma}, 0, \vec{x})] + Xc \cdot \mathsf{ect}[L(\vec{\gamma}, 1, \vec{x})] \\ &= (1 - X)c \sum_{b \in \{0,1\}^j} \tilde{\mathsf{eq}}(\vec{b}, \vec{\gamma}) \cdot \mathsf{ct}[L(\vec{b}, 0, \vec{x})] \\ &\quad + Xc \sum_{b \in \{0,1\}^j} \tilde{\mathsf{eq}}(\vec{b}, \vec{\gamma}) \cdot \mathsf{ct}[L(\vec{b}, 1, \vec{x})] \\ &= \sum_{b \in \{0,1\}^j} \left((1 - X)c \cdot \tilde{\mathsf{eq}}(\vec{b}, \vec{\gamma})\right) \cdot \mathsf{ct}[L(\vec{b}, 0, \vec{x})] \\ &\quad + \sum_{b \in \{0,1\}^j} \left(Xc \cdot \tilde{\mathsf{eq}}(\vec{b}, \vec{\gamma})\right) \cdot \mathsf{ct}[L(\vec{b}, 1, \vec{x})] \end{aligned}$$

Each of these final sums have 2^j terms. When $X = 0$, the second sum is 0, so we can omit it. □

Now, fix a sumcheck round $j \in \{0, \ldots, 2s_{i+1} - 1\}$ with challenges $\vec{\gamma} = (\gamma_0, \ldots, \gamma_{j-1}) \in \mathbb{F}_{q^R}^j$. For all $X \in \{0, 2, 3\}$ and $w \in \{0,1\}^{2s_{i+1}-j-1}$, we have access to the plaintext sparse tables $\hat{A}_j[X][w]$ and $\hat{M}_j[X][w]$ from Lemma 3.7.

Remark C.3 (Indicator Table sparsity)**.** For each $X \in \{0, 2, 3\}$, these tables (combined) have at most $\min\{S_i, 2^{2s_{i+1}-j-1}\}$ nonzero evaluations.

Operation	Count	Input Level	Output Level
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$ (Promotion)	$3S_i$	0.5	0.0
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$ (Add)	$3S_i$	0.0, 0.0	0.0

Table 24: FHE Cost for $R_j^{(1)}(X)$ (for all $X \in \{0, 2, 3\}$)

Case 1: $j < s_{i+1}$ (First Half of Sumcheck)

The round polynomial is $R_j(X) = \sum_{\vec{x}} \sum_{\vec{y}} g^{(i)}(\vec{r}, \vec{\gamma}, X, \vec{x}, \vec{y})$.

$$R_j(X) = \sum_{\vec{x}} \sum_{\vec{y}} \tilde{\mathsf{add}}_i(\cdots) \cdot \left(\mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}, X, \vec{x})] + \mathsf{ct}[L_{i+1}(\vec{y})]\right) + \tilde{\mathsf{mul}}_i(\cdots) \cdot \left(\mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}, X, \vec{x})] \cdot \mathsf{ct}[L_{i+1}(\vec{y})]\right)$$

We decompose $R_j(X) = R_j^{(1)}(X) + R_j^{(2)}(X) + R_j^{(3)}(X)$, where:

$$R_j^{(1)}(X) = \sum_{\vec{y}} \mathsf{ct}[L_{i+1}(\vec{y})] \cdot \left(\sum_{\vec{x}} \tilde{\mathsf{add}}_i(\cdots)\right)$$

$$R_j^{(2)}(X) = \sum_{\vec{x}} \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}, X, \vec{x})] \cdot \left(\sum_{\vec{y}} \tilde{\mathsf{add}}_i(\cdots)\right)$$

$$R_j^{(3)}(X) = \sum_{\vec{x}} \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}, X, \vec{x})] \cdot \left(\sum_{\vec{y}} \tilde{\mathsf{mul}}_i(\cdots) \cdot \mathsf{ct}[L_{i+1}(\vec{y})]\right)$$

Calculating $R_j^{(1)}(X)$. The inner sum $c_y = \sum_{\vec{x}} \tilde{\mathsf{add}}_i(\ldots)$ is a plaintext $\mathbb{F}_{q^R}$ scalar. The outer sum is $\sum_{\vec{y}} \mathsf{ct}[L_{i+1}(\vec{y})] \cdot c_y$. This is a sparse sum of at most S_i terms (due to $\hat{A}_j$).

- S_i ops of $\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$ (Lvl 0.5 $\to$ Lvl 0.0).
- S_i ops of $\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$ (at Lvl 0.0).

Total cost for $R_j^{(1)}$ (all $X \in \{0, 2, 3\}$) is $3S_i$ Promotions and $3S_i$ Additions, reflected in Table 24.

Calculating $R_j^{(2)}(X)$. We can start with $\mathsf{ct}[L_{i+1}[x]]$ at level 0.5. This is

$$R_j^{(2)}(X) = \sum_{\vec{x} \in \{0,1\}^{s_{i+1}-j-1}} \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}, X, \vec{x})] \cdot c_x$$

.

- Computing all the c_x's is a cost we ignore (because all can be computed in $3S_i$ operations outside of FHE due to Remark C.3). This is seen by the formula

$$c_x = \sum_{\vec{y} \in \{0,1\}^{s_{i+1}} : (x,y) \in \mathsf{spt}(\hat{A}_j[X])} \hat{A}_j[X][\vec{x}, \vec{y}]$$

- Computing each term $T_x := c_x \cdot \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}, X, \vec{x})]$ costs $5 \cdot 2^j$ promotions (at level 0.5) and $5 \cdot 2^j$ $\mathsf{ect} + \mathsf{ect}$ additions (at level 0) as per Lemma C.2.

Operation	Count	Input Level	Output Level
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$ (Promotion)	$5S_{i+1}/2$	0.5	0.0
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$ (Add)	$5S_{i+1}/2 + S_{i+1}/(2^{j+1})$	0.0, 0.0	0.0

Table 25: FHE Cost for $R_j^{(2)}(X)$ (for all $X \in \{0, 2, 3\}$)

Operation	Count	Input Level(s)	Output Level
Step 1: Compute c_x terms			
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$3S_i$	1.5	1.0
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$3S_i$	1.0, 1.0	1.0
Step 2: Compute d_x terms			
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$5S_{i+1}/2$	1.5	1.0
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$5S_{i+1}/2$	1.0, 1.0	1.0
Step 3: Compute T_x products (e.g., $T_x = c_x \cdot d_x$)			
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	$\mathbf{3S_{i+1}/(2^{j+1})}$	**1.0, 1.0**	**0.0**
Step 4: Sum T_x terms			
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$3S_{i+1}/(2^{j+1})$	0.0, 0.0	0.0

Table 26: FHE Cost for $R_j^{(3)}(X)$ (for all $X \in \{0, 2, 3\}$)

There are $2^{s_{i+1}-j-1}$ terms T_x that need to be summed. In total this computation costs $5S_{i+1}/2$ promotions (at level 0.5) and $5S_{i+1}/2 + S_{i+1}/(2^{j+1})$ $\mathsf{ect} + \mathsf{ect}$ additions (at level 0). This is reflected in Table 25.

Calculating $R_j^{(3)}(X)$. This is $R_j^{(3)}(X) = \sum_{\vec{x}} d_x \cdot c_x$.

- $d_x = \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}, X, \vec{x})]$ is a ect at **Lvl 1.0**.
- $c_x = \sum_{\vec{y}} \tilde{\mathsf{mul}}_i(\ldots) \cdot \mathsf{ct}[L_{i+1}(\vec{y})]$ is a ect at **Lvl 1.0** (it's a sparse inner product of $\mathsf{ct} \cdot \mathbb{F}_{q^R}$).

Let's analyze c_x first. It's a sparse sum of $\leq S_i$ terms in total (across all x).

1. **Compute $S_{i+1}/(2^{j+1})$ terms c_x:** (Done 3 times for $X = 0, 2, 3$).
   - $3S_i$ ops of $\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$ (Lvl 1.5 $\to$ Lvl 1.0) by Remark C.3.
   - $\approx 3S_i$ ops of $\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$ (at Lvl 1.0).
2. **Compute $S_{i+1}/(2^{j+1})$ terms d_x:** (Done 3 times for $X = 0, 2, 3$).
   - For each term: $5 \cdot 2^j$ ops of $\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$ (Lvl 1.5 $\to$ Lvl 1.0) by Lemma C.2.
   - For each term: $5 \cdot 2^j$ ops of $\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$ (at Lvl 1.0).
3. **Compute $S_{i+1}/(2^{j+1})$ products ($T_x = c_x \cdot d_x$):**
   - $3S_{i+1}/(2^{j+1})$ ops of $\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$ (Lvl 1.0, Lvl 1.0 $\to$ **Lvl 0.0**).
4. **Sum $S_{i+1}/(2^{j+1})$ terms (T_x):**
   - $3S_{i+1}/(2^{j+1})$ ops of $\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$ (at Lvl 0.0).

This is the depth bottleneck. The operation $\mathsf{ect}[1.0] \cdot \mathsf{ect}[1.0] \to \mathsf{ect}[0.0]$ consumes 1.0 level. The inputs to this operation are at Lvl 1.0, which themselves cost 0.5 levels to produce. The total depth required is $0.5 + 1.0 = 1.5$. The three resulting ects are at **Level 0.0**. This is reflected in Table 26.

Operation	Count	Input Level	Output Level
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$ (Promotion)	$5S_{i+1}/2$	0.5	0.0
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$ (Add)	$5S_{i+1}/2 + S_{i+1}/(2^{j'+1})$	0.0, 0.0	0.0

Table 27: FHE Cost for $R_j^{(4)}(X)$ (for all $X \in \{0, 2, 3\}$)

Final Round Polynomial. The prover computes the final round polynomials $R_j(X) = R_j^{(1)}(X) + R_j^{(2)}(X) + R_j^{(3)}(X)$ for $X \in \{0, 2, 3\}$. This requires two additions per X at level 0, a negligible cost we ignore.

Case 2: $j \geq s_{i+1}$ (Second Half of Sumcheck)

For notational convenience, let $j' = j - s_{i+1}$, which means $j' \in \{0, \ldots, s_{i+1} - 1\}$. The j challenges received so far are $\vec{\gamma} = (\vec{\gamma}_{lo}, \vec{\gamma}_{hi})$, where $\vec{\gamma}_{lo} \in \mathbb{F}_{q^R}^{s_{i+1}}$ are the challenges from the first s_{i+1} rounds (Case 1) and $\vec{\gamma}_{hi} \in \mathbb{F}_{q^R}^{j'}$ are the j' challenges from this second half.

We compute once and for all the value $\mathsf{ect}[a] := \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}_{lo})]$. This is a single MLE evaluation based on the challenges from Case 1. We assume this negligible one-time cost is paid, and we store two copies of the resulting ciphertext: one at **Level 1.0** and one at **Level 0.5** for use in different computations.

The round polynomial we need to compute is:

$$R_j(X) = \sum_{\vec{y}} \tilde{\mathsf{add}}_i(\cdots) \cdot \left(\mathsf{ect}[a] + \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}_{hi}, X, \vec{y})]\right) + \tilde{\mathsf{mul}}_i(\vec{r}, \vec{\gamma}, X, \vec{y}) \cdot \left(\mathsf{ect}[a] \cdot \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}_{hi}, X, \vec{y})]\right)$$

Note that $\vec{y}$ ranges over the $s_{i+1} - j' - 1$ remaining Boolean variables for the $\vec{y}$ portion of the $g^{(i)}$ polynomial. We decompose $R_j(X) = R_j^{(4)}(X) + R_j^{(5)}(X)$, where:

$$R_j^{(4)}(X) = \mathsf{ect}[a] \cdot \left(\sum_{\vec{y}} \tilde{\mathsf{add}}_i(\vec{r}, \vec{\gamma}, X, \vec{y})\right) + \sum_{\vec{y}} \tilde{\mathsf{add}}_i(\vec{r}, \vec{\gamma}, X, \vec{y}) \cdot \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}_{hi}, X, \vec{y})]$$

$$R_j^{(5)}(X) = \mathsf{ect}[a] \cdot \sum_{\vec{y}} \tilde{\mathsf{mul}}_i(\vec{r}, \vec{\gamma}, X, \vec{y}) \cdot \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}_{hi}, X, \vec{y})]$$

Calculating $R_j^{(4)}(X)$. This term decomposes into two parts. The first part, $\mathsf{ect}[a] \cdot (\sum \tilde{\mathsf{add}}_i(\ldots))$, is a $\mathsf{ect} \cdot \mathbb{F}_{q^R}$ operation, since the inner sum is a plaintext scalar. We use the $\mathsf{ect}[a]$ copy at **Level 0.5**. This costs a single Promotion (Lvl 0.5 $\to$ Lvl 0.0), which is a negligible cost we ignore.

The second part is $S(X) = \sum_{\vec{y} \in \{0,1\}^{s_{i+1} - j' - 1}} c_y(X) \cdot \mathsf{ect}[\tilde{L}_{i+1}(\vec{\gamma}_{hi}, X, \vec{y})]$, where $c_y(X) = \tilde{\mathsf{add}}_i(\ldots)$ is a plaintext $\mathbb{F}_{q^R}$ scalar. This computation is analogous to $R_j^{(2)}(X)$. We apply Lemma C.2 (with $j = j'$) for each of the $2^{s_{i+1} - j' - 1}$ terms in the sum. The input cts (for $\tilde{L}_{i+1}$) are at Level 0.5. The full cost for computing $R_j^{(4)}(X)$ can be found in Table 27.

Calculating $R_j^{(5)}(X)$. This term is $R_j^{(5)}(X) = \mathsf{ect}[a] \cdot S'(X)$, where $S'(X) = \sum_{\vec{y}} \tilde{\mathsf{mul}}_i(\ldots) \cdot \mathsf{ect}[\tilde{L}_{i+1}(\ldots)]$. This calculation proceeds in two steps:

Operation	Count	Input Level(s)	Output Level
Step 1: Compute $S'(X)$ terms (the inner sum)			
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	$5S_{i+1}/2$	1.5	1.0
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	$5S_{i+1}/2 + S_{i+1}/(2^{j'+1})$	1.0, 1.0	1.0

Table 28: FHE Cost for $R_j^{(5)}(X)$ (for all $X \in \{0, 2, 3\}$)

1. **Compute $S'(X)$:** We compute the inner sum $S'(X)$ for all $X \in \{0, 2, 3\}$. This is analogous to the $R_j^{(4)}$ computation, but we start from the $\mathsf{ct}[L_{i+1}]$ inputs at **Level 1.5** to leave room for the final multiplication.
2. **Compute Final Product:** We multiply $S'(X)$ by $\mathsf{ect}[a]$ (using the copy at **Level 1.0**). For each $X \in \{0, 2, 3\}$, this is one $\mathsf{ect} \cdot \mathsf{ect}$ multiplication at input level **1.0**, a negligible cost compared to step 1, which we ignore.

The full cost for computing $R_j^{(5)}(X)$ can be found in Table 28.

Total cost of all sumcheck rounds for Layer $i \to i+1$ reduction

We now sum the costs over all $2s_{i+1}$ rounds of the sumcheck protocol, carefully separating operations by their type and FHE level. The total cost is the sum of operations from "Case 1" ($j = 0, \ldots, s_{i+1} - 1$) and "Case 2" ($j = s_{i+1}, \ldots, 2s_{i+1} - 1$, with $j' = j - s_{i+1}$).

- **CT-CT Multiplications ($\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$) @ Lvl 1.0 $\to$ 0.0:** This is the depth-critical operation.
  - Case 1 ($R_j^{(3)}$): $\sum_{j=0}^{s_{i+1}-1} \frac{3S_{i+1}}{2^{j+1}} \approx 3S_{i+1}$.

  Total: $\mathbf{3S_{i+1}}$.
- **Promotions ($\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$) @ Lvl 1.5 $\to$ 1.0:** These operations build the inputs for the CT-CT multiplications.
  - Case 1 ($R_j^{(3)}$): $\sum_{j=0}^{s_{i+1}-1}(3S_i + 5S_{i+1}/2) = 3s_{i+1}S_i + s_{i+1}(5S_{i+1}/2)$.
  - Case 2 ($R_j^{(5)}$): $\sum_{j'=0}^{s_{i+1}-1}(5S_{i+1}/2) = s_{i+1}(5S_{i+1}/2)$.

  Total: $\mathbf{3s_{i+1}S_i + 5s_{i+1}S_{i+1}}$.
- **Promotions ($\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$) @ Lvl 0.5 $\to$ 0.0:** These are used for the additive (non-multiplicative) terms.
  - Case 1 ($R_j^{(1)}, R_j^{(2)}$): $\sum_{j=0}^{s_{i+1}-1}(3S_i + 5S_{i+1}/2) = 3s_{i+1}S_i + s_{i+1}(5S_{i+1}/2)$.
  - Case 2 ($R_j^{(4)}$): $\sum_{j'=0}^{s_{i+1}-1}(5S_{i+1}/2) \approx s_{i+1}(5S_{i+1}/2)$. (Ignoring the negligible 3 ops per round).

  Total: $\mathbf{3s_{i+1}S_i + 5s_{i+1}S_{i+1}}$.
- **Additions ($\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$) @ Lvl 1.0:**
  - Case 1 ($R_j^{(3)}$): $\sum_{j=0}^{s_{i+1}-1}(3S_i + 5S_{i+1}/2) = 3s_{i+1}S_i + s_{i+1}(5S_{i+1}/2)$.
  - Case 2 ($R_j^{(5)}$): $\sum_{j'=0}^{s_{i+1}-1}(5S_{i+1}/2 + S_{i+1}/(2^{j'+1})) \approx s_{i+1}(5S_{i+1}/2) + S_{i+1}$.

Operation	Level (Input → Output)	Total Count (Approx.)
$\mathsf{ect} \cdot \mathsf{ect} \to \mathsf{ect}$	**1.0 → 0.0**	$\mathbf{3S_{i+1}}$
$\mathsf{ct} \cdot \mathbb{F}_{q^R} \to \mathsf{ect}$	1.5 → 1.0 0.5 → 0.0	$3s_{i+1}S_i + 5s_{i+1}S_{i+1}$ $3s_{i+1}S_i + 5s_{i+1}S_{i+1}$
Total Promotions		$6s_{i+1}S_i + 10s_{i+1}S_{i+1}$
$\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$	1.0 0.0	$3s_{i+1}S_i + 5s_{i+1}S_{i+1} + S_{i+1}$ $3s_{i+1}S_i + 5s_{i+1}S_{i+1} + 5S_{i+1}$
Total Additions		$6s_{i+1}S_i + 10s_{i+1}S_{i+1} + 6S_{i+1}$

Table 29: Total FHE Cost for one Layer Reduction Sumcheck ($i \to i+1$)

Total: $\mathbf{3s_{i+1}S_i + 5s_{i+1}S_{i+1} + S_{i+1}}$.

- **Additions ($\mathsf{ect} + \mathsf{ect} \to \mathsf{ect}$) @ Lvl 0.0:**
  - Case 1 ($R_j^{(1)}, R_j^{(2)}, R_j^{(3)}$): $\sum_{j=0}^{s_{i+1}-1}(3S_i) + (5S_{i+1}/2 + S_{i+1}/(2^{j+1})) + (3S_{i+1}/(2^{j+1}))$ $\approx 3s_{i+1}S_i + s_{i+1}(5S_{i+1}/2) + S_{i+1} + 3S_{i+1} = 3s_{i+1}S_i + s_{i+1}(5S_{i+1}/2) + 4S_{i+1}$.
  - Case 2 ($R_j^{(4)}$): $\sum_{j'=0}^{s_{i+1}-1}(5S_{i+1}/2 + S_{i+1}/(2^{j'+1})) \approx s_{i+1}(5S_{i+1}/2) + S_{i+1}$.

  Total: $\mathbf{3s_{i+1}S_i + 5s_{i+1}S_{i+1} + 5S_{i+1}}$.

We summarize these total costs for the entire $2s_{i+1}$-round sumcheck in Table 29.

C.1.3 Total cost of GKR prover under FHE

The full $\mathsf{BlindGKR}.P$ algorithm involves running the layer reduction sub-protocol d times, once for each layer $i \in \{0, \ldots, d-1\}$. We now analyze the total cost by summing the concrete costs from Table 29 over all d reductions.

We recall the following parameter notation for the circuit of interest:

- S_i: The number of wires in Layer i.
- s_i: The number of variables for Layer i, i.e. $s_i = \log S_i$.
- $n = \sum_{i=0}^{d} S_i$: The total number of wires in the circuit.
- $s_{\max} = \max_{i=1,\ldots,d}\{s_i\} \le \log n$.

Total Operation Count. We sum the operation counts from Table 29 over $i = 0, \ldots, d-1$. For quasi-linear terms involving s_{i+1}, we use the upper bound $s_{i+1} \le s_{\max}$ to simplify expressions.

- **Total CT-CT Multiplications (Lvl 1.0 → 0.0):** The total count is $\sum_{i=0}^{d-1} 3S_{i+1}$.

$$\sum_{i=0}^{d-1}(3S_{i+1} + 3s_{i+1}) = 3\sum_{i=0}^{d-1} S_{i+1} \le 3n$$

This is bounded by $3n$.

- **Total Promotions (Lvl 1.5 → 1.0):** The count is $\sum_{i=0}^{d-1}(3s_{i+1}S_i + 5s_{i+1}S_{i+1})$.

$$\begin{aligned}\sum_{i=0}^{d-1}(3s_{i+1}S_i + 5s_{i+1}S_{i+1}) &\le s_{\max}\sum_{i=0}^{d-1}(3S_i + 5S_{i+1}) \\ &\le 8n \cdot s_{\max}\end{aligned}$$

- **Total Promotions (Lvl 0.5 → 0.0):** This cost comes from two sources: the sumcheck (Table 29) and the "final round" MLE evaluations (costing S_{i+1} ops per layer).

$$\begin{aligned}\text{Cost} &= \sum_{i=0}^{d-1}(3s_{i+1}S_i + 5s_{i+1}S_{i+1}) + \sum_{i=0}^{d-1} S_{i+1} \\ &\le s_{\max}(3n_0 + 5n') + n' \\ &= \mathbf{s_{\max}(3n_0 + 5n') + n'} \\ &\le s_{\max} \cdot 8n + n \\ &\approx 8n \cdot s_{\max}\end{aligned}$$

- **Total Additions (Lvl 1.0):** The full count is $\sum_{i=0}^{d-1}(3s_{i+1}S_i + 5s_{i+1}S_{i+1} + S_{i+1})$.

$$\begin{aligned}\text{Cost} &= \sum_{i=0}^{d-1}(3s_{i+1}S_i + 5s_{i+1}S_{i+1}) + \sum_{i=0}^{d-1} S_{i+1} \\ &\le s_{\max}(8n) + n \\ &\approx 8n \cdot s_{\max}\end{aligned}$$

- **Total Additions (Lvl 0.0):** The full count is $\sum_{i=0}^{d-1}(3s_{i+1}S_i + 5s_{i+1}S_{i+1} + 5S_{i+1})$.

$$\begin{aligned}\text{Cost} &= \sum_{i=0}^{d-1}(3s_{i+1}S_i + 5s_{i+1}S_{i+1}) + \sum_{i=0}^{d-1} 5S_{i+1} \\ &\le s_{\max}(8n) + n \\ &\approx 8n \cdot s_{\max}\end{aligned}$$

These operation counts are summarized in Table 8.

C.2 Folded BhIOP Prover FHE Operations for Rotation-Free Payload Circuits

In this subsection, we derive the operation counts presented in Table 9.

C.2.1 Compute-Only Layer Reduction Operation Counts for Folded BhIOP Prover

Fix layer $i \in \{0, \ldots, d-1\}$. The layer reduction consists of $2s_{i+1} + \log N$ rounds of sumcheck, where we first process the x, y variables, then the ℓ variables.

Case 1: $0 \le j < 2s_{i+1}$. The blind prover needs to compute round polynomial $R_j(X)$ for $X \in \{0, 2, 3\}$. The following observation is crucial.

$$\mathsf{emask}[R_j(X)] \equiv \mathsf{FOLD}\left(\mathsf{ept}[c] \cdot \mathsf{ect}[R_j(X)]\right),$$

where

- $\mathsf{ect}[R_j(X)]$ are the N round-j polynomials (one per slot) that would be computed in the *base* BhIOP's corresponding layer reduction sumcheck,
- and $\mathsf{ept}[c] \equiv \left(\tilde{\mathsf{eq}}(\vec{\psi}, 0^{\log N}), \cdots, \tilde{\mathsf{eq}}(\vec{\psi}, 1^{\log N})\right)$.

Thus, $R_j(X)$ can be computed by computing $R_j(X)$, followed by a plaintext multiplication (introducing 1 level), followed by a FOLD operation (introducing θ_{fold} levels).

Case 2: $2s_{i+1} \leq j < 2s_{i+1} + \log N$. Let $\vec{r} \in \mathbb{F}_{q^R}^{2s_{i+1}}$ denote the first $2s_{i+1}$ sumcheck challenges, and let $\vec{\lambda} \in \mathbb{F}_{q^R}^{j-2s_{i+1}}$ denote the remaining received sumcheck challenges thus far. We can exploit an optimization due to Gruen (see Appendix B). The consequence is that we need only compute a degree 2 round polynomial at $X \in \{0, 2\}$. Let's use the following shorthand

- $a := \tilde{\mathsf{add}}(\vec{\gamma}, \vec{r})$, $m := \tilde{\mathsf{mul}}(\vec{\gamma}, \vec{r})$.
- $c_l := \tilde{\mathsf{eq}}(\vec{\ell}, \psi_{j+1}, \ldots, \psi_{\log N - 1})$
- $w := L_i(r_0, \ldots, r_{s_{i+1}-1}, \cdots)$
- $z := L_i(r_{s_{i+1}}, \ldots, r_{2s_{i+1}-1}, \cdots)$

The blind prover needs to compute

$$\begin{aligned}
\mathsf{emask}[R_j(X)] &= \mathsf{emask}[A_j(X) + M_j(X)] \\
\mathsf{emask}[A_j(X)] &:= \sum_{\vec{\ell} \in \{0,1\}^{\log N - j - 1}} a \cdot c_l \cdot (\mathsf{emask}[\tilde{w}(\vec{\lambda}, X, \vec{\ell})] + \mathsf{emask}[\tilde{z}(\vec{\lambda}, X, \vec{\ell})]) \\
\mathsf{emask}[M_j(X)] &:= \sum_{\vec{\ell} \in \{0,1\}^{\log N - j - 1}} m \cdot c_l \cdot \mathsf{emask}[\tilde{w}(\vec{\lambda}, X, \vec{\ell})] \cdot \mathsf{emask}[\tilde{z}(\vec{\lambda}, X, \vec{\ell})]
\end{aligned}$$

For all $x \in \{0,1\}^{\log N}$, let $d_x := \tilde{\mathsf{eq}}(\vec{x}, (\vec{\lambda}, X, \psi_{j+1}, \ldots, \psi_{\log N - 1}))$. Let plaintext $\mathsf{ept}[a]$ be equivalent to the following list of $\mathbb{F}_{q^R}$ elements: $(a \cdot d_{0^{\log N}}, \cdots, a \cdot d_{1^{\log N}})$. Let plaintext $\mathsf{ept}[m]$ be equivalent to the following list of $\mathbb{F}_{q^R}$ elements: $(m \cdot d_{0^{\log N}}, \cdots, m \cdot d_{1^{\log N}})$.

Observe the following:

$$\mathsf{emask}[A_j(X) + M_j(X)] = \mathsf{FOLD}\left(\mathsf{ept}[a] \cdot (\mathsf{ect}[w] + \mathsf{ect}[z])\right] + \mathsf{ept}[m] \cdot \mathsf{ect}[w] \cdot \mathsf{ect}[z])$$

Thus computing $R_j(X)$ at $X = 0, 2$ requires:

- two $\mathsf{ect} \cdot \mathsf{ect}$ multiplications and two $\mathsf{ect} + \mathsf{ect}$ additions
- Followed by four $\mathsf{ect} \cdot \mathsf{ept}$ multiplications
- Followed by two FOLD operators.

C.2.2 Total Operation Counts

The operation counts presented in Table 9 follow from the Layer Reduction Operation Counts presented above and the following two observations:

- The protocol from Section 4.2.1 has fewer than $2ds_{\max}$ sumcheck rounds.
- There are $d \log N$ additional sumcheck rounds.

C.3 Base BhIOP Prover FHE Operations for (c, f)-Layered Payload Circuits

In this subsection, we derive the operation counts presented in Table 12.

Notation Recap We fix once and for all a (c, f)-payload circuit Ckt (with c compute-only layers and f forward-only layers). We let $d = c + f$ denote the total number of gate layers. We partition the set of gate layers $\{0, \dots, d-1\}$ into $\mathcal{C}$ and $\mathcal{F}$ (compute and forward respectively).

S_i denotes the number of ciphertexts required to represent layer i and $T_i = S_i \cdot N$ denotes the number of wires in the ith layer. As usual, $s_i := \log_2(S_i)$ and $t_i := \log_2(T_i) = s_i + \log N$.

We let $n_c := \sum_{i \in \mathcal{C}} S_i$ be the total number of compute gates and $n_f := \sum_{i \in \mathcal{F}} S_i$ be the total number of forward gates. We let $n := \sum_{0 \le i \le d} S_i$ be the total number of gates (including inputs). Observe that $n = n_c + n_f + S_d$.

C.3.1 Operation Counts for Compute-Only Layer Reductions

The compute layer reduction is identical to the layer reduction presented in Appendix C.1.

For ease of accounting, we use a very loose upper bound when totaling the operation counts across all compute-only layers, namely assuming that there are *no forwarding gates*. This allows us to simply copy the operation counts derived previously.

C.3.2 Operation Counts for Forward-Only Layer Reduction

Fix $i \in \mathcal{F}$ and fix a sumcheck round $j \in \{0, \dots, s_{i+1} - 1\}$. For $X \in \{0, 2\}$ the blind verifier needs to learn the following:

$$\begin{aligned} R_j(X) &= \sum_{x \in \{0,1\}^{s_{i+1}-j-1}} \sum_{l \in \{0,1\}^{\log N}} \tilde{\mathsf{fwd}}((\vec{\gamma}, \vec{\psi}), (\vec{r}, X, \vec{x}, \vec{l})) \cdot \tilde{L}_{i+1}(\vec{r}, X, \vec{x}, \vec{l}) \\ &= \sum_x \sum_w \sum_l c_{x,w,l} \cdot L_{i+1}(\vec{w}, \vec{x}, \vec{l}) \end{aligned}$$

In the above equations,

- $\vec{r} \in \mathbb{F}_{q^R}^j$ represents the received sumcheck challenges thus far.
- $c_{x,w,l} = \tilde{\mathsf{fwd}}((\vec{\gamma}, \vec{\psi}), (\vec{r}, X, \vec{x}, \vec{l})) \cdot \tilde{\mathsf{eq}}((\vec{r}, X), \vec{w})$

  From the view of the blind prover,

$$\mathsf{emask}[R_j(X)] = \mathsf{FOLD}\left(\sum_{y \in \{0,1\}^{s_{i+1}}} \mathsf{ept}[c_y] \cdot \mathsf{ct}[L_{i+1}(y)] \right)$$

So, in each of the s_{i+1} rounds, the (base) blind prover computes $2S_{i+1}$ multiplications of the form $\mathsf{ct} \cdot \mathsf{ept}$ at level 1, then $2S_{i+1}$ additions of the form $\mathsf{ect} + \mathsf{ect}$ at level 0. This ciphertext is given to the blind verifier, who decrypts and performs a fold on the plaintext.

Across all forward-only layers, the blind prover does fewer than $2n_f \cdot s_{\max}$ multiplications of the form $\mathsf{ct} \cdot \mathsf{ept}$ at level 1 and the same number of $\mathsf{ect} + \mathsf{ect}$ additions at level 0.

C.4 Folded BhIOP Prover FHE Operations for (c, f)-Layered Payload Circuits

In this subsection, we derive the operation counts presented in Table 13.

C.4.1 Operation Counts for Compute-Only Layer Reductions

The compute layer reduction is identical to the layer reduction presented in Appendix C.2.

As before, we use a very loose upper bound when totaling the operation counts across all compute-only layers, namely assuming that there are *no forwarding gates*. This allows us to simply copy the operation counts derived previously.

C.4.2 Operation Counts for Forward-Only Layer Reductions

Fix forward-only layer index $i \in \mathcal{F}$ and sumcheck round $j \in \{0, \ldots, s_{i+1} + \log N - 1\}$.

Case 1: $0 \leq j < s_{i+1}$ For these sumcheck rounds, the folded blind prover does exactly what the base blind prover does, just at θ_{fold} levels higher. Then, the folded blind prover performs a fold operation. Overall, this tallies to

- $2S_{i+1}$ $\mathsf{ept} \cdot \mathsf{ct}$ multiplications at level $\theta_{\text{fold}} + 1$.
- $2S_{i+1}$ $\mathsf{ect} + \mathsf{ect}$ additions at level θ_{fold}
- $2R$ FOLD operations at level θ_{fold}.

Case 2: $s_{i+1} \leq j < s_{i+1} + \log N$ For each of these additional[31] sumcheck rounds, the folded prover performs

- 2 $\mathsf{ect} \cdot \mathsf{ect}$ multiplications at level $\theta_{\text{fold}} + 2$
- 2 $\mathsf{ect} + \mathsf{ect}$ additions at level $\theta_{\text{fold}} + 1$
- 4 $\mathsf{ept} \cdot \mathsf{ect}$ multiplications at level $\theta_{\text{fold}} + 1$
- 2 $\mathsf{ect} + \mathsf{ect}$ additions at level θ_{fold}
- $2R$ FOLD operations at level θ_{fold}

Totaling Operation Counts in one sumcheck. Overall, the blind prover does work bounded by

- $2 \log N$ $\mathsf{ect} \cdot \mathsf{ect}$ multiplications at level $\theta_{\text{fold}} + 2$
- $2 \log N$ $\mathsf{ect} + \mathsf{ect}$ additions at level $\theta_{\text{fold}} + 1$
- $2(S_{i+1} s_{\max} + 2 \log N)$ $\mathsf{ept} \cdot \mathsf{ct}$ multiplications at level $\theta_{\text{fold}} + 1$.
- $2(S_{i+1} s_{\max} + \log N)$ $\mathsf{ect} + \mathsf{ect}$ additions at level θ_{fold}
- $2R(s_{\max} + \log N)$ FOLD operations at level θ_{fold}.

Totaling Operation Counts across all forward-only sumchecks. Crudely bounding $\sum_{i \in \mathcal{F}} S_{i+1} \leq n$, we bound the blind prover work across all forward-only sumchecks like so:

- $2f \log N$ $\mathsf{ect} \cdot \mathsf{ect}$ multiplications at level $\theta_{\text{fold}} + 2$

[31] The base protocol skips these rounds.

- $2f \log N$ $\mathsf{ect} + \mathsf{ect}$ additions at level $\theta_{\text{fold}} + 1$
- $2(ns_{\max} + 2f \log N)$ $\mathsf{ept} \cdot \mathsf{ct}$ multiplications at level $\theta_{\text{fold}} + 1$.
- $2(ns_{\max} + f)$ $\mathsf{ect} + \mathsf{ect}$ additions at level θ_{fold}
- $2Rf(s_{\max} + \log N)$ FOLD operations at level θ_{fold}.