Packed Sumcheck over Fields of Small Characteristic

Yuanju Wei[1,2], Kaixuan Wang[3], Binwu Xiang[4,5], Xinxuan Zhang[1,2], Yi Deng[1,2], Xudong Zhu[1,2], Hailong Wang[6], Li Lin[6], and Lei Wang[3]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] Shanghai Jiao Tong University
[4] East China Normal University
[5] State Key Laboratory of Cryptology
[6] Digital Technologies, Ant Group
{weiyuanju, xiangbinwu, zhangxinxuan, deng, zhuxudong}@iie.ac.cn, {wangkaixuan, wanglei_hb}@sjtu.edu.cn, whl383799@antgroup.com, felix.ll@alibaba-inc.com

Abstract. The sumcheck protocol is a fundamental primitive in the construction of probabilistic proof systems. Its soundness relies on the Schwartz–Zippel lemma and is therefore typically instantiated over large fields. However, proving small field computations such as FHE via sumcheck over large fields incurs the substantial overhead of large field arithmetic. So there is strong interest in sumcheck over small fields. In this work, we present *packed sumcheck*, a new protocol for small prime fields that combines repetition with folding to preserve round-by-round soundness while keeping all computations in the base field $\mathbb{F}_p$. Under the assumption $p^k = O(2^\lambda)$, where λ is the security parameter, proving a sumcheck instance consisting of the product of d multilinear polynomial over $(\log k + l)$ variables (with $N = 2^l$ and instance size kN) requires $O((kd^2+k^2d)N)$ multiplications and $O(k^2d^2N)$ additions over $\mathbb{F}_p$. While the standard approach requires $O(k^3d^2N)$ operations over $\mathbb{F}_p$. For concrete performance, we instantiate our protocol over the Babybear field and obtain at least a 2.78× speedup over the state of the art.

As a direct application, we design a TFHE-friendly SNARK. We express TFHE bootstrapping (J. Cryptol 2020) as a collection of vector relations, commit them using a variant of Brakedown (CRYPTO 2023) and Binius (EUROCRYPT 2025), and verify them via our packed sumcheck. The experiments demonstrate a proof generation time of 2.02s for a single bootstrapping, surpassing the best previously reported results.

Keywords: SNARKs · Sumcheck · Verifiable TFHE.

1 Introduction

The sumcheck protocol [44] is a public-coin interactive proof that enables a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ of claims of the form $\sum_{x\in\{0,1\}^\ell} f(\mathbf{x}) = h$, where

f is an ℓ-variate polynomial of total degree at most d over a finite field $\mathbb{F}_p$, and $h \in \mathbb{F}_p$ is the claimed sum. It is the core component of succinct non-interactive arguments of knowledge (SNARKs) [54,49,8,7,9,35,55,56], where the sumcheck protocol is transformed into a non-interactive proof via the Fiat–Shamir transformation [54,49,35,55]. The soundness of this transformation usually requires round-by-round soundness [14]. Informally, the round-by-round soundness error is the probability that, in any round, a prover can transform a transcript of a false instance into one that the verifier accepts. For sumcheck, this property follows from the Schwartz–Zippel lemma [50]: a verifier's random challenge detects the false instance with probability at least $1 - \frac{d}{|\mathbb{F}_p|}$. Consequently, the field size directly controls the round-by-round soundness error and protocols are therefore typically instantiated over large finite fields to ensure the desired guarantee.

Motivation. Arithmetic operations over large finite fields are not efficient on modern CPUs. By contrast, computations over small prime fields, such as the 32-bit Babybear and 64-bit Goldilocks, fit in native machine words and admit highly optimized implementations. Many advanced and urgent applications (notably, FHE [31,11,12,32,19,24]) operate over such small fields and require sumcheck for verification. For example, TFHE [20] ciphertext operations are defined over 32- or 64-bit fields. Proving TFHE directly over small fields results in a relatively large round-by-round soundness error. A typical approach is to work over an extension field $\mathbb{F}_{p^k}$, which reduces the error to $d/|\mathbb{F}|^k$ [42]. However, this reintroduces the cost of extension field arithmetic. So it is important to develop a sumcheck protocol that is both secure and efficient over small finite fields.

A growing line of work studies sumcheck over small fields [5,37]. These constructions still rely on field extensions to ensure round-by-round soundness, but they mitigate overhead by replacing extension field operations with base field computations via algebraic techniques. However, such substitutions introduce many base field multiplications. Bagad, Domb and Thaler [5] show that the number of these multiplications grows exponentially with the number of sumcheck rounds. Consequently, the technique is practical only for the initial rounds; the later rounds must still be executed over the extension field.

One approach to avoiding extension fields is to increase soundness via repetition. However, naive sequential and parallel repetition are unsuitable for sumcheck over small fields. Bagad, Domb and Thaler [5] show that,[7] under the Fiat–Shamir transformation, sequential repetition does not increase security. Parallel repetition, in turn, requires an impractically large number of repetitions: an l-round protocol needs $k \cdot l$ repetitions to gain a factor-k increase in soundness, which is prohibitive for typical $l \geq 20$. This motivates the following question:

Is there a repetition strategy for sumcheck over small fields that simultaneously achieve efficiency and round-by-round soundness?

[7] https://a16zcrypto.com/posts/article/17-misconceptions-about-snarks/#section--13

1.1 Our Results

In this work, we answer this question by presenting the *packed sumcheck* protocol that performs all computations natively in the base field while preserving round-by-round soundness. Our core technique combines a forward–difference–based folding scheme with challenge repetition, thereby avoiding field extensions while preserving round-by-round soundness. Specifically, under the assumption $p^k = O(2^\lambda)$, proving a degree-d sumcheck instance of size kN (each multilinear polynomial has $\log(kN)$ variables) by packed sumcheck requires

$$O((kd^2 + k^2 d)N) \cdot \mathsf{mul} + O(k^2 d^2 N) \cdot \mathsf{add},$$

where mul and add denote *base field operations*.[8] While the standard approach needs $O(kd^2N) \cdot \mathsf{MUL} + O(kd^2N) \cdot \mathsf{ADD}$, where MUL and ADD denote *extension field operations*. [9]

We instantiate our protocol over the Babybear field $\mathbb{F}_{2^{31}-2^{27}+1}$ and set the repetition parameter to $k \in \{4, 5\}$, targeting ≥ 100-bit and ≥ 128-bit security, respectively. Compared with the fastest algorithm over Babybear filed in [5] (using a 4-fold extension to guarantee ≥ 121-bit security), our protocol achieves $3.25\times$–$4.03\times$ speedup when $k = 4$ and $2.78\times$–$3.36\times$ speedup when $k = 5$. See Tab. 10(a) for details. Beyond efficiency improvement, our protocol also offers greater flexibility in parameter selection, as it does not rely on any specific extension field structure.

As a direct application, we construct a TFHE-friendly SNARK. First, we adapt the TFHE [20] framework to support our SNARK construction through careful parameter design. Second, we decompose the core homomorphic operations into a sequence of vector relations, which are committed using a variant of Brakedown [35] and Binius [23], and efficiently verified via our packed sumcheck protocol. Experimental evaluation shows that we can generate a proof for a single TFHE bootstrapping in 2.02 seconds, which is $534\times$ faster than the work of Thibault and Walter (CCS 2025) [51] and $1.98\times$ faster than Hasteboots [42].

1.2 Technique Overview

We start with explaining the known attacks [3] on the Fiat-Shamir transformation of simple repetitions of the sumcheck protocol over small field in a unified and theoretic prospective.

Do not Apply Fiat-Shamir to Practically Simulatable Protocols. Though the security of the Fiat-Shamir transformation in general still remains unclear, it is known, among other negative results [15,34], that this transformation is completely insecure when applying it to public-coin zero knowledge protocols [6,25].

[8] Applying Strassen's algorithm to the arising matrix multiplications reduces the complexity to $O((kd^2 + k^{1.8}d)N) \cdot \mathsf{mul} + O((k^2d^2)N) \cdot \mathsf{add}$ for $(k \geq 2)$.

[9] One MUL in a k-fold field extension typically costs $O(k^2)(\mathsf{mul} + \mathsf{add})$. For certain structured extensions, this can be reduced to $O(k^{1.58})$ via the Karatsuba algorithm.

Recall that if a protocol (P, V) is zero knowledge, it admits an efficient simulator that can somehow make the verifier accept a false (hard) instance. Thus, when doing the Fiat-Shamir transformation to (P, V) using a hash function $\mathcal{H}$, we obtain a non-interactive protocol for which the same simulator works by treating $\mathcal{H}$ as the malicious verifier, and this will render the non-interactive version *unsound* completely, since a malicious prover can take the simulation strategy and cheat on a false instance.

This applies to generally-defined simulatable protocols. Intuitively, if a public-coin protocol (P, V) admits a black-box simulator that runs time T, then for the non-interactive version of (P, V) obtained via the Fiat-Shamir, a malicious prover P^*, following the same strategy of the simulator, will cheat on some false instance with probability close to 1.

As showed in [33], both sequential and parallel repetitions of public-coin *proofs* reduce the soundness error exponentially: repeating (in sequential or parallel) a public-coin proof with soundness error ϵ results in a proof system with soundness error ϵ^k. This applies the sumcheck protocol over small field in the standard model.

However, the sequential/parallel repetition of the sumcheck protocol can be simulated much more efficiently than expected[10], and as explained above, this will lead to concrete attacks [3] on the soundness of the non-interactive version of these repetitions obtained via the Fiat-Shamir transformation[11].

Specifically, Consider a degree-d sumcheck instance over $\mathbb{F}_p$, $\sum_{\mathbf{x}\in\{0,1\}^\ell} f(\mathbf{x}) = h$. For the k interactive sequential repetitions, there exists a simulator $\mathcal{S}_s$ with expected running time $O(\frac{kp}{d})$: it attacks only the first round of each repetition; the attack succeeds with probability $\frac{d}{p}$, so each repetition requires on average $\frac{p}{d}$ rewindings. For the k interactive parallel repetitions: there exists a simulator $\mathcal{S}_p$ with expected running time $O(l(\frac{p}{d})^{k/l})$: it targets only k/l instances per round, yielding success probability $(\frac{d}{p})^{k/l}$, so each round requires on average $(\frac{p}{d})^{k/l}$ rewindings. Consequently, after applying the Fiat-Shamir transformation, k sequential and k parallel repetitions admit adversaries with expected running times $O(\frac{kp}{d})$ and $O(l(\frac{p}{d})^{k/l})$ respectiverly, that succeed with probability close to 1. When p is small, these running times can be practical.

Combining Folding and Repetition for Sumcheck. A folding scheme [41] compresses m instances into one of the same size. *Folding scheme guarantees that the folded instance is true only if all the original m instances are true.*

In the setting of k interactive parallelized sumcheck over $\mathbb{F}_p$, the simulator/attack $\mathcal{S}_p$ succeeds due to two reasons: 1) acceptance of a false instance in a repetition is persistent across rounds. Once $\mathcal{S}_p$ produces an accepting transcript for a false instance in some round, the verifier will accept that instance in all

[10] Notice that sequential repetition preserves zero knowledge, and the running time of the corresponding simulator increases just linearly in the number of repetitions.

[11] Though presented as concrete attacks in [3], these algorithms are, in fact, the simulators for the corresponding sequential and parallel repetitions of the sumcheck protocol in the plain model.

subsequent rounds in the same repetution; and, 2) At the end of each rewinding, the simulator can check if a single repetition is accepting. This allows the simulator to focus on a single round of a few repetitions (k/l among those k repetions) and rewind on it until these target repetitions are all accepting, which makes the simulator/attack more efficient.

To defend this simulation/attack strategy, we propose a packed sumcheck protocol which combine folding scheme with k interactive parallel repetition. In the first round, given a single instance, the prover replicates it into k copies and proves these k instances in parallel. In each subsequent round, the prover folds k parallel instance into a single instance. To verify this folded instance, the verifier provides k parallel challenges, which induce k concurrent checks. Consequently, the prover again holds k parallel instances to be proved in the next round. Moreover, our packed sumcheck protocol can naturally extend to proving k distinct instances, which can be regarded as a "batch" sumcheck protocol. This protocol improves the prover efficiency by k times.

This repetition approach defends the simulator/attack. If the simulator/attack only cheats on a subset of the false sub-instances in one round and leaves at least one false sub-instance unchanged, the folded instance remains false by the property of folding scheme. This makes the two reasons for the success of simulator/attack invalid. First, the folded instance itself is false and the acceptance is not persistent. Second, because the verifier uses k challenges to verify the same folded instance, the simulator/attack cannot process the k challenges separately to determine which of the k instances before folding was accepted. The simulator/attack needs to cheat on all k challenges in one round to make the verifier accept the folded instance, allowing the simulation/attack process to continue. According to the Schwartz–Zippel lemma, the round success probability is at most $(d/p)^k$. Therefore, the expected number of rewindings in one round is $O\big((p/d)^k\big)$, so the expected running time of the simulator/attack is $O\big(l(p/d)^k\big)$. This achieves the purpose of enhancing security with k parallel repetitions.

The Packed Sumcheck Protocol. In practical applications, the sumcheck protocol is typically used to verify polynomial summation expressed as the product of d multilinear polynomials. We consider the setting over $\mathbb{F}_p$ with the field size condition $|\mathbb{F}_p|^k = O(2^\lambda)$. And the prover needs to prove a degree d-sumcheck instance of size kN (each multilinear polynomial has $\lceil \log(kN) \rceil$ variables)[12].

We build on the idea of combining folding with repetition to construct our packed sumcheck protocol. Firstly, the prover decomposes an instance of size kN

[12] In practice, for any integers M and k, one can partition M items into k blocks, each of size at most $\lceil M/k \rceil$; hence expressing the instance size as kN is merely a notational convenience for describing the protocol. Likewise, our convention that the sumcheck relation is a product of d multilinear polynomials is adopted purely for expository clarity; in general, a sumcheck instance may involve any constant-size combination of sums and products of multilinear polynomials, as detailed in Remark 2.

into $2k$ sub-instances, each of size $N/2$. With $2^l = N$, this yields $2k$ relations:

$$\sum_{\mathbf{x}\in\{0,1\}^l} \prod_{j=0}^{d-1} f_0^{(j)}(\mathbf{x}) = h_0, \cdots, \sum_{\mathbf{x}\in\{0,1\}^l} \prod_{j=0}^{d-1} f_{2k-1}^{(j)}(\mathbf{x}) = h_{2k-1},$$

We adapt Gruen's univariate skip [37] to the packed sumcheck, replacing the multilinear polynomial in "sumfold" [40] with univariate interpolation. Using univariate polynomials avoids the heavier representation of multilinear polynomials, thereby reducing the prover's interpolation cost. Specifically, the prover folds these $2k$ instances into a single univariate polynomial. For a subset $W = \{w_0, \ldots, w_{2k-1}\} \subset \mathbb{F}_p$, we construct Lagrange basis polynomials $L_0, \ldots, L_{2k-1}$ of degree $(2k-1)$, satisfying $L_i(w_j) = \delta_{ij}$, where δ_{ij} equals 1 when $i = j$ and 0 otherwise. These are used to interpolate a combined relation into a univariate polynomial $F(r)$:

$$F(r) = \sum_{\mathbf{x}\in\{0,1\}^l} \prod_{s=0}^{d-1} \left(\sum_{i=0}^{2k-1} L_i(r) f_i^{(s)}(\mathbf{x}) \right).$$

The prover sends to the verifier the evaluations of F at $d(2k-1)$ distinct points, after which the verifier reconstructs F via polynomial interpolation. Then the verifier checks $F(w_i) = h_i$ for all $i \in [0, 2k-1]$, then samples k random challenges to further test $F(r)$. After receiving these challenges and updating the polynomials, the prover must simultaneously prove k sumcheck instances of size $N/2$, which further decompose into $2k$ instances of size $N/4$. The protocol proceeds recursively.

We now analyze the round-by-round soundness. If the prover cheats in a given round, at least one of the $2k$ sub-instances must be incorrect. In this case, the prover can interpolate only a different polynomial $G(r) \neq F(r)$. Since both $G(r)$ and $F(r)$ have degree at most $d(2k-1)$, the Schwartz–Zippel lemma implies that the probability of $G(r_0) = F(r_0)$ for a random $r_0 \in \mathbb{F}_p$ is bounded by $\frac{d(2k-1)}{p}$. Because the verifier samples k independent random points, the probability that all equalities hold is at most $\frac{(d(2k-1))^k}{p^k}$. Thus, in each round, the probability that the prover successfully converts an incorrect instance into a valid one is bounded by $\frac{(d(2k-1))^k}{p^k}$, establishing round-by-round soundness.

Forward Differences for Efficient Folding. In the packed sumcheck protocol, the prover's most computationally intensive step is computing the coefficients of $F(r)$, a polynomial of degree $d(2k-1)$. We observe that forward difference method can be used to accelerate this computation and derive the overall complexity of the packed sumcheck prover.

The forward difference method is a classical numerical tool for accelerating polynomial evaluation at sequences of equally spaced points [36]. By replacing multiplications with structured additions, it enables efficient computation of successive values, thereby significantly improving prover efficiency.

For a univariate polynomial f of degree $(k-1)$ evaluated at integer points $j \in \mathbb{F}_p$, the forward differences satisfy [30]:

$$\Delta^i f(j) = \begin{cases} f(j), & i = 0, \\ \Delta^{i-1} f(j+1) - \Delta^{i-1} f(j), & i > 0. \end{cases}$$

A fundamental property of forward differences is that, for any univariate polynomial of degree at most $k-1$, its $(k-1)$-th forward difference is constant [36]. Given the sequence $\Delta^0 f(j), \Delta^1 f(j-1), \ldots, \Delta^{k-1} f(j-k+1)$, one can compute $f(j+1)$ using only additions by forward difference:

$$f(j+1) = \Delta^0 f(j) + \Delta^1 f(j-1) + \cdots + \Delta^{k-1} f(j-k+1),$$

When computing $f(j+1)$, we can simultaneously maintain the forward differences $\Delta^1 f(j), \ldots, \Delta^{k-1} f(j-k+2)$ so that the same routine immediately gets $f(j+2)$, and so forth. Specifically, during the evaluation of $f(j+s)$, let $\Delta^i_{\text{prev}} \coloneqq \Delta^i f\big(j+s-(i+1)\big)$ denote the i-th forward difference carried from the previous step, and let $\Delta^i_{\text{new}} \coloneqq \Delta^i f\big(j+s-i\big)$ be the updated value. Because the $(k-1)$-st forward difference is a constant c, we have $\Delta^{k-1}_{\text{prev}} = \Delta^{k-1}_{\text{new}} = c$. For $0 \le i \le k-2$, the differences satisfy the update rule: $\Delta^i_{\text{new}} = \Delta^i_{\text{prev}} + \Delta^{i+1}_{\text{new}}$. After obtaining $f(j+s)$, set $\Delta^i_{\text{prev}} \leftarrow \Delta^i_{\text{new}}$ for all i, enabling the next step to compute $f(j+s+1)$ identically. Hence, given the initial values $f(0), \ldots, f(k-1)$, all subsequent evaluations of f can be generated using additions only.

We apply the forward difference method to the packed sumcheck, where the prover's most computationally intensive step is computing the coefficients of:

$$F(r) = \sum_{\mathbf{x} \in \{0,1\}^l} \prod_{s=0}^{d-1} \left(\sum_{i=0}^{2k-1} L_i(r) f_i^{(s)}(\mathbf{x}) \right).$$

Define the inner polynomials: $\phi_{\mathbf{x}}^{(s)}(r) = \sum_{i=0}^{2k-1} L_i(r) f_i^{(s)}(\mathbf{x})$. Each $\phi_{\mathbf{x}}^{(s)}(r)$ is a polynomial of degree $(2k-1)$ satisfying $\phi_{\mathbf{x}}^{(s)}(w_i) = f_i^{(s)}(\mathbf{x})$ for $i \in [0, 2k-1]$. By selecting an equally spaced evaluation set W (e.g., $W = 0, \ldots, 2k-1$), the prover can apply the forward difference method to efficiently compute the sequence of values $\phi_{\mathbf{x}}^{(s)}(t)$ for $t \in [0, d(2k-1)]$, incurring $O(k^2 d^2 N)$ additions. We then define $\phi_{\mathbf{x}}(r) = \prod_{s=0}^{d-1} \phi_{\mathbf{x}}^{(s)}(r)$. Subsequently, for each $t \in [0, d(2k-1)]$, the prover computes $\phi_{\mathbf{x}}(t)$, which requires an additional $O(kd^2 N)$ multiplications. Finally, the prover evaluates $F(t) = \sum_{\mathbf{x}} \phi_{\mathbf{x}}(t), t \in [0, d(2k-1)]$, which incurs $O(d(2k-1)N)$ further additions. Hence, the overall cost of evaluating $F(r)$ is $O(kd^2 N)$ multiplications and $O(k^2 d^2 N)$ additions. After receiving the challenges $r_0, \ldots, r_{k-1}$, the prover updates the polynomials

$$f_j'^{(s)}(\mathbf{x}) = \sum_{i=0}^{2k-1} L_i(r_j)\, f_i^{(s)}(\mathbf{x}), \quad j \in [0, k-1],\ s \in [0, d-1],$$

which requires $O(k^2 dN)$ field multiplications and additions. Therefore, the prover's computational cost in the first round is $O((kd^2 + k^2 d)N) \cdot \mathsf{mul} + O(k^2 d^2 N) \cdot \mathsf{add}$.

Because each round reduces the size of instance by half, the total cost of the packed sumcheck protocol is bounded by twice the first-round cost.

This complexity can be further optimized. The polynomial update step consists of structured linear combinations, which can be accelerated using Strassen's algorithm. Even for small k, this method reduces the number of multiplications, yielding a cost of $O(k^{1.8}dN)$. The overall prover complexity then becomes $O((kd^2 + k^2d)N) \cdot \mathsf{mul} + O(k^{1.8}d^2N) \cdot \mathsf{add}$.

A TFHE-friendly SNARK. Given an LWE ciphertext $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, TFHE bootstrapping consists of four steps: (i) *Modulus switching.* Map (a_i, b) to $(\tilde{a}_i, \tilde{b}) \in \mathbb{Z}_{2N_R}^{n+1}$. (ii) *Blind rotation.* Apply the embedding $\mathcal{F} : \mathbb{Z}_{2N_R} \to R$, where $\mathcal{F}(\alpha) = X^\alpha$ for $\alpha \in \{\tilde{a}_0, \ldots, \tilde{a}_{n-1}, \tilde{b}\}$, and execute the refresh procedure to obtain an RLWE ciphertext with reduced noise. (iii) *Sample extraction.* Transform the RLWE output into a length-N_R LWE ciphertext over modulus Q. (iv) *Modulus/key switching.* Convert the modulus/keys back to (q, n).

We instantiate $N_R = n = 1024$ and set $q = Q = p = 2^{31} - 2^{27} + 1$, an NTT-friendly prime. In this way, the secret key and ciphertext moduli of LWE and RLWE can be kept consistent, incurring only a minor computational overhead for FHE. Thus, neither key switching nor modulus switching is required after blind rotation; and since sample extraction is essentially free, we delegate it to the verifier as in [51]. Thus, only steps (i)-(ii) require proofs.

All computations arising in step (i)-(ii) are expressed as a sequence of vector relations. Commitments to the these vectors are instantiated with a variant of Brakedown [35] and Binius [23], and the relations are verified using our packed sumcheck. We now describe how to prove the step (i)-(ii) one by one. An overview of our SNARK workflow is provided in Fig. 1.

Verifiable Modulus Switching. To prove modulus switching, it suffices to prove the rounding operation $\beta = \left\lfloor \frac{\alpha \cdot (2N_R)}{q} \right\rceil$. Because rounding is cumbersome to express with simple arithmetic constraints, we prove it via a table lookup. Assume $2N_R \mid (q-1)$, which is consistent with standard TFHE parameter choices. Under this assumption, $\mathbb{Z}_q$ is partitioned into $2N_R$ contiguous regions: $I_0 = \left[0, \frac{(q-1)}{2N_R}\right]$ and $I_t = \left[\frac{t(q-1)}{2N_R} + 1, \frac{(t+1)(q-1)}{2N_R}\right]$ for $t = 1, \ldots, 2N_R - 1$. The index t of the region containing α determines β after modulus switching (specifically, $\beta = t$). We therefore use a table lookup to identify the region of α and thereby compute β, which implements the modulus switch within our proof system.

Blind Rotation Initialization. Prior to blind rotation, the prover needs to prove the mapping $\mathcal{F}(\alpha) = X^\alpha$. X^α can be represented in an NTT basis:

$$X^\alpha := (\omega^\alpha, \omega^{3\alpha}, \ldots, \omega^{(2N_R-1)\alpha}),$$

where ω is a primitive $2N_R$-th root of unity in $\mathbb{Z}_Q$ and ω^i denotes its i-th power. We first verify ω^α via a lookup table argument showing that (α, ω^α) belongs to the precomputed table $\{(j, \omega^j)\}_{j=0}^{2N_R-1}$. The remaining $N_R - 1$ entries are then

certified using the multiplicative recurrence

$$\omega^{i\alpha} \;=\; \omega^{(i-2)\alpha} \cdot (\omega^{\alpha})^2 \qquad \text{for } i \in \{\,3, 5, \ldots, 2N_R - 1\,\}.$$

Because multiplicative constraints are cheaper than additional table lookups, this design reduces the prover's cost.

Blind Rotation. Blind rotation mainly includes four core operations: gadget decomposition, NTT, Hadamard product, and inverse NTT. Gadget decomposition is certified via a lookup table argument that enforces the value ranges of the decomposed components. Both the NTT and the inverse NTT are verified with a vector NTT check, while the Hadamard product is validated with a vector Hadamard product check.

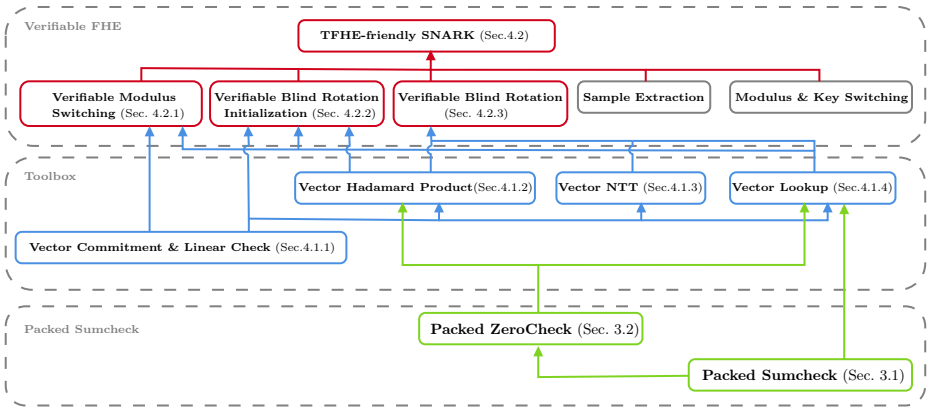

Fig. 1: The Workflow of Our TFHE-friendly SNARK

1.3 Related Work

Sumcheck over Small Fields A growing line of work has focused on efficient sumcheck over small fields. Early theoretical progress was achieved by Ron-Zewi et al. [48] and Holmgren et al. [39], who developed multi-sumcheck protocols using tensor codes and code-switching. Gruen [37] accelerates proving over extension fields by univariate skip. Most recently, Bagad, Domb and Thaler [5] improve prover efficiency by replacing a portion of extension field operations with a larger number of base field computations. In contrast, we take a different route to obtain round-by-round soundness by combining folding and repetition.

A separate line of work studies specialized sumcheck instances [22,21,4] in which one multilinear polynomial must be the multilinear extension of the equality function. Since our setting does not impose this restriction, we do not compare with this line of work.

Concurrent work Recently, a *concurrent work* by Liu et al. [43] offered a helpful refinement by reducing the number of extension field multiplications through optimized challenge design, thereby improving prover efficiency in settings where the extension factor is a power of two. Their approach still relies on the field extension to guarantee round-by-round soundness. And we take a fundamentally different route: by combining repetition with folding, we eliminate extension field multiplications entirely while supporting arbitrary extension factors.

Verifiable FHE Recent advances in SNARKs [54,28,49,18,8,7,9,35,55,56,13] have established them as a central tool for constructing verifiable FHE. Several works have explored this direction. Fiore et al. [27] and Bois et al. [10] used commit-and-prove SNARKs to prove basic FHE ciphertext operations. Viand et al. [52] benchmarked mainstream SNARKs for proving the BGV scheme [12]. Other works have proposed specialized proof systems adapted to the algebraic structure of FHE. Atapoor et al. [2] employed double-CRT arithmetic and a lattice-based SNARK, while Ganesh et al. [29] used Rinocchio, a SNARK designed for ring arithmetic. Both of these constructions are based on the Linear PCP model and therefore require a trusted setup. In a different direction, Cascudo et al. [16] proposed modifying the FHE ciphertext structure to enable decomposition into SNARK-friendly Galois fields. In contrast to this SNARK-based line of work, Zhou et al. [58] demonstrated proofs of BGV [12] and CKKS [17] using VOLE-based zero knowledge protocols. All of these approaches do not address the problem of verifiable bootstrapping. As such, they are largely orthogonal to the contributions of this work.

The problem of verifying TFHE bootstrapping [20] has only recently been studied. Thibault and Walter [51] presented a complete proof for TFHE bootstrapping using the Plonky2, instantiating both the FHE and SNARK components over the 64-bit Goldilocks field. Their construction relies on IVC to recursively prove each CMUX gate in the blind rotation. More recently, Liu et al. proposed Hasteboots [42], which constructs a proof system for bit-wise bootstrapping using the Brakedown polynomial commitment scheme together with the sumcheck protocol. Their scheme operates over the Babybear field and requires a 4-fold field extension to achieve soundness.

Our work differs from these approaches in two fundamental respects. First, we introduce the packed sumcheck protocol, which allows all computations to be performed directly in the base field. Second, we simplify the structure of TFHE bootstrapping through appropriate parameter selection, and transform the proof of bootstrapping to the proof of vector relations.

2 Preliminaries

We denote a finite field by $\mathbb{F}$, the security parameter by λ, and a negligible function in λ by $\mathsf{negl}(\lambda)$. We use PPT to denote probabilistic polynomial time. For any field $\mathbb{F}$ and integer n, we write $\mathbb{F}[n]$ to denote the set of polynomials in n variables with coefficients in $\mathbb{F}$. For any vector $\mathbf{a}$, we use $\mathbf{a}[i]$ to denote its i-th

entry. We use $\circ$ to denote Hadamard product. Due to space constraints, several preliminaries are deferred to Appendix A.

2.1 Round-by-Round Soundness

Definition 1 (Round-by-Round Soundness). *[14] Let $\Pi = (P, V)$ be a $2r$-message public coin interactive proof system for a language L. For any $x \in \{0,1\}^*$, and for any prefix τ of a protocol transcript, let $V(x, \tau)$ denote the distribution of the next message (or output) of V when the transcript so far is τ and V was executed on input x.*

We say that Π has **round-by-round soundness** *error $\epsilon(\cdot)$ if there exist a deterministic (not necessarily efficiently computable) function* State *that takes as input an instance x and a transcript prefix τ and outputs either* accept *or* reject *such that the following properties hold:*

1. *If $x \notin L$, then $\text{State}(x, \emptyset) =$ **reject**, where $\emptyset$ denotes the empty transcript.*
2. *If $\text{State}(x, \tau) = \text{reject}$ for a transcript prefix τ, then for every potential prover message α, it holds that*

$$\Pr_{\beta \leftarrow V(x,\tau|\alpha)} [\text{State}(x, \tau|\alpha|\beta) = \text{accept}] \leq \epsilon(n)$$

3. *For all full transcript prefix τ, if $\text{State}(x, \tau) = \text{reject}$ then $V(x, \tau) = 0$.*

We say that Π is round-by-round sound *if it has round-by-round soundness error ϵ for some $\epsilon(n) = \text{negl}(n)$.*

2.2 Fully Homomorphic Encryption

Definition 2. *A fully homomorphic encryption scheme FHE defined over a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$ is a tuple of PPT algorithms* (FHE.KeyGen,FHE.Enc,FHE.Eval,FHE.Dec) *where the algorithm are defined as follows:*

1. $\text{FHE.KeyGen}(1^\lambda)$*: Upon input the security parameter λ,* FHE.KeyGen *outputs a pair of keys* pk,sk *where* pk *and* sk *are the public key and secret resp.*
2. $\text{FHE.Enc}(pk, x)$*: Upon input the public key* pk *and a message x,* FHE.Enc *outputs a ciphertext $c = \hat{x}$.*
3. $\text{FHE.Eval}(pk, f, c_1, \cdots, c_n)$*: Upon input the public key* pk*. a circuit $f : \mathcal{M}^n \to \mathcal{M}$ and a set of ciphertexts $(c_1, \cdots, c_n)$ that encrypt the message $x_s = (x_1, \cdots, x_n)$,* FHE.Eval *outputs a ciphertext $c = f(\hat{x}_s)$.*
4. $\text{FHE.Dec}(\text{sk}, c)$*: Upon input the secret key* sk *and a ciphertext $c = \hat{x}$,* FHE.Dec *outputs a message x.*

– *Correctness: An FHE scheme is said to be correct if, for all $m \in \mathcal{M}$, $c \in \mathcal{C}$, and f that can be efficiently evaluated, it holds that:*

$$\Pr[\text{FHE.Dec}(sk, c_r) = f(x_1, \cdots, x_n)] \geq 1 - negl(\lambda).$$

where $c_r = \text{FHE.Eval}(pk, f, c_1, \cdots, c_n)$.

– *Security: An FHE scheme is said to be secure if, for all $m \in \mathcal{M}, c \in \mathcal{C}$, and a* PPT *adversary $\mathcal{A}$, c does not leak any information about its underlying message m to $\mathcal{A}$ beyond what is public know.*

Ciphertext Structure The security of TFHE bootstrapping is based on the hardness of the Learning With Errors (LWE) [47] and Ring-LWE (RLWE) [45], and typically involves four ciphertext types: LWE, RLWE, RLWE', and RGSW. Let $q, Q \in \mathbb{Z}_{>0}$, $R = \mathbb{Z}[X]/(X^{N_R}+1)$, and $R_Q = R/QR$. Unless specified otherwise, all encryptions share a fixed secret key (for LWE, $\mathbf{s} \in \mathbb{Z}_q^n$; for RLWE, $s' \in R$).

For some error distribution χ over $\mathbb{Z}_q$ (typically a discrete Gaussian or a bounded uniform distribution) and its ring analogue χ_R over R_Q, an LWE encryption of $m \in \mathbb{Z}_q$ samples $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ uniformly and $e \leftarrow \chi$, and outputs

$$(\mathbf{a}, -\langle \mathbf{a}, \mathbf{s} \rangle + m + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q.$$

An RLWE encryption of $m \in R_Q$ samples $a \leftarrow R_Q$ uniformly and $e \leftarrow \chi_R$, and outputs

$$(a,\ -a \cdot s' + m + e) \in R_Q^2.$$

Let $\mathbf{g} = (g_0, \ldots, g_{D_d-1})$ be a *gadget vector* (e.g., $g_i = B^i$ with $B \geq 2$ and $D_d = \lceil \log_B Q \rceil$) such that every $t \in R_Q$ admits a *gadget decomposition* $t = \sum_{i=0}^{D_d-1} g_i\, t_i$ with small coefficients $t_i \in R$. We adopt the definitions of RLWE' and RGSW from [46]: for $m \in R_Q$ and secret $s' \in R$,

$$\mathsf{RLWE}'_{s'}(m) := \big(\, \mathsf{RLWE}_{s'}(g_0 m),\ \mathsf{RLWE}_{s'}(g_1 m),\ \ldots,\ \mathsf{RLWE}_{s'}(g'_{D_d-1} m) \,\big) \in R_Q^{2D_d},$$

$$\mathsf{RGSW}_{s'}(m) := \big(\, \mathsf{RLWE}'_{s'}(sm),\ \mathsf{RLWE}'_{s'}(m) \,\big) \in R_Q^{2\times 2D_d},$$

where each $\mathsf{RLWE}_{s'}(\cdot)$ component is instantiated with fresh, independent randomness and noise.

2.3 TFHE Bootstrapping

TFHE bootstrapping takes as input an LWE ciphertext $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e + \ m)$ and a function f. It returns a refreshed LWE ciphertext under the same key and modulus whose message is transformed to $f(m)$ and whose noise e' satisfies $|e'| < |e|$. The high-level overview of the bootstrapping framework is as follows:

$$\mathsf{LWE}_{q,\mathbf{s}}(m) \xrightarrow{\mathsf{MS}} \mathsf{LWE}_{2N_R,\mathbf{s}} \xrightarrow{\mathsf{BR}} \mathsf{RLWE}_{Q,s'} \xrightarrow{\mathsf{Ext}} \mathsf{LWE}_{Q,\mathbf{s}'} \xrightarrow{\mathsf{MS+KS}} \mathsf{LWE}_{q,\mathbf{s}}(f(m))$$

Step 1: Modulus Switching The modulus switching step maps an input LWE ciphertext $(\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$ to $(\mathbf{a}', b') \in \mathbb{Z}_{2N_R}^{n+1}$ by scaling and flooring each coefficient:

$$a'_i = \left\lfloor \tfrac{a_i \cdot 2N_R}{q} \right\rceil, \qquad b' = \left\lfloor \tfrac{b \cdot 2N_R}{q} \right\rceil.$$

This aligns the ciphertext with the subgroup $\langle X \rangle \subset R_{2N_R}$ for blind rotation.

Step 2: Blind Rotation We first apply the embedding $\mathcal{F} : \alpha \mapsto X^{\alpha}$ to the modulus-switched ciphertext $(\mathbf{a}, b) \in \mathbb{Z}_{2N_R}^{n+1}$, obtaining $(X^{a_0}, \ldots, X^{a_{n-1}}, X^b)$. Then we initialize $c = (0, X^b \cdot tv(X)) \in R_Q^2$, where $tv(X)$ is a test polynomial For each $i \in [n]$, update $c \leftarrow (X^{a_i} - 1)c \odot C_i + c$, where $C_i = \mathsf{RGSW}_{s'}(s_i)$ and $\odot$ denotes the external product (involving gadget decomposition, NTT, Hadamard multiplication, and inverse NTT). The final ciphertext is $c = \mathsf{RLWE}_{Q,s'}\big(X^{b+\langle \mathbf{a}, \mathbf{s} \rangle} \cdot tv(X)\big)$, whose constant term equals $f(m)$ by design of $tv(X)$.

Step 3: Sample Extraction Given $\mathbf{c} = (a, b) \in R_Q^2$ with $a = \sum_{j=0}^{N_R-1} \alpha_j X^j$ and secret s', define $\mathbf{a}' = (\alpha_0, -\alpha_{N_R-1}, \ldots, -\alpha_1), b' = b_0$. The extracted ciphertext $(\mathbf{a}', b') \in \mathbb{Z}_Q^{N_R+1}$ satisfies $b' - \langle \mathbf{a}', \mathbf{s}' \rangle = (b - a \cdot s')_0$, thus obtaining an $\mathsf{LWE}_{Q,\mathbf{s}'}(f(m))$ encryption.

Step 4: Modulus Switching and Key Switching Finally, the ciphertext under $\mathbf{s}'$ and modulus Q is converted back to the original key $\mathbf{s} \in \{0,1\}^n$ and q. One can refer to the details in [46].

3 The Packed Sumcheck Protocol

In this section, we present the packed sumcheck protocol. The forward difference based polynomial evaluation Alg. 2 is detailed in App. B.1. Sec. 3.1 develops the packed sumcheck protocol in detail, and Sec. 3.2 introduces the packed ZeroCheck protocol built on the packed sumcheck.

3.1 Construction of Packed Sumcheck Protocol

The core idea of our packed sumcheck protocol is to combine repetition with folding so as to obtain round-by-round soundness. Our design is inspired by the "sumfold" technique in NeutronNova [40], but introduces adaptations for the small field setting. Concretely, to fold multiple sumcheck instances in a single interaction round, we interpolate with univariate rather than multilinear polynomials, and we further accelerate the folding step using the forward difference.

Consider a sumcheck instance over a prime field $\mathbb{F}_p$ with security parameter λ and repetition parameter k, where $p^k = O(2^{\lambda})$. The prover needs to prove

$$\sum_{\mathbf{y} \in \{0,1\}^{\log(kN)}} f^{(0)}(\mathbf{y}) \times \cdots \times f^{(d-1)}(\mathbf{y}) = h$$

where each $f^{(s)}$ is a multilinear polynomail, so the total degree of the sumcheck instance is at most d. Assume $p > d(2k-1)$ so that forward differences Alg. 2 apply to the folded polynomial.

Before the protocol starts, the prover partitions the size-kN instance into $2k$ sub-instances, each of size $N/2$.

$$\sum_{\mathbf{x} \in \{0,1\}^l} \prod_{j=0}^{d-1} f_0^{(j)}(\mathbf{x}) = h_0, \cdots, \sum_{\mathbf{x} \in \{0,1\}^l} \prod_{j=0}^{d-1} f_{2k-1}^{(j)}(\mathbf{x}) = h_{2k-1},$$

where $f^{(s)}(\mathbf{y}) = eq(\mathbf{b}, \mathbf{i}) f_i^{(s)}(\mathbf{x})$ for $s \in [0, d-1]$, with $\mathbf{y} = \mathbf{b}||\mathbf{x} \in \{0,1\}^{\log(2k)} \times \{0,1\}^{\log(N/2)}$ and $\mathbf{i}$ denoting the binary representation of i. We assume the prover commits to the $2kd$ polynomials $f_i^{(s)}$ once at the beginning.

The prover folds these $2k$ sub-instances into a univariate polynomial. For a subset $W = \{w_0, \ldots, w_{2k-1}\} \subseteq \mathbb{F}_p$, define Lagrange basis polynomials $L_0, \ldots, L_{2k-1}$ such that $L_i(w_j) = 1$ if $i = j$ and 0 otherwise. Each L_i has degree at most $2k-1$. The prover computes a univariate polynomial $F(r)$ of degree at most $d(2k-1)$ and sends it to the verifier:

$$F(r) = \sum_{\mathbf{x} \in \{0,1\}^{\log(N/2)}} \prod_{s=0}^{d-1} \left(\sum_{i=0}^{2k-1} L_i(r) f_i^{(s)}(\mathbf{x}) \right)$$

Upon receiving F, the verifier checks that $F(w_i) = h_i$ for all $i \in [0, 2k-1]$, and verifies that $\sum_i h_i = h$. If these checks succeed, the verifier samples k random challenges $r_0, \ldots, r_{k-1} \in \mathbb{F}_p$ and sends them to the prover. Upon receiving the challenges, the prover updates the polynomials:

$$f_i'^{(s)}(\mathbf{x}) = \sum_{j=0}^{2k-1} L_j(r_i) f_j^{(s)}(\mathbf{x}), \quad i \in [0, k-1], \ s \in [0, d-1].$$

Next, the prover defines the sliced polynomials $g_i^{(s)}(\mathbf{x}') = f_i'^{(s)}(0, \mathbf{x}'), g_{i+k}^{(s)}(\mathbf{x}') = f_i'^{(s)}(1, \mathbf{x}')$, for all $i \in [0, k-1]$ and $s \in [0, d-1]$, where $\mathbf{x}'$ is obtained from $\mathbf{x}$ by removing its first variable. The prover recommits to the $2kd$ polynomials $\{g_i^{(s)}\}$. To certify that these sliced polynomials are correctly derived from the challenge folds $\{f_i^{(s)}\}$, the verifier requests openings at k verifier-chosen evaluation points for the relevant polynomial oracles ($f_i^{(s)}$ and $g_i^{(s)}$) and checks the equalities for each i and s: $g_i^{(s)}(\mathbf{x}') = f_i'^{(s)}(0, \mathbf{x}'), g_{i+k}^{(s)}(\mathbf{x}') = f_i'^{(s)}(1, \mathbf{x}')$. After this consistency step, the remaining task is to prove $2k$ sumcheck sub-instances of size $N/4$; we therefore recurse by invoking a packed sumcheck of total size $kN/2$, so that the instance size halves in every round. We formally present the packed sumcheck protocol in Fig. 2.

Remark 1. Since the commitment size halves in each round, the total commitment cost in the protocol never exceeds the initial commitment. Moreover, under Brakedown polynomial commitments, no additional commitments are required per round: commitments encode polynomials as row-linear combinations of a fixed coefficient matrix, allowing the verifier to update the combination coefficients in place as a function of the challenges. We discuss this further in App. B.2.

Remark 2. For ease of exposition, we assume that the sumcheck takes the form of a product of d multilinear polynomials. In fact, the packed sumcheck protocol naturally extends to instances of the form $P(\mathbf{x}) = \mathsf{Combine}(f^{(0)}, \ldots, f^{(D)})$, where D is a constant, $\mathsf{Combine}$ denotes an arbitrary composition of additions and multiplications, and the total degree is at most d. In this setting, it suffices to replace each occurrence of $f^{(s)}$ in $\mathsf{Combine}$ with $\sum_{i=0}^{2k-1} L_i(r)\, g_i^{(s)}$, where $g_i^{(s)}$ denotes the multilinear polynomial obtained by decomposing $f^{(s)}$ into $2k$ parts.

Packed Sumcheck Protocol

Consider a sumcheck protocol over the field $\mathbb{F}_p$, where $p^k = O(2^\lambda)$. Suppose the sumcheck instance has size kN:

$$\sum_{\mathbf{y}\in\{0,1\}^{\log(kN)}} f^{(0)}(\mathbf{y}) \cdot \cdots \cdot f^{(d-1)}(\mathbf{y}) = h.$$

Prior to protocol execution, the prover and verifier select a subset $W = \{0, 1, \ldots, 2k-1\} \subset \mathbb{F}_p$. Define $2k$ univariate polynomials $\{L_i(r)\}_{i=0}^{2k-1}$ of degree at most $2k-1$ over $\mathbb{F}_p$ such that $L_i(j) = 1$ if $i = j$ and 0 otherwise.

1. The prover decomposes the sumcheck instance of size kN into $2k$ sub-instances, letting $2^l = N$:

$$\sum_{\mathbf{x}\in\{0,1\}^{l-1}} \prod_{j=0}^{d-1} f_0^{(j)}(\mathbf{x}) = h_0, \cdots, \sum_{\mathbf{x}\in\{0,1\}^{l-1}} \prod_{j=0}^{d-1} f_{2k-1}^{(j)}(\mathbf{x}) = h_{2k-1},$$

   Here, $f^{(s)}(\mathbf{y}) = \sum_{\mathbf{b}\in\{0,1\}^{\log(2k)}} eq(\mathbf{b}, \mathbf{i}) f_i^{(s)}(\mathbf{x})$ for $s \in [0, d-1]$, with $\mathbf{y} = \mathbf{b}||\mathbf{x} \in \{0,1\}^{\log(2k)} \times \{0,1\}^{\log(N/2)}$, $\mathbf{i}$ denoting the binary representation of i and $eq(\mathbf{b}, \mathbf{i}) = \prod_{j=0}^{\log(2k-1)}((1-b_j)(1-i_j) + b_j i_j)$. The prover commits to the $2kd$ polynomials $\{f_i^{(s)}\}$ and sends the values $\{h_i\}$ to the verifier. The verifier then checks that $\sum_i h_i = h$.
2. Prover computes the folded polynomial

$$F(r) = \sum_{\mathbf{x}\in\{0,1\}^{l-1}} \prod_{s=0}^{d-1} \left(\sum_{j=0}^{2k-1} L_j(r) f_j^{(s)}(\mathbf{x}) \right).$$

   Then the prover sends $F(r)$ to the verifier.
3. The verifier checks $F(w_i) = h_i, \forall i \in [0, 2k-1]$. Then the verifier samples random challenges $\{r_i\}_{i=0}^{k-1} \in \mathbb{F}_p$, computes $h'_i = F(r_i)$, and sends $\{r_i\}$ to the prover.
4. For each $s \in [0, d-1]$ and $i \in [0, k-1]$, the prover computes

$$f_i'^{(s)}(\mathbf{x}') = \sum_{j=0}^{2k-1} L_j(r_i) f_j^{(s)}(\mathbf{x}).$$

   the prover computes the sliced polynomials $g_i^{(s)}(\mathbf{x}') = f_i'^{(s)}(0, \mathbf{x}'), g_{i+k}^{(s)}(\mathbf{x}') = f_i'^{(s)}(1, \mathbf{x}')$, for all $i \in \{0, \ldots, k-1\}$ and $s \in \{0, \ldots, d-1\}$, where $\mathbf{x}'$ is obtained from $\mathbf{x}$ by removing its first variable.
5. The prover recommits to the $2kd$ polynomials $\{g_i^{(s)}\}$. And the verifier opens at k random evaluation points for the relevant polynomial oracles ($f_i^{(s)}$ and $g_i^{(s)}$) and checks the equalities for each i and s:

$$g_i^{(s)}(\mathbf{x}') = f_i'^{(s)}(0, \mathbf{x}'), \qquad g_{i+k}^{(s)}(\mathbf{x}') = f_i'^{(s)}(1, \mathbf{x}'),$$

Fig. 2: Packed Sumcheck Protocol

The Packed Sumcheck Protocol (continued)

6. If $kN/2$ is smaller than a constant c, the prover sends the $2kd$ polynomials $\{g_i^{(s)}\}$ to the verifier, who then completes the remaining verification locally. Otherwise, the prover and verifier recursively invoke the packed sumcheck on the polynomials $\{g_i^{(s)}\}$.

- With Brakedown as the polynomial commitment, Step 5 is not required.

Fig. 2: Packed Sumcheck Protocol (continued)

Lemma 1. *Instantiating Protocol 2 with the Brakedown polynomial commitment achieves a round-by-round soundness error of* $\left(\frac{(2k-1)d}{p}\right)^k$. *For other polynomial commitments, the round-by-round soundness error is* $\max\left(\left(\frac{(2k-1)d}{p}\right)^k, \frac{2kd(\log N-2)^k}{p^k}\right)$. *The prover complexity of packed sumcheck is* $O((kd^2+k^2d)N)\cdot \mathsf{mul} + O(k^2d^2N)\cdot \mathsf{add}$.

The proof details of lemma 1 are given in App. B.3.

Remark 3. For polynomial updates, Strassen's algorithm can further reduce complexity, particularly when k is small. By partitioning each length-2^{l-1} vector into k-sized blocks, these combinations reduce to two $k \times k$ matrix multiplications per round. Using Strassen's algorithm (requiring $O(k^{2.807})$ multiplications and additions), the total cost becomes $O\big((kd^2+k^{1.807}d)N\big)\cdot \mathsf{mul} + O(k^2d^2N)\cdot \mathsf{add}$.

3.2 Construction of the ZeroCheck Protocol

The sumcheck protocol is the key building block for ZeroCheck. Our construction extends naturally to a packed ZeroCheck protocol, which preserves the same asymptotic complexity as packed sumcheck. We formally present the packed ZeroCheck protocol in Fig. 3.

Lemma 2. *The packed ZeroCheck in Fig. 3 with Brakedown Commitment achieves a round-by-round soundness error of*

$$\max\left(\frac{k\cdot l^k}{p^k}, \frac{(dk+1)^k}{p^k}, \frac{((2k-1)(d+1))^k}{p^k}\right).$$

Moreover, the prover's computational complexity is bounded by

$$O\big((kd^2+k^2d)N\big)\cdot \mathsf{mul} + O(k^2d^2N)\cdot \mathsf{add}.$$

The proof details of lemma 2 are given in App. B.4.

The Packed ZeroCheck Protocol

Consider a sumcheck protocol over the field $\mathbb{F}_p$ with repetition parameter k chosen so that $p^k = O(2^\lambda)$. Let $N = 2^\ell$ and partition the instance into k sub-instances indexed by $i \in \{0, \ldots, k-1\}$. For every $\mathbf{x} \in \{0,1\}^\ell$, the asserted relations are:

$$\prod_{s=0}^{d-1} f_0^{(s)}(\mathbf{x}) = h_0(\mathbf{x}), \cdots, \prod_{s=0}^{d-1} f_{k-1}^{(s)}(\mathbf{x}) = h_{k-1}(\mathbf{x})$$

1. The verifier samples k random vectors $\{\mathbf{r}_j\}_{j=0}^{k-1} \in \mathbb{F}_p^l$ and sends them to the prover.
2. For each $i, j \in [0, k-1]$, the prover is required to demonstrate

$$\sum_{\mathbf{x}\in\{0,1\}^l} eq(\mathbf{r}_j, \mathbf{x}) \prod_{s=0}^{d-1} f_i^{(s)}(\mathbf{x}) = \sum_{\mathbf{x}\in\{0,1\}^l} eq(\mathbf{r}_j, \mathbf{x}) h_i(\mathbf{x}).$$

   where $eq(\mathbf{y}, \mathbf{x}) = \prod_{j=0}^{l-1} ((1-y_i)(1-x_i) + y_i x_i)$. We focus on the left-hand sumcheck as the prover's proof strategy.
3. The prover computes $h_{i,j} = \sum_{\mathbf{x}\in\{0,1\}^l} eq(\mathbf{r}_j, \mathbf{x}) \prod_{s=0}^{d-1} f_i^{(s)}(\mathbf{x})$ and sends $\{h_{i,j}\}$ to the verifier. The prover then constructs the $(k+1)$-variate polynomial

$$F(\mathbf{s}, r) = \sum_{\mathbf{x}\in\{0,1\}^l} \left(\Big(\sum_{i=0}^{k-1} s_i \cdot eq(\mathbf{r}_i, \mathbf{x}) \Big) \prod_{s=0}^{d-1} \Big(\sum_{j=0}^{k-1} L_j(r) f_j^{(s)}(\mathbf{x}) \Big) \right),$$

   where $\mathbf{s} = (s_0, \ldots, s_{k-1})$, and sends $F(\mathbf{s}, r)$ to the verifier.
4. Let $\mathbf{s}^{(j)}$ denote the k-dimensional standard basis vector with a 1 in position j and 0 elsewhere. The verifier first checks that $F(\mathbf{s}^{(j)}, w_i) = h_{i,j}$ for all $i, j \in [0, k-1]$. It then samples random challenges $\{(\mathbf{a}_i, b_i)\}_{i=0}^{k-1}$, computes $h'_i = F(\mathbf{a}_i, b_i)$, and sends the results to the prover.
5. The prover updates the polynomials for all $i \in [0, k-1]$ and $j \in [0, d-1]$:

$$f_i'^{(j)}(\mathbf{x}) = \sum_{e=0}^{k-1} L_e(b_i) f_e^{(j)}(\mathbf{x}), \quad E_i(\mathbf{x}) = \sum_{e=0}^{k-1} \mathbf{a}_i[e] \cdot eq(\mathbf{r}_e, \mathbf{x}).$$

   The prover then uses the packed sumcheck protocol to prove

$$\sum_{\mathbf{x}\in\{0,1\}^l} E_i(\mathbf{x}) \prod_{s=0}^{d-1} f_i'^{(s)}(\mathbf{x}) = h'_i, \quad \forall i \in [0, k-1].$$

Fig. 3: The Packed ZeroCheck Protocol

4 TFHE-friendly SNARK

In this section, we present a TFHE-friendly SNARK construction. In Sec. 4.1, we introduce a vector commitment scheme for linear checks, and we further present vector relation checks including Hadamard product checks, NTT checks, and lookup checks. In Sec. 4.2, we show how to use these tools to prove modulus switching, blind rotation initialization and blind rotation in TFHE bootstrapping. In Sec. 4.3, we put all of this together to construct our TFHE-friendly SNARK.

4.1 Vector Relation Proofs

In this subsection, we introduce vector relation proofs, which serve as the fundamental building blocks of our TFHE-friendly SNARK construction. All vectors are defined over the finite field $\mathbb{F}_p$, where the field size satisfies $p^k = O(2^\lambda)$, with λ denoting the security parameter.

Vector Commitment for Linear Check Our scheme can be viewed as a variant of the Brakedown [35] and Binius [23] protocols, since these polynomial commitment schemes are fundamentally designed to check linear combinations of vectors. Specifically, we employ a block Reed–Solomon (RS) encoding scheme.

We adopt a Reed–Solomon encoding method that incorporates block-level encoding [23]. The encoding procedure is summarized in Alg. 1.

Algorithm 1: Block RS code Enc Algorithm: $\mathbb{F}_p^n \to \mathbb{F}_p^{\gamma n}$

Input: $\mathbf{x} \in \mathbb{F}_p^n$
Output: $\mathbf{w}' \in \mathrm{GF}(p,k)^{\gamma n/k}$

1. Partition the input vector into groups of k elements. From each group, extract the elements at positions $0, \cdots, k-1$ to form vectors $\mathbf{x}_0, \cdots, \mathbf{x}_{k-1} \in \mathbb{F}_p^{n/k}$;
2. Select an NTT basis W of size $\gamma n/k$ over $\mathbb{F}_p$, and apply Reed–Solomon encoding with respect to W to each vector $\mathbf{x}_i$, obtaining the encoded vectors $\mathbf{w}_i \in \mathbb{F}_p^{\gamma n/k}$;
3. Recombine these encoded vectors into a single vector $\mathbf{w}$ of length γn, defined by $\mathbf{w}[k \cdot i + j] = \mathbf{w}_i[j]$;
4. By packing every k consecutive entries of $\mathbf{w}$ into a single element over $\mathrm{GF}(p,k)$, we obtain the vector $\mathbf{w}' \in \mathrm{GF}(p,k)^{\gamma n/k}$, which is returned as the final codeword;

Lemma 3. *Over the field $GF(p,k)$, the encoding in Alg. 1 achieves relative distance $1 - \frac{1}{\gamma}$.*

Proof. The encoding method in Alg. 1 is inspired by the concept of block-level encoding. The essential idea is to treat every k consecutive elements over the base field $\mathbb{F}_p$ as a single element over the degree-k extension field $\mathrm{GF}(p,k)$.

Accordingly, a vector $\mathbf{x} \in \mathbb{F}_p^n$ is reinterpreted as $\hat{\mathbf{x}} \in \mathrm{GF}(p,k)^{n/k}$. The vector $\hat{\mathbf{x}}$ is then viewed as the coefficient representation of a polynomial, which is evaluated on the NTT basis W. Since W is defined over $\mathbb{F}_p$, the polynomial evaluation involves only multiplications between $\mathbb{F}_p$ and $\mathrm{GF}(p,k)$. Each such multiplication can be regarded as k independent multiplications over $\mathbb{F}_p$, which justifies the decomposition used in Alg. 1. After encoding, $\hat{\mathbf{x}}$ achieves a relative code distance of $1 - \frac{1}{\gamma}$. □

The vector commitment and verification IOPs are defined as follows:
Commit Phase. $P(\mathbf{a}, n_c) \to c$, where $\mathbf{a} \in \mathbb{F}_p^{n_c}$ and the parameter n_c specifies the vector length:

1. The prover encodes the vector $\mathbf{a}$ using the block-RS code in the Alg.1, obtaining a codeword $\tilde{\mathbf{a}} \in \mathrm{GF}(p,k)^{\gamma n_c/k}$.
2. The prover builds a Merkle tree over the entries of $\tilde{\mathbf{a}}$ and takes its root c as the commitment to $\mathbf{a}$.
3. The prover sends the commitment c to the verifier.

Code Test IOP. $(P(\{\mathbf{a}_i\}_{i\in[m-1]}, n_c), V(\{c_i\}_{i\in[m-1]}, n_c)) \to \{0,1\}$

1. The verifier samples a random matrix $R \in \mathbb{F}_p^{k\times m}$.
2. The prover interprets each $\mathbf{a}_i$ as a row of a matrix $U \in \mathrm{GF}(p,k)^{m\times(n_c/k)}$ by grouping every k consecutive elements. The prover computes $V = R \cdot U \in \mathrm{GF}(p,k)^{k\times(n_c/k)}$ and sends V to the verifier.
3. The verifier selects a random query set Q of size $\Theta(\lambda)$. For each column index $j \in Q$, the verifier queries the j-th coordinate of each $\tilde{\mathbf{a}}_i$, receives the values together with their Merkle proofs, and checks the validity of the proofs. For each row V_i of V, the verifier further verifies that $\mathsf{Enc}(V_i)[j] = \sum_{s=0}^{m-1} R_i[s] \cdot \tilde{\mathbf{a}}_s[j]$, where Enc denotes the block-RS encoding algorithm.

Linear Evaluation IOP. $(P(\{\mathbf{a}_i\}_{i\in[m-1]}, n_c), V(\{c_i\}_{i\in[m-1]}, \{r_i\}_{i\in[m-1]}, n_c)) \to \{0,1\}$ This protocol enables the verifier to check a linear combination $\mathbf{v} = \sum_i r_i \cdot \mathbf{a}_i$ for verifier-chosen challenges r_i:

1. The verifier sends $(r_0, \ldots, r_{m-1})$.
2. The prover computes $\mathbf{v}$ and commits to it.
3. The verifier queries $\Theta(\lambda)$ random columns from all $\tilde{\mathbf{a}}_i$ and from $\mathbf{v}$, receives the values and their Merkle proofs, and checks that the proofs are valid. The verifier then checks $\mathsf{Enc}(\mathbf{v})[j] = \sum_{s=0}^{m-1} r_s \cdot \tilde{\mathbf{a}}_s[j]$.

Lemma 4. *Assume that the Block-RS code has relative distance β. If the prover can pass the code test IOP with probability exceeding $\frac{\beta/3+1}{p^k} + (1-\frac{\beta}{3})^{\Theta(\lambda)}$, and can also pass the linear evaluation IOP with probability $(1-\frac{2\beta}{3})^{\Theta(\lambda)}$, then the prover necessarily returns the correct vector $\mathbf{v} = \sum_i r_i \cdot \mathbf{a}_i$.*

Proof. The proof follows the one in [53] showing the binding property of polynomial commitment schemes. Lemma 4 can be regarded as a special-case formulation of this proof over finite fields. □

Remark 4. It is straightforward to derive the Brakedown polynomial commitment scheme from the above vector commitment. Consider a multilinear polynomial in l variables, with coefficient size $N = 2^l$. We reshape the coefficient vector into an $\sqrt{N} \times \sqrt{N}$ matrix and apply the vector commitment scheme row by row. Instantiating the parameters as $n_c = \sqrt{N}$ and $m = \sqrt{N}$ within our framework, and employing block-RS encoding, we obtain the following complexities: the prover runs in time $O(N \log N)$, the verifier runs in time $O(\sqrt{N} \log N)$, and the proof size is $O(\sqrt{N} \log N)$.

Vector Hadamard Product Check We next consider the Hadamard product check over $\mathbb{F}_p$ for the relation $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$, where $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{F}_p^M$ and $M = m \cdot N_c$.

Let $\mathbf{a}, \mathbf{b}, \mathbf{c}$ be represented as $m \times N_c$ matrices A, B, C. The prover invokes the vector commitment algorithm with the vector length parameter explicitly set to $n_c = N_c$, so that the commitments to $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are naturally interpreted as commitments to the individual rows of A, B, C, respectively.

Definition 3 (Vector Hadamard Product Relation over $\mathbb{F}_p$). *The relation $\mathcal{R}_{VHPR}$ is the set of all tuples $(\mathbb{x}; \mathbb{w}) = ((Com(\mathbf{a}), Com(\mathbf{b}), Com(\mathbf{c})); (\mathbf{a}, \mathbf{b}, \mathbf{c}))$ such that $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$.*

Vector Hadamard Product IOP Construction.

1. The prover represents the matrices A, B, C as multilinear polynomials $\tilde{A}(\mathbf{x})$, $\tilde{B}(\mathbf{x}), \tilde{C}(\mathbf{x})$ over $\log M$ variables.
2. The parties execute a packed ZeroCheck protocol to prove the identity $\tilde{A}(\mathbf{x}) \cdot \tilde{B}(\mathbf{x}) = \tilde{C}(\mathbf{x})$ for all $\mathbf{x} \in \{0,1\}^{\log M}$, running for $\log m$ rounds.
3. After $\log m$ rounds, the prover sends the resulting polynomials from the partial sumcheck to the verifier. The parties then invoke the vector commitment linear evaluation IOP to verify these polynomials, after which the verifier locally completes the remaining $\log N_c$ rounds of the sumcheck.

Lemma 5. *Assuming that the ZeroCheck protocol has soundness error ϵ_{ZERO}, the Vector Hadamard Product IOP is perfectly complete and inherits the same soundness error ϵ_{ZERO}.*

Proof. The completeness and soundness of the Vector Hadamard Product IOP follow directly from the guarantees of the ZeroCheck protocol. □

Vector NTT Check We now turn to the Number Theoretic Transform (NTT) check over $\mathbb{F}_p$. In our TFHE-friendly SNARK, the relevant polynomials have degree $N_R - 1$. Hence, we partition vectors into blocks of size N_R. Let $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^M$ with $M = N_R \cdot m$. Partition $\mathbf{a}$ and $\mathbf{b}$ into m blocks $\mathbf{a}_i, \mathbf{b}_i$ for $i \in [0, m-1]$. The prover's goal is to prove that for all i, the relation $\mathbf{a}_i = H \cdot \mathbf{b}_i$ holds, where H is the $N_R \times N_R$ NTT matrix.

The prover invokes the vector commitment scheme with the vector length parameter set to $n_c = N_R$, and commits to all sub-vectors $\mathbf{a}_i$ and $\mathbf{b}_i$. Because the NTT relation is linear, random linear combinations can be used to fold the m individual NTT checks into a single aggregated check.

Definition 4 (Vector NTT Relation over $\mathbb{F}_p$). *The relation $\mathcal{R}_{VNR}$ is the set of all tuples $(\mathbb{x}, \mathbb{w}) = ((Com(\mathbf{a}), Com(\mathbf{b})); (\mathbf{a}, \mathbf{b}))$ such that $\mathbf{a}_i = H\mathbf{b}_i$ where $\mathbf{a}, \mathbf{b} \in \mathbb{F}_p^M$ with $M = N_R \cdot m$, and both vectors are partitioned into m blocks $\mathbf{a}_i, \mathbf{b}_i$ for $i \in [0, m-1]$ and $H \in \mathbb{F}_p^{N_R \times N_R}$ denotes the NTT matrix.*

Vector NTT IOP Construction.

1. The verifier samples k random challenges $\mathbf{r}_0, \cdots, \mathbf{r}_{k-1} \in \mathbb{F}_p^m$ and sends them to the prover.
2. For each $s \in \{0, \cdots, k-1\}$, the prover computes the folded vectors:

$$\tilde{\mathbf{a}}_s = \sum_{i=0}^{m-1} \mathbf{r}_s[i] \cdot \mathbf{a}_i, \quad \tilde{\mathbf{b}}_s = \sum_{i=0}^{m-1} \mathbf{r}_s[i] \cdot \mathbf{b}_i.$$

   The prover sends commitments to $\{\tilde{\mathbf{a}}_s\}_{s\in[0,k-1]}$ and $\{\tilde{\mathbf{b}}_s\}_{s\in[0,k-1]}$ to the verifier.
3. The parties invoke the vector linear evaluation IOP to verify the correctness of these linear combinations.
4. The verifier locally checks that $\tilde{\mathbf{a}}_s = H \cdot \tilde{\mathbf{b}}_s$ holds for each $s \in \{0, \cdots, k-1\}$.

Remark 5. In the Vector NTT IOP, the verifier's computational complexity is $O(m + N_R \log N_R)$. When $m = O(N_R)$ (as is the case in Verifiable FHE; see Sec. 4.2 for details), the verifier's complexity remains sublinear in the vector length M, thereby preserving succinctness.

Lemma 6. *The Vector NTT IOP achieves perfect completeness and has soundness error $\frac{1}{p^k}$.*

The proof details of lemma 6 are given in App. C.1.

Vector Lookup Check Finally, we describe the vector lookup argument over $\mathbb{F}_p$. For a vector $\mathbf{a} \in \mathbb{F}_p^N$ and a public table $\mathbf{t} \in \mathbb{F}_p^M$, the goal is to prove that every element of $\mathbf{a}$ is contained in $\mathbf{t}$. Let $N = n \cdot N_c$ and $M = m \cdot N_c$. The prover partitions the vectors $\mathbf{a}$ and $\mathbf{t}$ into blocks of length N_c and commits to each block using the vector commitment scheme. Our construction is based on the cq lookup protocol [26].

Definition 5 (Vector Lookup Relation over $\mathbb{F}_p$). *The relation $\mathcal{R}_{VLTR}$ is set of all tuples $(\mathbb{x}; \mathbb{w}) = ((Com(\mathbf{a}), \mathbf{t}); (\mathbf{a}, \mathbf{h}))$, where $\mathbf{a} \in \mathbb{F}_p^N$, $\mathbf{t} \in \mathbb{F}_p^M$, and $\mathbf{h} \in \mathbb{F}_p^M$ is a multiplicity vector such that every element of $\mathbf{a}$ appears in the table $\mathbf{t}$.*

The protocol builds on the following lemma, which reduces the lookup claim to a rational function identity.

Lemma 7 ([26]). *All elements of $\mathbf{a} \in \mathbb{F}_p^N$ belong to the table $\mathbf{t} \in \mathbb{F}_p^M$ if and only if there exists a multiplicity vector $\mathbf{h} \in \mathbb{F}_p^M$ satisfying:*

$$\sum_{j=0}^{N-1} \frac{1}{X + \mathbf{a}[j]} = \sum_{e=0}^{M-1} \frac{\mathbf{h}[e]}{X + \mathbf{t}[e]}.$$

Vector Lookup IOP Construction.

1. The verifier samples s random challenges $r_0, \cdots, r_{s-1} \in \mathbb{F}_p$ and sends them to the prover.
2. For each $i \in [0, s-1]$, the prover defines two new vectors:
   - $\mathbf{f}_i \in \mathbb{F}_p^N$, where $\mathbf{f}_i[j] = \frac{1}{r_i + \mathbf{a}[j]}$ for $j \in [0, N-1]$.
   - $\mathbf{g}_i \in \mathbb{F}_p^M$, where $\mathbf{g}_i[e] = \frac{\mathbf{h}_i[e]}{r_i + \mathbf{t}[e]}$ for $e \in [0, M-1]$.

   Using the vector length parameter N_c, the prover partitions and commits to $\{\mathbf{f}_i\}_{i\in[0,s-1]}$ and $\{\mathbf{g}_i\}_{i\in[0,s-1]}$.
3. The parties invoke a packed sumcheck protocol to prove that for each $i \in [0, s-1]$:
$$\sum_{j=0}^{N-1} \mathbf{f}_i[j] = \sum_{e=0}^{M-1} \mathbf{g}_i[e].$$
4. The parties invoke a packed ZeroCheck protocol to prove the following for each $i \in [0, s-1]$:
   - $\mathbf{f}_i[j] \cdot (r_i + \mathbf{a}[j]) = 1$ for all $j \in [0, N-1]$.
   - $\mathbf{g}_i[e] \cdot (r_i + \mathbf{t}[e]) = \mathbf{h}[e]$ for all $e \in [0, M-1]$.

Lemma 8. *Assume that the sumcheck protocol has soundness error ϵ_{SUM} and the ZeroCheck protocol has soundness error ϵ_{ZERO}. Then the Vector Lookup IOP achieves perfect completeness and has soundness error at most*

$$s\epsilon_{\mathsf{SUM}} + 2s\epsilon_{\mathsf{ZERO}} + \frac{(N+M-1)^s}{p^s}.$$

The proof details of Lemma 8 are given in Appendix C.2. We also construct a lookup for l tuples. The construction is similar to the vector lookup IOP, and we defer the details to Appendix C.2.

4.2 Verifiable Computational Steps in Bootstrapping

In our TFHE-friendly SNARK, the prover commits to a sequence of vectors produced along the bootstrapping pipeline. We assume the prover and verifier agree in advance on the exact order in which these vectors appear. Upon receiving a commitment to a given vector, the verifier automatically associates it with its designated role. Because the bootstrapping procedure is public, this assumption is natural and significantly simplifies the proof system.

By carefully selecting parameters, we can align the secret key length and modulus of LWE with those of RLWE, thereby avoiding the need for subsequent modulus and key switching in Sec. 2.3. The detailed parameter design is deferred to Sec. 5.2. And since sample extraction is essentially free, we delegate it to the verifier as in [51]

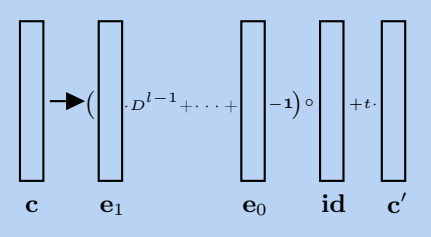

Fig. 4: Modulus Switching

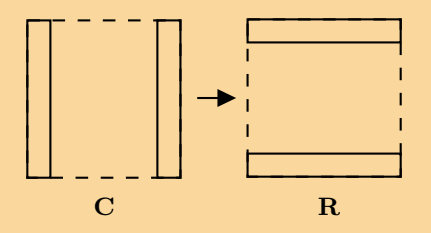

Fig. 5: Blind Rotation Initialization

Modulus Switching The modulus switching operation maps $\mathbf{c} = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$ to $\mathbf{c}' = (\mathbf{a}', b') \in \mathbb{Z}_{2N_R}^{n+1}$. To prove this relation, we rely on the following lemma:

Lemma 9. *Suppose* $2N_R \mid (q-1)$ *and let* $q = t \cdot (2N_R) + 1$. *For any* $\alpha \in \mathbb{Z}_q \setminus \{0\}$, *there exists a unique decomposition* $\alpha = \beta \cdot (2N_R) + \gamma$, *where* $\beta \in [0, 2N_R - 1]$ *and* $\gamma \in [1, t]$, *such that*

$$\left\lfloor \frac{\alpha \cdot (2N_R)}{q} \right\rceil = \beta.$$

The proof details of Lemma 9 are given in App. D.1.

Lemma 10. *Let* $\mathbf{id}, \mathbf{r}, \mathbf{c}' \in \mathbb{Z}_q^{n+1}$ *such that every entry of* $\mathbf{id}$ *lies in* $\{0, 1\}$, *every entry of* $\mathbf{r}$ *lies in* $[1, t]$, *and every entry of* $\mathbf{c}'$ *lies in* $[0, 2N_R - 1]$. *If*

$$\mathbf{c} = (t \cdot \mathbf{c}' + \mathbf{r}) \circ \mathbf{id},$$

then $\mathbf{c}'$ *is exactly the result of applying modulus switching to* $\mathbf{c}$.

The proof details of lemma 10 are given in App. D.1.

To facilitate proving that $\mathbf{r} \in [1, t]^{n+1}$, it suffices to show that $\mathbf{r} - \mathbf{1} \in [0, t-1]^{n+1}$. To further reduce table size, we decompose $\mathbf{r} - \mathbf{1}$ into l blocks under a D-ary expansion. We present the proof framework in Fig. 4, and the protocol details in Fig. 6.

Blind Rotation Initialization Let $w_1, \ldots, w_{2N_R-1}$ denote the odd powers of a primitive $2N_R$-th root of unity in $\mathbb{Z}_Q$, forming an NTT basis of $\mathbb{Z}_Q[X]/(X^{N_R}+1)$. After modulus switching, the entries of $\mathbf{c}'$ are placed at these exponent indices, yielding an $(n+1) \times N_R$ matrix $\mathbf{C}$ whose rows encode $X^{a_i'}$ or $X^{b'}$ in the NTT basis. In verifiable TFHE bootstrapping, the prover commits to $\mathbf{C}$ column-wise, denoting the i-th column as $\mathbf{C}_{\cdot,i}$. First, by a lookup against the table $(\alpha, w_1^\alpha)_{\alpha \in [0, 2N_R-1]}$, the verifier checks correctness of the first column $\mathbf{C}_{\cdot,0}$. For subsequent columns, observe that $(\mathbf{C}_{\cdot,0} \circ \mathbf{C}_{\cdot,0}) \circ \mathbf{C}_{\cdot,i} = \mathbf{C}_{\cdot,i+1}$, so the remaining columns can be verified using vector Hadamard product relations IOP. This certifies the validity of the entire matrix $\mathbf{C}$.

Since $\mathbf{C}$ is committed column-wise but each row corresponds to a polynomial ring element, we also construct a row-wise commitment matrix R. To check consistency between $\mathbf{C}$ and $\mathbf{R}$, the verifier samples random matrices $\mathbf{S}_0 \in \mathbb{Z}_p^{(n+1) \times k}$

The Verifiable Modulus Switch

We define a vector $\mathbf{id} \in \mathbb{Z}_q^{n+1}$ such that $\mathbf{id}[i] = 0$ if $\mathbf{c}[i] = 0$ and $\mathbf{id}[i] = 1$ otherwise. The prover additionally commits to $\mathbf{id}$. Moreover, the prover sets the vector length parameter to $n_c = n + 1$ when committing to $\mathbf{c}, \mathbf{c}'$, and $\{\mathbf{e}_i\}_{i\in[0,l-1]}$.

- Using the vector Hadamard product IOP, the prover demonstrates that

$$\mathbf{c} = \left(\mathbf{e}_{l-1} \cdot D^{l-1} + \cdots + \mathbf{e}_0 + \mathbf{1} + \mathbf{c}' \cdot t\right) \circ \mathbf{id}.$$

- Using the vector lookup IOP, the prover proves that all entries of $\mathbf{id}$ lie in $\{0, 1\}$, each element of $\{\mathbf{e}_j\}_{j\in[0,l-2]}$ lies in $[0, D-1]$, each element of $\mathbf{e}_{l-1}$ lies in $[0, \lfloor \frac{t-1}{D^{l-1}} \rfloor]$, and each element of $\mathbf{c}'$ lies in $[0, 2N_R - 1]$.

Fig. 6: The Verifiable Modulus Switch

and $\mathbf{S}_1 \in \mathbb{Z}_Q^{N_R\times k}$, requests the prover to compute $\mathbf{T}_0 = \mathbf{S}_0^\top C, \mathbf{T}_1 = R\,\mathbf{S}_1$, and verifies $\mathbf{T}_0^\top \mathbf{S}_1 = \mathbf{T}_1^\top \mathbf{S}_0$. We present the proof framework in Fig. 5, and the protocol details in Fig. 7.

The Verifiable Blind Rotation Initialization

The prover commits to the columns of matrix $\mathbf{C}$ with parameter $n_c = n + 1$, and to the rows of matrix $\mathbf{R}$ with parameter $n_c = N$.

- By performing a vector lookup IOP against the table $\{(\alpha, w_1^\alpha)\}_{\alpha\in[0,2N_R-1]}$, the verifier checks the correctness of the first column $\mathbf{C}_{\cdot,0}$. The remaining columns are then verified using the vector Hadamard product IOP by proving $(\mathbf{C}_{\cdot,0} \circ \mathbf{C}_{\cdot,0}) \circ \mathbf{C}_{\cdot,i} = \mathbf{C}_{\cdot,i+1}$.
- To enforce consistency between $\mathbf{C}$ and $\mathbf{R}$, the verifier samples random matrices/vectors $\mathbf{S}_0 \in \mathbb{Z}_p^{(n+1)\times k}$ and $\mathbf{S}_1 \in \mathbb{Z}_p^{N_R\times k}$, asks the prover to compute

$$\mathbf{T}_0 = \mathbf{S}_0^\top \mathbf{C}, \qquad \mathbf{T}_1 = \mathbf{R}\,\mathbf{S}_1,$$

and verifies that $\mathbf{T}_0^\top \mathbf{S}_1 = \mathbf{T}_1^\top \mathbf{S}_0$.

Fig. 7: The Verifiable Blind Rotation Initialization

Blind Rotation Blind rotation constitutes the primary computational cost in TFHE bootstrapping and requires n highly repetitive iterations (Fig. 8). Each vector corresponds to an element of $\mathbb{Z}_p[X]/(X^{N_R}+1)$. In our verifiable protocol, the prover aggregates ciphertexts from the same position across iterations into length-nN vectors and proves the corresponding relations in a unified manner. By contrast, the approach of Thibault and Walter [51] recursively proves each iteration, which is less efficient. Leveraging the repetitive structure enables unified proofs and achieves significant performance improvements.

In Fig. 8, the index i ranges over $\{1, \ldots, n\}$. During the initialization phase of blind rotation, the RLWE ciphertext is represented in interpolation form. Before Round 1, it is denoted $(\mathbf{ai}_0, \mathbf{bi}_0) \in \mathbb{Z}_Q^{2N_R}$. After an inverse NTT (INTT), the protocol enters an iterative loop of n CMUX gates, yielding $(\mathbf{ai}_n, \mathbf{bi}_n) \in \mathbb{Z}_Q^{2N_R}$.

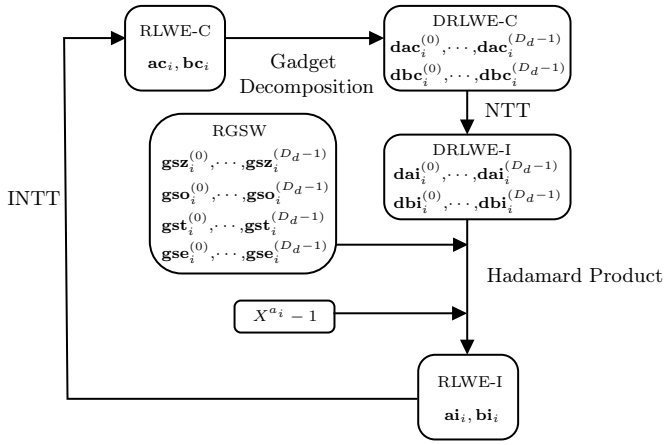

Fig. 8: Round-i CMUX Operation

Verifiable n CMUX Gates

The prover first commits to the relevant vectors using the vector length parameter $n_c = N_R$. For clarity, subscripts are omitted to indicate aggregated vectors. Specifically, we define the concatenated vectors $\mathbf{ai}' := \mathbf{ai}_0||\cdots||\mathbf{ai}_{n-1}$, $\mathbf{bi}' := \mathbf{bi}_0||\cdots||\mathbf{bi}_{n-1}$, $\mathbf{ai} := \mathbf{ai}_1||\cdots||\mathbf{ai}_n$, $\mathbf{bi} := \mathbf{bi}_1||\cdots||\mathbf{bi}_n$.

- **Gadget decomposition via vector lookup.**

$$\mathbf{ac} = \sum_{s\in[0,D_d-1]} \mathbf{dac}^{(s)} \cdot B^s, \quad \mathbf{bc} = \sum_{s\in[0,D_d-1]} \mathbf{dbc}^{(s)} \cdot B^s.$$

  The prover then uses vector lookup IOP to prove that entries of $\{\mathbf{dac}^{(i)}, \mathbf{dbc}^{(i)}\}_{i\in[0,D_d-2]}$ lie in $[0, B-1]$, while entries of $\mathbf{dac}^{(D_d-1)}, \mathbf{dbc}^{(D_d-1)}$ lie in $\left[0, \lfloor p/B^{(D_d-1)} \rfloor\right]$.
- **Vector NTT checks.** The prover proves that the $2D_d$ relevant pairs $\{(\mathbf{dac}^{(i)}, \mathbf{dai}^{(i)}), (\mathbf{dbc}^{(i)}, \mathbf{dbi}^{(i)})\}_{i\in[0,D_d-1]})$ satisfy the NTT relation.
- **Vector Hadamard product checks.** We define two auxiliary vectors:

$$\mathbf{at} = \sum_{s\in[0,D_d-1]} \left(\mathbf{gsz}^{(s)} \circ \mathbf{dai}^{(s)}\right) + \sum_{s\in[0,D_d-1]} \left(\mathbf{gst}^{(s)} \circ \mathbf{dbi}^{(s)}\right),$$
$$\mathbf{bt} = \sum_{s\in[0,D_d-1]} \left(\mathbf{gso}^{(s)} \circ \mathbf{dai}^{(s)}\right) + \sum_{s\in[0,D_d-1]} \left(\mathbf{gse}^{(s)} \circ \mathbf{dbi}^{(s)}\right).$$

  The prover prove the correctness of $\mathbf{at}$ and $\mathbf{bt}$ and then prove:

$$\mathbf{ai} = (X^{\mathbf{a}} - 1) \circ \mathbf{at} + \mathbf{ai}', \qquad \mathbf{bi} = (X^{\mathbf{a}} - 1) \circ \mathbf{bt} + \mathbf{bi}'.$$

  where $(X^{\mathbf{a}} - 1) := (X^{a_1} - 1||\cdots||X^{a_n} - 1)$
- **Final NTT checks.** The prover proves that $(\mathbf{ac}, \mathbf{ai})$ and $(\mathbf{bc}, \mathbf{bi})$ satisfy the NTT relation.

Fig. 9: The Verifiable Blind Rotation

A final INTT produces the output RLWE ciphertext $(\mathbf{ac}_{n+1}, \mathbf{bc}_{n+1}) \in \mathbb{Z}_Q^{2N_R}$. The INTT step before and after the n CMUX iterations can be directly proven using the vector NTT check. We focus on the proof of the n CMUX gates, as illustrated in Fig. 9.

4.3 Putting Everything Together

By combining the verifiable modulus switching, verifiable blind rotation initialization, and verifiable blind rotation, we obtain a SNARK specialized for TFHE bootstrapping. The overall circuit size of the bootstrapping procedure is $C = O(nN_R)$ and $n = N_R = O(\sqrt{C})$. (The detailed parameter design is deferred to Sec. 5.2.)

Table 1: TFHE-friendly SNARK Complexity

	Prover	**Verifier**	**Proof Size**
Vector Commitment	$(nN_R \log N_R)$	$O(n + N_R \log N_R)$	$O(n + N_R)$
Vector Hadamard Product	$O(nN_R)$	$O(\log n + N_R)$	$O(\log n + N_R)$
Vector NTT	$O(nN_R)$	$O(n + N_R \log N_R)$	$O(N_R)$
Vector Lookup	$O(nN_R)$	$O(\log n + N_R)$	$O(\log n + N_R)$
SNARK	$O(C \log C)$	$O(\sqrt{C} \log C)$	$O(\sqrt{C})$

In summary, the resulting SNARK achieves prover complexity $O(C \log C)$, verifier complexity $O(\sqrt{C} \log C)$, and proof size $O(\sqrt{C})$ (Tab. 1).

5 Implementation

5.1 Packed Sumcheck Performance

We conducted benchmark experiments on an Apple M2 Pro processor, evaluating performance over the Babybear field $\mathbb{F}_p$ where $p = 2^{31} - 2^{27} + 1$ [13]. We evaluate packed sumcheck with repetition parameters $k = 4$ ($\geq$ 100-bit round-by-round soundness) and $k = 5$ ($\geq$ 128-bit round-by-round soundness). We also benchmark Algorithm 1 from [5] in the BabyBear field using a 4-fold extension ($\geq$ 121-bit round-by-round soundness).

As shown in Fig. 10(a), our protocol achieves 3.25×–4.03× speedups for $k = 4$ and 2.78×–3.36× for $k = 5$ across instance sizes $N = 2^\ell$ and degrees d. Beyond the speed, packed sumcheck exhibits flexibility in handling field expansions. Prior approaches require complete re-implementation of field expansion for different multiples, with performance being sensitive to field expansion structures. In contrast, packed sumcheck accommodates any repetition factor k

[13] We additionally benchmarked both algorithms on an Intel i9-13900H processor to validate performance consistency across different hardware platforms. The detailed data are provided in Appendix E.

through simple parameter adjustment, completely independent of the underlying base field structure. This combination of consistent performance advantages and architectural flexibility makes packed sumcheck particularly valuable for practical applications.

5.2 Verifiable TFHE Bootstrapping

TFHE is instantiated with LWE parameters $n = 1024$, $q = 2^{31} - 2^{27} + 1$, and RLWE parameters $N_R = 1024$, $Q = 2^{31} - 2^{27} + 1$. The gadget decomposition parameter is set to $D_d = 4$ [1]. For TFHE-friendly SNARK, we target $\lambda = 100$ bits security (a typical choice in recent works [38,57]). Given that the base field has size 2^{31}, an expansion factor $k = 4$ is selected to achieve 100 bits security. For the vector commitment, a block RS code with rate 1/2 (relative distance 1/2) is employed. By lemma 4, the verifier needs to open 378 columns to achieve 100 bits of soundness because $\left(1 - \frac{1}{3} \cdot \frac{1}{2}\right)^{378} = \left(\frac{5}{6}\right)^{378} < 2^{-100}$. For the vector lookup IOP we set $s = 10$, which ensures $\left(\frac{2^{20}}{2^{31}}\right)^{10} < 2^{-100}$ by lemma 8.

We evaluate our SNARK performance on the 2×AMD EPYC 9474F 48-Core Processor server with 1TB RAM [14] . Tab. 2 reports a detailed cost breakdown, and Tab. 3 compares our system with the work of Thibault and Walter (TW24) [51] and Hasteboots [42]. Our prover attains a proof generation time of 2.02s, achieving a 534× speedup over TW24 [51] and a 1.98× improvement over Hasteboots [42].[15]

Table 2: Cost Breakdown of our TFHE-friendly SNARK

	Proving Time	**Verification Time**	**Proof Size**
Vector Commitment	259.04ms	50.67ms	32.58MB
Blind Rotation Initialization	16.19ms	2.15ms	32.78KB
NTT/INTT	5.36ms	4.23ms	294.84KB
Gadget Decomposition	1466.92ms	46.45ms	1.75MB
Hadamard Product	275.16ms	5.38ms	292.8KB
TFHE-friendly SNARK	**2.02s**	**108.88ms**	**34.95MB**

[14] Because modulus switching involves only five vectors of length 1024, whereas the other two phases operate on vectors of size $O(2^{20})$, its proof cost has negligible impact on TFHE-friendly SNARK performance. We therefore focus on the remaining phases.

[15] Our implementation and TW24 were evaluated on the same server. Hasteboots code is not publicly available; we therefore use the figures reported in their paper, which do not provide verifier time on Hpc7a.96xlarge (192 cores)

Tab. (a) Sumcheck Benchmark Results

Method	K	d	λ	Time (ms) at $l = \{14, 16, 18, 20\}$			
Ours	4	2	109	1.14 (4.01×)	4.54 (4.03×)	18.92 (3.88×)	79.25 (3.69×)
	4	3	106	2.19 (3.62×)	8.88 (3.63×)	37.59 (3.44×)	159.98 (3.25×)
	4	4	105	3.57 (3.48×)	14.47 (3.49×)	59.58 (3.44×)	251.38 (3.33×)
	4	5	103	5.18 (3.49×)	21.25 (3.49×)	85.90 (3.47×)	359.98 (3.34×)
	5	2	134	1.38 (3.31×)	5.45 (3.36×)	22.62 (3.25×)	95.82 (3.05×)
	5	3	131	2.68 (2.96×)	10.80 (2.99×)	44.25 (2.93×)	187.00 (2.78×)
	5	4	129	4.26 (2.92×)	17.27 (2.92×)	70.25 (2.92×)	298.02 (2.81×)
	5	5	128	6.19 (2.92×)	25.08 (2.95×)	102.89 (2.90×)	432.46 (2.78×)
[BDT24][5] (Alg. 1)	4	2	123	4.57	18.29	73.48	292.03
	4	3	122	7.94	32.25	129.49	519.34
	4	4	122	12.42	50.47	204.81	838.30
	4	5	121	18.06	74.10	298.29	1204.10

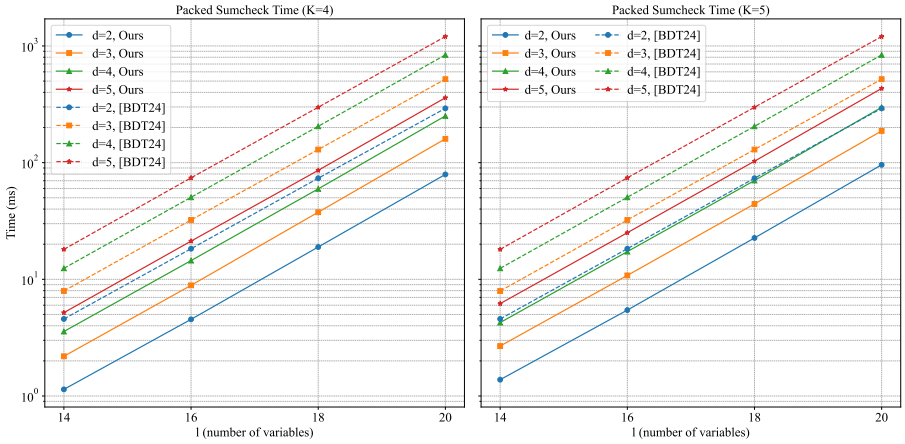

Fig. (b) Performance of our Sumcheck protocol over the Babybear field compared with Algorithm 1 from [5]

Fig. 10: Sumcheck results: (a) benchmark table and (b) corresponding performance curves.

Table 3: Verifiable TFHE Bootstrapping Performance Comparison.

	Machine	Prover Time	Verifier Time	Proof Size
TW24 [51]	2×AMD EPYC 9474F 48-Core Processor	17min59s	7.3ms	192.48KB
Hasteboots [42]	Hpc7a.96xlarge (192 cores)	4s	–	107MB
Our work	2×AMD EPYC 9474F 48-Core Processor	2.02s	108.88ms	34.95MB

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015), http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml
2. Atapoor, S., Baghery, K., Pereira, H.V.L., Spiessens, J.: Verifiable FHE via lattice-based snarks. IACR Commun. Cryptol. **1**(1), 24 (2024). https://doi.org/10.62056/A6KSDKP10, https://doi.org/10.62056/a6ksdkp10
3. Attema, T., Fehr, S., Klooß, M.: Fiat-shamir transformation of multi-round interactive proofs. In: Kiltz, E., Vaikuntanathan, V. (eds.) Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13747, pp. 113–142. Springer (2022). https://doi.org/10.1007/978-3-031-22318-1_5, https://doi.org/10.1007/978-3-031-22318-1_5
4. Bagad, S., Dao, Q., Domb, Y., Thaler, J.: Speeding up sum-check proving. IACR Cryptol. ePrint Arch. p. 1117 (2025), https://eprint.iacr.org/2025/1117
5. Bagad, S., Domb, Y., Thaler, J.: The sum-check protocol over fields of small characteristic. IACR Cryptol. ePrint Arch. p. 1046 (2024), https://eprint.iacr.org/2024/1046
6. Barak, B.: How to go beyond the black-box simulation barrier. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, Las Vegas, Nevada, USA, October 14-17, 2001. pp. 106–115. IEEE Computer Society (2001). https://doi.org/10.1109/SFCS.2001.959885, https://doi.org/10.1109/SFCS.2001.959885
7. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018), https://doi.org/10.4230/LIPIcs.ICALP.2018.14
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptol. ePrint Arch. p. 46 (2018), http://eprint.iacr.org/2018/046
9. Ben-Sasson, E., Carmon, D., Ishai, Y., Kopparty, S., Saraf, S.: Proximity gaps for reed-solomon codes. In: Irani, S. (ed.) 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020. pp. 900–909. IEEE (2020), https://doi.org/10.1109/FOCS46700.2020.00088
10. Bois, A., Cascudo, I., Fiore, D., Kim, D.: Flexible and efficient verifiable computation on encrypted data. In: Garay, J.A. (ed.) Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12711, pp. 528–558. Springer (2021). https://doi.org/10.1007/978-3-030-75248-4_19, https://doi.org/10.1007/978-3-030-75248-4_19
11. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer (2012), https://doi.org/10.1007/978-3-642-32009-5_50

12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Trans. Comput. Theory **6**(3), 13:1–13:36 (2014), https://doi.org/10.1145/2633600
13. Brehm, M., Chen, B., Fisch, B., Resch, N., Rothblum, R.D., Zeilberger, H.: Blaze: Fast snarks from interleaved RAA codes. In: Fehr, S., Fouque, P. (eds.) Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 15604, pp. 123–152. Springer (2025). https://doi.org/10.1007/978-3-031-91134-7_5, https://doi.org/10.1007/978-3-031-91134-7_5
14. Canetti, R., Chen, Y., Holmgren, J., Lombardi, A., Rothblum, G.N., Rothblum, R.D., Wichs, D.: Fiat-shamir: from practice to theory. In: Charikar, M., Cohen, E. (eds.) Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019. pp. 1082–1090. ACM (2019). https://doi.org/10.1145/3313276.3316380, https://doi.org/10.1145/3313276.3316380
15. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: Vitter, J.S. (ed.) Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998. pp. 209–218. ACM (1998). https://doi.org/10.1145/276698.276741, https://doi.org/10.1145/276698.276741
16. Cascudo, I., Costache, A., Cozzo, D., Fiore, D., Guimarães, A., Soria-Vazquez, E.: Verifiable computation for approximate homomorphic encryption schemes. In: Kalai, Y.T., Kamara, S.F. (eds.) Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part VII. Lecture Notes in Computer Science, vol. 16006, pp. 643–677. Springer (2025). https://doi.org/10.1007/978-3-032-01907-3_21, https://doi.org/10.1007/978-3-032-01907-3_21
17. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer, Cham, Switzerland, Hong Kong, China (Dec 3–7, 2017). https://doi.org/10.1007/978-3-319-70694-8_15
18. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zksnarks with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) 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. Lecture Notes in Computer Science, vol. 12105, pp. 738–768. Springer (2020), https://doi.org/10.1007/978-3-030-45721-1_26
19. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10031, pp. 3–33 (2016). https://doi.org/10.1007/978-3-662-53887-6_1, https://doi.org/10.1007/978-3-662-53887-6_1
20. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1), 34–91 (2020). https://doi.org/10.1007/S00145-019-09319-X, https://doi.org/10.1007/s00145-019-09319-x

21. Dao, Q., Thaler, J.: Constraint-packing and the sum-check protocol over binary tower fields. IACR Cryptol. ePrint Arch. p. 1038 (2024), https://eprint.iacr.org/2024/1038
22. Dao, Q., Thaler, J.: More optimizations to sum-check proving. IACR Cryptol. ePrint Arch. p. 1210 (2024), https://eprint.iacr.org/2024/1210
23. Diamond, B.E., Posen, J.: Succinct arguments over towers of binary fields. In: Fehr, S., Fouque, P. (eds.) Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 15604, pp. 93–122. Springer (2025). https://doi.org/10.1007/978-3-031-91134-7_4, https://doi.org/10.1007/978-3-031-91134-7_4
24. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 617–640. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_24, https://doi.org/10.1007/978-3-662-46800-5_24
25. Dwork, C., Naor, M., Reingold, O., Stockmeyer, L.J.: Magic functions. J. ACM **50**(6), 852–921 (2003). https://doi.org/10.1145/950620.950623, https://doi.org/10.1145/950620.950623
26. Eagen, L., Fiore, D., Gabizon, A.: cq: Cached quotients for fast lookups. IACR Cryptol. ePrint Arch. p. 1763 (2022), https://eprint.iacr.org/2022/1763
27. Fiore, D., Nitulescu, A., Pointcheval, D.: Boosting verifiable computation on encrypted data. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12111, pp. 124–154. Springer (2020). https://doi.org/10.1007/978-3-030-45388-6_5, https://doi.org/10.1007/978-3-030-45388-6_5
28. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. p. 953 (2019), https://eprint.iacr.org/2019/953
29. Ganesh, C., Nitulescu, A., Soria-Vazquez, E.: Rinocchio: Snarks for ring arithmetic. J. Cryptol. **36**(4), 41 (2023), https://doi.org/10.1007/s00145-023-09481-3
30. Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III. Lecture Notes in Computer Science, vol. 14006, pp. 257–286. Springer (2023). https://doi.org/10.1007/978-3-031-30620-4_9, https://doi.org/10.1007/978-3-031-30620-4_9
31. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
32. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings,

Part I. Lecture Notes in Computer Science, vol. 8042, pp. 75–92. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4_5, https://doi.org/10.1007/978-3-642-40041-4_5
33. Goldreich, O.: Probabilistic proof systems. In: Modern Cryptography, Probabilistic Proofs and Pseudorandomness, pp. 39–72. Springer (1999)
34. Goldwasser, S., Kalai, Y.T.: On the (in)security of the fiat-shamir paradigm. In: 44th Symposium on Foundations of Computer Science, FOCS 2003, Cambridge, MA, USA, October 11-14, 2003, Proceedings. pp. 102–113. IEEE Computer Society (2003). https://doi.org/10.1109/SFCS.2003.1238185, https://doi.org/10.1109/SFCS.2003.1238185
35. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part II. Lecture Notes in Computer Science, vol. 14082, pp. 193–226. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023). https://doi.org/10.1007/978-3-031-38545-2_7
36. Graham, R.L.: Concrete mathematics: a foundation for computer science. Pearson Education India (1994)
37. Gruen, A.: Some improvements for the PIOP for zerocheck. IACR Cryptol. ePrint Arch. p. 108 (2024), https://eprint.iacr.org/2024/108
38. Guo, Y., Liu, X., Huang, K., Qu, W., Tao, T., Zhang, J.: Deepfold: Efficient multilinear polynomial commitment from reed-solomon code and its application to zero-knowledge proofs. IACR Cryptol. ePrint Arch. p. 1595 (2024), https://eprint.iacr.org/2024/1595
39. Holmgren, J., Rothblum, R.: Faster sounder succinct arguments and iops. IACR Cryptol. ePrint Arch. p. 994 (2022), https://eprint.iacr.org/2022/994
40. Kothapalli, A., Setty, S.T.V.: Neutronnova: Folding everything that reduces to zero-check. IACR Cryptol. ePrint Arch. p. 1606 (2024), https://eprint.iacr.org/2024/1606
41. Kothapalli, A., Setty, S.T.V., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 13510, pp. 359–388. Springer (2022). https://doi.org/10.1007/978-3-031-15985-5_13, https://doi.org/10.1007/978-3-031-15985-5_13
42. Liu, F., Liang, H., Zhang, T., Hu, Y., Xie, X., Tan, H., Yu, Y.: Hasteboots: Proving FHE bootstrapping in seconds. IACR Cryptol. ePrint Arch. p. 261 (2025), https://eprint.iacr.org/2025/261
43. Liu, T., Zhang, Y.: Efficient snarks for boolean circuits via sumcheck over tower fields. IACR Cryptol. ePrint Arch. p. 594 (2025), https://eprint.iacr.org/2025/594
44. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I. pp. 2–10. IEEE Computer Society (1990). https://doi.org/10.1109/FSCS.1990.89518, https://doi.org/10.1109/FSCS.1990.89518
45. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic

Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer (2010). https://doi.org/10.1007/978-3-642-13190-5_1, https://doi.org/10.1007/978-3-642-13190-5_1
46. Micciancio, D., Polyakov, Y.: Bootstrapping in fhew-like cryptosystems. In: WAHC '21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021. pp. 17–28. WAHC@ACM (2021). https://doi.org/10.1145/3474366.3486924, https://doi.org/10.1145/3474366.3486924
47. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005. pp. 84–93. ACM (2005). https://doi.org/10.1145/1060590.1060603, https://doi.org/10.1145/1060590.1060603
48. Ron-Zewi, N., Rothblum, R.D.: Proving as fast as computing: succinct arguments with constant prover overhead. In: Leonardi, S., Gupta, A. (eds.) STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022. pp. 1353–1363. ACM (2022). https://doi.org/10.1145/3519935.3519956, https://doi.org/10.1145/3519935.3519956
49. Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 704–737. Springer (2020), https://doi.org/10.1007/978-3-030-56877-1_25
50. Thaler, J.: Proofs, arguments, and zero-knowledge. Found. Trends Priv. Secur. **4**(2-4), 117–660 (2022). https://doi.org/10.1561/3300000030, https://doi.org/10.1561/3300000030
51. Thibault, L.T., Walter, M.: Towards verifiable FHE in practice: Proving correct execution of tfhe's bootstrapping using plonky2. IACR Cryptol. ePrint Arch. p. 451 (2024), https://eprint.iacr.org/2024/451
52. Viand, A., Knabenhans, C., Hithnawi, A.: Verifiable fully homomorphic encryption. CoRR **abs/2301.07041** (2023). https://doi.org/10.48550/ARXIV.2301.07041, https://doi.org/10.48550/arXiv.2301.07041
53. Wei, Y., Zhang, X., Deng, Y.: Transparent snarks over galois rings. In: Jager, T., Pan, J. (eds.) Public-Key Cryptography - PKC 2025 - 28th IACR International Conference on Practice and Theory of Public-Key Cryptography, Røros, Norway, May 12-15, 2025, Proceedings, Part I. Lecture Notes in Computer Science, vol. 15674, pp. 418–451. Springer (2025). https://doi.org/10.1007/978-3-031-91820-9_14, https://doi.org/10.1007/978-3-031-91820-9_14
54. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 733–764. Springer (2019), https://doi.org/10.1007/978-3-030-26954-8_24
55. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV. Lecture Notes in Computer

Science, vol. 13510, pp. 299–328. Springer (2022), https://doi.org/10.1007/978-3-031-15985-5_11
56. Zeilberger, H., Chen, B., Fisch, B.: BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part X. Lecture Notes in Computer Science, vol. 14929, pp. 138–169. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024). https://doi.org/10.1007/978-3-031-68403-6_5
57. Zhang, X., Wang, R., Liu, Z., Xiang, B., Deng, Y., Lu, X.: FHE-SNARK vs. SNARK-FHE: from analysis to practical verifiable computation. IACR Cryptol. ePrint Arch. p. 302 (2025), https://eprint.iacr.org/2025/302
58. Zhou, Z., Li, Y., Wang, Y., Yang, Z., Zhang, B., Hong, C., Wei, T., Chen, W.: ZHE: efficient zero-knowledge proofs for HE evaluations. In: Blanton, M., Enck, W., Nita-Rotaru, C. (eds.) IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025. pp. 3328–3346. IEEE (2025). https://doi.org/10.1109/SP61157.2025.00199, https://doi.org/10.1109/SP61157.2025.00199

Appendix

A Additional Preliminaries

A.1 Folding Scheme

Definition 6 (Folding Scheme). *Consider a relation $\mathcal{R}$ over public parameters, structure, instance, and witness tuples. A folding scheme for $\mathcal{R}$ consists of a PPT generator algorithm $\mathcal{G}$, a deterministic encoder algorithm $\mathcal{K}$, and a pair of PPT algorithms $\mathcal{P}$ and $\mathcal{V}$ denoting the prover and verifier respectively, with the following interface:*

- $\mathcal{G}(1^\lambda) \to \mathsf{pp}$*: on input security parameter λ, samples public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, s) \to (\mathsf{pk}, \mathsf{vk})$*: on input* pp *and a common structure s between instances to be folded, outputs a prover key* pk *and a verifier key* vk.
- $\mathcal{P}(\mathsf{pk}, (u_1, w_1), (u_2, w_2)) \to (u, w)$*: on input instance–witness tuples (u_1, w_1) and (u_2, w_2), outputs a new instance–witness tuple (u, w) of the same size.*
- $\mathcal{V}(\mathsf{vk}, u_1, u_2) \to u$*: on input instances u_1 and u_2, outputs a new instance u.*

Let

$$(u, w) \leftarrow \big\langle \mathcal{P}(\mathsf{pk}, w_1, w_2),\ \mathcal{V}(\mathsf{vk}) \big\rangle (u_1, u_2)$$

denote the verifier's output instance u and the prover's output witness w from the interaction of $\mathcal{P}$ and $\mathcal{V}$ on witnesses (w_1, w_2), prover key pk*, verifier key* vk *and instances (u_1, u_2). Likewise, let*

$$\mathsf{tr} \ = \ \big\langle \mathcal{P}(\mathsf{pk}, w_1, w_2),\ \mathcal{V}(\mathsf{vk}) \big\rangle (u_1, u_2)$$

denote the corresponding interaction transcript. A folding scheme satisfies perfect completeness if for all PPT adversaries $\mathcal{A}$,

$$Pr\left[(\mathsf{pp}, \mathsf{s}, u, w) \in \mathcal{R} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (s, (u_1, w_1), (u_2, w_2)) \leftarrow \mathcal{A}(\mathsf{pp}), \\ (\mathsf{pp}, s, u_1, w_1), (\mathsf{pp}, s, u_2, w_2) \in \mathcal{R}, \\ (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp}, s), \\ (u, w) \leftarrow \big\langle \mathcal{P}(\mathsf{pk}, w_1, w_2),\ \mathcal{V}(\mathsf{vk}) \big\rangle (u_1, u_2) \end{array} \right] = 1.$$

A folding scheme satisfies knowledge soundness if for any expected polynomial-time adversary $\mathcal{P}^*$ there exists an expected polynomial-time extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (pp,s,u_1,w_1) \in \mathcal{R} & (s,(u_1,u_2)) \leftarrow \mathcal{P}^*(\mathsf{pp}), \\ (pp,s,u_2,w_2) \in \mathcal{R} & (\mathsf{pk},\mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp},s), \\ & \mathsf{tr} \leftarrow \langle \mathcal{P}^*(\mathsf{pk},\cdot),\, \mathcal{V}(\mathsf{vk}) \rangle (u_1,u_2), \\ & (u,w) \leftarrow \mathcal{E}(\mathsf{pp},\mathsf{pk},\mathsf{vk},s,u_1,u_2,\mathsf{tr}) : \ (\mathsf{pp},s,u,w) \in \mathcal{R} \end{array}\right] \geq$$

$$\Pr\left[\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathcal{G}(1^\lambda), \\ (\mathsf{pp},s,u,w) \in \mathcal{R} & (s,(u_1,u_2)) \leftarrow \mathcal{P}^*(\mathsf{pp}), \\ & (\mathsf{pp},s,u_1,u_2) \in \mathcal{R}, \\ & (\mathsf{pk},\mathsf{vk}) \leftarrow \mathcal{K}(\mathsf{pp},s), \\ & (u,w^*) \leftarrow \langle \mathcal{P}^*(\mathsf{pk},\cdot),\, \mathcal{V}(\mathsf{vk}) \rangle (u_1,u_2) \end{array}\right] - \mathrm{negl}(\lambda).$$

where ρ denotes arbitrary input randomness for $\mathcal{P}^*$. We call a transcript an accepting transcript *if $\mathcal{P}$ outputs a satisfying folded witness w for the folded instance u. We consider a folding scheme* non-trivial *if the communication costs and $\mathcal{V}$'s computation are lower in the case where $\mathcal{V}$ participates in the folding scheme and then checks a witness sent by $\mathcal{P}$ for the folded instance than in the case where $\mathcal{V}$ checks witnesses sent by $\mathcal{P}$ for each of the original instances.*

A.2 SNARKs

We adapt the definition from Rinocchio [29]. Let $\mathcal{R}$ be an efficiently computable binary relation which consists of pairs of the form (x, w) where x is a statement and w is a witness. Let $\mathbf{L}$ be the language associated with the relation $\mathcal{R}$, i.e.$\mathcal{L} = \{x | \exists w, \text{s.t.} \mathcal{R}(x, w) = 1\}$.

A proof or argument system for R consists in a triple of PPT algorithms $\mathit{\Pi} = (\mathbf{Setup}, \mathbf{Prove}, \mathbf{Verify})$ defined as follows:

- $\text{Setup}(1^\lambda) \rightarrow (\sigma, \text{vk})$: take a security parameter λ and outputs a common (structured) reference string σ together with private verification information vk.
- $\text{Prove}(\sigma, x, w) \rightarrow \pi$: on input σ, a statement x and witness w, outputs an argument π.
- $\text{Verify}(\sigma, \text{vk}, x, \pi) \rightarrow 1/0$: on input σ, the private verification key vk, a statement x and a proof π, it outputs either 1 indicating accepting the argument or 0 for rejecting it.

Definition 7 (SNARK). *A triple of polynomial time algorithms* (**Setup, Prove**, **Verify**) *is a SNARK for an NP relation $\mathcal{R}$, if the following properties are satisfied:*

- *Completeness. For all $(x, w) \in \mathcal{R}$, the following holds:*

$$Pr\left[\mathbf{Verify}(\sigma, vk, x, \pi) = 1 \,\middle|\, \begin{array}{l} (\sigma, vk) \rightarrow \mathbf{Setup}(1^\lambda) \\ \pi \rightarrow \mathbf{Prove}(\sigma, x, w) \end{array}\right] = 1.$$

- *Knowledge Soundness. For any PPT adversary $\mathcal{A}$, there exists a PPT algorithm $\mathcal{E}_{\mathcal{A}}$ such that the following probability is negligible in λ:*

$$Pr\left[\begin{array}{c}\mathbf{Verify}(\sigma, vk, \tilde{x}, \tilde{\pi}) = 1 \\ \wedge \mathcal{R}(\tilde{x}, w') = 0\end{array} \middle| \begin{array}{l}(\sigma, vk) \rightarrow \mathbf{Setup}(1^\lambda) \\ ((\tilde{x}, \tilde{\pi}); w') \rightarrow \mathcal{A}|\mathcal{E}_{\mathcal{A}}(\sigma)\end{array}\right]$$

- *A non-interactive argument of knowledge satisfies knowledge is succinct if the size of proof π and the time to* **Verify** *are sublinear in the size of the statement proven.*

If a SNARK does not require a private verification key vk, then the SNARK scheme is referred to as a public-key SNARK. If a SNARK does not require a CRS σ, then the SNARK scheme is referred to as transparent.

A.3 Polynomial Commitment Scheme

We adapt the definition from Brakedown [35]. A polynomial commitment scheme for multilinear polynomial over $\mathbb{F}_p$ is a tuple of four protocols PC = (Gen,Commit, Open,Eval):

- $pp \leftarrow \text{Gen}(1^\lambda, \mu)$ takes as input μ (the number of variables in a multilinear polynomial); produces public parameters pp.
- $\mathcal{C} \leftarrow \text{Commit}(pp, \mathcal{G}, \gamma)$: takes any input a $\mu-$variate multilinear polynomial over the field $\mathcal{G} \in \mathbb{F}_p[\mu]$ and a random string γ produces a commitment $\mathcal{C}$.
- $b \leftarrow \text{Open}(pp, \mathcal{C}, \mathcal{G}, \gamma)$: verifies the opening of commitment $\mathcal{C}$ to the $\mu-$variate multilinear polynomial $\mathcal{G} \in \mathbb{F}_p[\mu]$; outputs $b \in \{0, 1\}$.
- $b \leftarrow \text{Eval}(pp, \mathcal{C}, z, v, \mu, \mathcal{G})$ is a protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. Both $\mathcal{V}$ and $\mathcal{P}$ hold a commitment $\mathcal{C}$, the number of variables μ, a scalar $v \in \mathbb{F}_p$ and $z \in \mathbb{F}_p^\mu$. $\mathcal{P}$ additionally knows a $\mu-$variate multilinear polynomial $\mathcal{G} \in \mathbb{F}_p[\mu]$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $\mathcal{G}(z) = v$. At the end of the protocol, $\mathcal{V}$ outputs $b \in \{0, 1\}$.

Definition 8 (Polynomial Commitment Scheme). *A tuple of four protocol (Gen,Commit,Open,Eval) is an extractable polynomial commitment scheme for multilinear polynomials over field $\mathbb{F}_p$ if the following conditions hold.*

- ***Completeness.*** *For any $\mu-$variate multilinear polynomial $\mathcal{G} \in \mathbb{F}_p[\mu]$,*

$$\Pr\left[\begin{array}{c}pp \rightarrow Gen(1^\lambda, \mu); \mathcal{C} \leftarrow Commit(pp, \mathcal{G}, \gamma) : \\ Eval(pp, \mathcal{C}, r, v, \mu, \mathcal{G}) = 1 \wedge v = \mathcal{G}(r)\end{array}\right] \geq 1 - negl(\lambda)$$

- ***Binding.*** *For any PPT adversary $\mathcal{A}$, size parameter $\mu \geq 1$,*

$$\Pr\left[\begin{array}{c}pp \leftarrow Gen(1^\lambda, \mu); \mathcal{C}, \mathcal{G}_0, \mathcal{G}_1 \leftarrow \mathcal{A}(pp); \\ b_0 \leftarrow Open(pp, \mathcal{C}, \mathcal{G}_0, \gamma); b_1 \leftarrow Open(pp, \mathcal{C}, \mathcal{G}_1, \gamma); \\ b_0 = b_1 \neq 0 \wedge \mathcal{G}_0 \neq \mathcal{G}_1\end{array}\right] \leq negl(\lambda)$$

- ***Knowledge soundness.*** *Eval is a succinct argument of knowledge for the following NP relation given $pp \leftarrow Gen(1^\lambda, \mu)$.*

$$\mathcal{R}_{Eval}(pp) = \{\langle(\mathcal{C}, z, v), (\mathcal{G})\rangle : \mathcal{G} \in GR(p^s, r)[\mu] \wedge \mathcal{G}(z) = v \wedge Open(pp, \mathcal{C}, \mathcal{G}, \gamma) = 1\}$$

B Supplementary Proofs for the Packed Sumcheck

B.1 Forward Difference

Definition 9. *The forward difference operator for a function $f(x)$ over $\mathbb{F}_p$ at discrete, equally spaced points $x_k = x_0 + kh$ (where h is the step size) is expressed as:*

$$\Delta f(x_k) = f(x_{k+1}) - f(x_k)$$

Higher-order differences are defined recursively. The n-th order forward difference is:

$$\Delta^n f(x_k) = \Delta(\Delta^{n-1} f(x_k)) = \Delta^{n-1} f(x_{k+1}) - \Delta^{n-1} f(x_k)$$

When the function f is clear from the context, we may denote $\Delta_k^n := \Delta^n f(x_k)$.

Fact 1 *[36] For a polynomial f of degree at most k over $\mathbb{F}_p$, its k-th finite difference, $\Delta^k f(x)$, is constant.*

Algorithm 2: Extrapolating f via Forward Differences

Input: Evaluations $f(0), \ldots, f(k-1)$ and an integer m
Output: Evaluations $f(k), \ldots, f(k+m-1)$

```
// Construct initial difference table
```
for $i \in [0, k-1]$ **do**
  Set $\Delta_i^0 = f(i)$;
for $i \leftarrow 1$ ***to*** $k-1$ **do**
  for $j \leftarrow 0$ ***to*** $k-1-i$ **do**
    $\Delta_j^i \leftarrow \Delta_{j+1}^{i-1} - \Delta_j^{i-1}$;
```
// Store the diagonal
```
for $i \in [0, k-1]$ **do**
  Set $D_i \leftarrow \Delta_{k-1-i}^i$;
```
// Extrapolate m new values
```
for $i \leftarrow 1$ ***to*** m **do**
  for $j \leftarrow k-2$ ***down to*** 0 **do**
    $D_j \leftarrow D_j + D_{j+1}$;
  Set $f(k-1+i) \leftarrow D_0$;
return $f(k), \ldots, f(k+m-1)$

Lemma 11. *Let f be a univariate polynomial of degree at most $k-1$ over $\mathbb{F}_p$, where $p > k$. Given the initial values $f(0), \ldots, f(k-1)$, for any integer m such that $p > k+m-1$, Algorithm 2 computes the evaluations $f(k), \ldots, f(k+m-1)$ using $O(k^2 + km)$ additions in $\mathbb{F}_p$.*

Proof. **Correctness.** The correctness of the algorithm follows from the fact that forward-difference extrapolation is equivalent to Lagrange polynomial interpolation. A univariate polynomial f of degree at most $k-1$ is uniquely determined

by its evaluations at k distinct points, say $f(s), \ldots, f(s+k-1)$. The Lagrange interpolation formula gives the evaluation at a new point $s+k$ as:

$$f(s+k) = \sum_{i=0}^{k-1} c_i \cdot f(s+i), \quad \text{where} \quad c_i = \prod_{\substack{j=0 \\ j \neq i}}^{k-1} \frac{k-j}{i-j}.$$

Algorithm 2 computes the same value $f(s+k)$ by iteratively updating a diagonal of difference terms. This process is a linear transformation of the initial values $f(s), \ldots, f(s+k-1)$. A direct expansion of the forward-difference formulas reveals that the coefficient of each $f(s+i)$ in the final computation of $f(s+k)$ is precisely the Lagrange coefficient c_i defined above. This equivalence holds for any sequence of k consecutive points, thus proving the algorithm's correctness.

Complexity Analysis. The computational cost is dominated by two phases:

- Initialization: The construction of the initial difference table involves computing Δ_j^i for $i \in [1, k-1]$ and $j \in [0, k-1-i]$. The total number of subtractions is $\sum_{i=1}^{k-1}(k-i) = \frac{k(k-1)}{2}$, which is $O(k^2)$.
- Extrapolation: The algorithm computes m new values. Each new value is obtained by updating the diagonal, which requires $k-1$ additions. This phase costs $m(k-1)$ additions, resulting in a complexity of $O(km)$.

Combining both phases, the total complexity is $O(k^2 + km)$ additions in $\mathbb{F}_p$. □

B.2 Avoiding Repeated Polynomial Commitments via Brakedown

Since the commitment size halves in each round, the total commitment cost over the protocol never exceeds the initial commitment. Moreover, under Brakedown polynomial commitments, no additional commitments are required per round: commitments encode polynomials as row-linear combinations of a fixed coefficient matrix, allowing the verifier to update the combination coefficients in place as a function of the challenges. Starting from an $(\ell-1)$-variate instance, after $(\ell/2)$ rounds of the packed sumcheck the resulting $(\ell/2-1)$-variate polynomials are row–linear combinations of the original matrix. At that point the prover sends these result polynomials once, the verifier checks the linear combination constraints via Brakedown's tensor IOP, and then completes the residual sumcheck of size $(\sqrt{N})$ locally.

The Brakedown framework enables efficient verification of linear combinations, where the prover encodes polynomial coefficient matrices row-wise while the verifier checks linear combinations of these encoded rows. In our packed sumcheck protocol, the prover commits to $2kd$ polynomials represented as $2kd \cdot 2^{l/2}$ length-$2^{l/2-1}$ vectors, while the verifier maintains k coefficient vectors of length $2kd \cdot 2^{l/2}$ to track linear combinations. Through iterative rounds, the verifier updates a $2k \times (kd \cdot 2^{l/2})$ matrix via challenge: driven linear transformations applied to the original matrix rows. Crucially, after $l/2$ rounds the verifier can locally complete the remaining sumchecks without requiring additional polynomial commitments from the prover. This approach eliminates polynomial recommitment each round.

B.3 Round-by-Round Soundness of the Packed Sumcheck

Lemma 12 (Lemma 1). *Instantiating Protocol 2 with the Brakedown polynomial commitment achieves a round-by-round soundness error of* $\left(\frac{(2k-1)d}{p}\right)^k$. *For other polynomial commitments, the round-by-round soundness error is* $\max\left((\frac{(2k-1)d}{p})^k, \frac{2kd(\log N-2)^k}{p^k}\right)$. *The prover complexity of packed sumcheck is* $O((kd^2+k^2d)N)\cdot \mathsf{mul} + O(k^2d^2N)\cdot \mathsf{add}$.

Proof. **Round-by-round soundness.**

- **Brakedown.** Consider the i-th round,$i \in [1, l-1]$, where the statements are

$$\sum_{\mathbf{x}\in\{0,1\}^{l-i}} \prod_{j=0}^{d-1} f_0^{(j)}(\mathbf{x}) = h_0, \quad \ldots, \quad \sum_{\mathbf{x}\in\{0,1\}^{l-i}} \prod_{j=0}^{d-1} f_{2k-1}^{(j)}(\mathbf{x}) = h_{2k-1}.$$

  If the prover attempts to cheat in round i, then at least one of these $2k$ statements is incorrect, which implies that the interpolated polynomial is invalid. In this case, a cheating prover would send a polynomial $\tilde{G}(r) \neq F(r)$ that must nevertheless satisfy $\tilde{G}(r_i) = F(r_i)$ for all k challenge points r_i. Since both F and $\tilde{G}$ have degree at most $(2k-1)d$, the Schwartz–Zippel lemma guarantees that for a random $r \in \mathbb{F}_p$, the probability that $F(r) = \tilde{G}(r)$ is at most $\frac{(2k-1)d}{p}$. With k independently chosen challenges, the cheating probability is bounded by $\left(\frac{(2k-1)d}{p}\right)^k$. Hence, the packed sumcheck protocol achieves round-by-round soundness error $\left(\frac{(2k-1)d}{p}\right)^k$.
- **Other Polynomial Commitment.** When using any other polynomial commitment scheme, the verifier must additionally check the correctness of the new commitment polynomial. The resulting multilinear polynomial has degree at most $\log N - 2$ (Round 1). Over k independently chosen points, the Schwartz–Zippel lemma gives an agreement probability of at most $\frac{(\log N-2)^k}{p^k}$. There are $2kd$ such polynomials; by a union bound, the total error probability is at most $\frac{2kd\,(\log N-2)^k}{p^k}$. Hence, the packed sumcheck protocol achieves round-by-round soundness error $\max\left((\frac{(2k-1)d}{p})^k, \frac{2kd(\log N-2)^k}{p^k}\right)$.

Prover complexity.

- Computation of $F(r)$:
  The polynomial $F(r)$ is the product of d terms, each a univariate polynomial of degree at most $2k-1$, so $\deg(F) \le d(2k-1)$. While $F(r)$ could be computed by interpolation using $d(2k-1)+1$ evaluations.
  1. Define $\psi_{\mathbf{x}'}^{(j)}(r) = \sum_{i=0}^{2k-1} L_i(r) g_i^{(j)}(\mathbf{x}')$ with $\deg_r(\psi_{\mathbf{x}'}^{(j)}) \le 2k-1$. For any $i \in \{0,\ldots,2k-1\}$, we have $\psi_{\mathbf{x}'}^{(j)}(i) = g_i^{(j)}(\mathbf{x}')$. The remaining $d(2k-1)+1-(2k-1)$ evaluations are obtained via forward differences algorithm 2, requiring only $O(k^2d^2)$ additions.

2. For each hypercube position $\mathbf{x}'$, compute $\psi_{\mathbf{x}'}(r) = \prod_{j=1}^{d} \psi_{\mathbf{x}'}^{(j)}(r)$. The prover then evaluates $\sum_{\mathbf{x}' \in \{0,1\}^{l-1}} \psi_{\mathbf{x}'}(r)$ at all points in $\{0, \ldots, d(2k-1)\}$. Using the precomputed factors from Step 1, each $\mathbf{x}'$ requires $d-1$ multiplications to form $\psi_{\mathbf{x}'}(r)$.

Overall, computing $F(r)$ requires $d(2k-1)(d-1) \cdot 2^{l-1} = O(kd^2) \cdot 2^{l-1}$ multiplications.

- Updating polynomials:
  1. After computing $F(r)$, the prover updates the polynomials through k rounds of linear combinations over $2kd$ polynomial vectors. A direct implementation requires $2k^2 d \cdot 2^{l-1}$ multiplications.

Thus, the first round costs

$$\big(d(2k-1)(d-1) + 2k^2 d\big) \cdot 2^{l-1} \cdot \mathsf{mul} + O(k^2 d^2) \cdot 2^{l-1} \cdot \mathsf{add}.$$

As each subsequent round halves the problem size, the total protocol cost is bounded by twice the first round's complexity. Therefore, the prover's overall complexity across all rounds is

$$O\big((kd^2 + k^2 d)N\big) \cdot \mathsf{mul} + O(k^2 d^2 N) \cdot \mathsf{add}.$$

Following Bagad et al. [5], the evaluations at k points r_i can be optimized via subtraction. For all $i \in [0, k-1]$:

$$\sum_{\mathbf{x}' \in \{0,1\}^{l-1}} \psi_{\mathbf{x}'}(r_i) + \sum_{\mathbf{x}' \in \{0,1\}^{l-1}} \psi_{\mathbf{x}'}(r_{k+i}) = h_i.$$

This relation allows the computation of only one term in each pair, deriving the other by subtraction, thereby reducing the workload.

For polynomial updates, Strassen's algorithm can further reduce complexity, particularly when k is small. By partitioning each length-2^{l-1} vector into k-sized blocks, these combinations reduce to two $k \times k$ matrix multiplications per round. Using Strassen's algorithm (requiring $O(k^{2.807})$ multiplications and additions), the total cost becomes

$$O\big((kd^2 + k^{1.807} d)N\big) \cdot \mathsf{mul} + O(k^2 d^2 N) \cdot \mathsf{add}.$$

□

B.4 Round-by-Round Soundness of the Packed ZeroCheck

Lemma 13 (Lemma 2). *The packed ZeroCheck algorithm 3 with Brakedown commitment achieves a round-by-round soundness error of*

$$\max\left(\frac{k \cdot l^k}{p^k}, \frac{(dk+1)^k}{p^k}, \frac{((2k-1)(d+1))^k}{p^k}\right).$$

Moreover, the prover's computational complexity is bounded by

$$O\big((kd^2 + k^2 d)N\big) \cdot \mathit{mul} + O(k^2 d^2 N) \cdot \mathit{add}.$$

Proof. **Round-by-round soundness.**

Define the multilinear extensions:

$$f'(r) = \sum_{x \in \{0,1\}^l} eq(r,x) f(x), \quad h'(r) = \sum_{x \in \{0,1\}^l} eq(r,x) h(x).$$

If $f \neq h$ on any point of the hypercube, then $f' \neq h'$ as polynomials. Both f' and h' are multilinear polynomials, so their total degree is at most l. By the Schwartz-Zippel lemma, for k random points we have:

$$\Pr[f'(r_i) = h'(r_i) \ \forall i \in [k]] \leq \left(\frac{l}{p}\right)^k.$$

When proving k statements, the union bound ensures that the soundness error is at most $\frac{k \cdot l^k}{p^k}$.

For the second round, if the prover submits $F' \neq F$ or $H' \neq H$ where $\deg(F), \deg(H) \leq d(k-1)+1$, then

$$\Pr[F'(r_i) = F(r_i) \ \forall i \in [k]] \leq \left(\frac{d(k-1)+1}{p}\right)^k.$$

The packed sumcheck protocol achieves a soundness error bounded by $\frac{((2k-1)(d+1))^k}{p^k}$. Therefore, the round-by-round soundness error ϵ satisfies

$$\epsilon \leq \max\left(\frac{l^k}{p^k}, \frac{(d(k-1)+1)^k}{p^k}, \frac{((2k-1)(d+1))^k}{p^k}\right).$$

Complexity of computing $F(\mathbf{s}, r)$.

Consider $F(\mathbf{s}, r)$:

$$\begin{aligned} F(\mathbf{s}, r) &= \sum_{\mathbf{x} \in \{0,1\}^l} \left(\sum_{i=0}^{k-1} s_i \cdot eq(\mathbf{r}_i, \mathbf{x})\right) \prod_{s=0}^{d-1} \left(\sum_{j=0}^{k-1} L_j(r) f_j^{(s)}(\mathbf{x})\right) \\ &= \sum_{i=0}^{k-1} s_i \sum_{\mathbf{x} \in \{0,1\}^l} eq(\mathbf{r}_i, \mathbf{x}) \prod_{s=0}^{d-1} \left(\sum_{j=0}^{k-1} L_j(r) f_j^{(s)}(\mathbf{x})\right). \end{aligned}$$

Define

$$g_{\mathbf{r}_i}(r) = \sum_{\mathbf{x} \in \{0,1\}^l} eq(\mathbf{r}_i, \mathbf{x}) \prod_{s=0}^{d-1} \left(\sum_{j=0}^{k-1} L_j(r) f_j^{(s)}(\mathbf{x})\right).$$

The degree of $g_{\mathbf{r}_i}(r)$ is at most $(d-1)(k-1)$. From the expression of $g_{\mathbf{r}_i}(r)$, if we can obtain $g_{\mathbf{r}_0}, \ldots, g_{\mathbf{r}_{k-1}}$, then we can derive the expression of $F(\mathbf{s}, r)$. We next evaluate these polynomials over $[0, (k-1)(d-1)]$.

1. Define the polynomial components:

$$\phi_{\mathbf{x}}^{(j)}(r) = \sum_{i=0}^{k-1} L_i(r) f_i^{(j)}(\mathbf{x}), \quad \phi_{\mathbf{x}}(r) = \prod_{j=0}^{d-1} \phi_{\mathbf{x}}^{(j)}(r).$$

Here $\deg_r(\phi_{\mathbf{x}}^{(j)}) \leq k-1$. The evaluations $\phi_{\mathbf{x}}^{(j)}(i) = f_i^{(j)}(\mathbf{x})$ for $i \in \{0, \ldots, k-1\}$ allow the computation of $\phi_{\mathbf{x}}(r)$ over $[0, d(k-1)]$ using:

- $O(k^2 d^2 N)$ additions via forward differences
- $(d(k-1)+1)(d-1)N$ multiplications for $\phi_{\mathbf{x}}(r)$

2. For each $\mathbf{r}_i$,

$$g_{\mathbf{r}_i}(r) = \sum_{\mathbf{x} \in \{0,1\}^l} eq(\mathbf{r}_i, \mathbf{x}) \phi_{\mathbf{x}}(r).$$

Computing $g_{\mathbf{r}_0}, \ldots, g_{\mathbf{r}_{k-1}}$ requires $O(k^2 dN)$ multiplications and additions.

The prover then performs linear combinations on the polynomials $f_i^{(j)}$ and computes E_i, which costs $k^2(d+1)N$ multiplications and additions.

Therefore, before executing the packed sumcheck, the total complexity is bounded by $O((kd^2+k^2d)N)\cdot\mathsf{mul}+O(k^2d^2N)\cdot\mathsf{add}$, which matches the complexity of the packed sumcheck itself. Hence, the overall prover complexity for ZeroCheck is

$$O((kd^2 + k^2 d)N) \cdot \mathsf{mul} + O(k^2 d^2 N) \cdot \mathsf{add}.$$

□

C Supplementary Proofs for the Vector Relation

C.1 Vector NTT IOP Soundness

Lemma 14 (Lemma 6). *The Vector NTT IOP achieves perfect completeness and has soundness error* $\frac{1}{p^k}$.

Proof. Perfect completeness follows directly. We now prove that the soundness error is at most $\frac{1}{p^k}$. For each row index i, define the polynomial

$$p_i(\mathbf{r}) = \left(\sum_{j=0}^{m-1} \mathbf{r}[j] \cdot \mathbf{a}_j \right)[i] - \left(H \left(\sum_{j=0}^{m-1} \mathbf{r}[j] \cdot \mathbf{b}_j \right) \right)[i],$$

where $\mathbf{r}$ is the random challenge vector. If the prover cheats, then there exists at least one index i^* such that p_{i^*} is a nonzero polynomial. Since $\deg(p_{i^*}) \leq 1$, the Schwartz–Zippel lemma implies that for a single random evaluation point, the probability that p_{i^*} vanishes is at most $\frac{1}{p}$. As the protocol samples k independent random points, the probability that all k evaluations vanish is at most $\frac{1}{p^k}$. Hence, the soundness error is bounded by $\frac{1}{p^k}$. □

C.2 Technical Supplement to the Vector Lookup Check

Lemma 15 (Lemma 8). *Assume that the sumcheck protocol has soundness error ϵ_{SUM} and the ZeroCheck protocol has soundness error ϵ_{ZERO}. Then the Vector Lookup IOP achieves perfect completeness and has soundness error at most*

$$s\epsilon_{\mathsf{SUM}} + 2s\epsilon_{\mathsf{ZERO}} + \tfrac{(N+M-1)^s}{p^s}.$$

Proof. Let $LS(X) = \sum_{j=0}^{N-1} \frac{1}{X+\mathbf{a}[j]}$ and $RS(X) = \sum_{e=0}^{M-1} \frac{\mathbf{h}[e]}{X+\mathbf{t}[e]}$. By Lemma 7, if not all elements of $\mathbf{a}$ are contained in the table $\mathbf{t}$, then $LS(X) \neq RS(X)$. Suppose we sample a random point $\alpha \in \mathbb{F}_p$. If $LS(\alpha) = RS(\alpha)$, then we obtain

$$\prod_{z=0}^{N-1}(\alpha+\mathbf{a}[z]) \cdot \prod_{z=0}^{M-1}(\alpha+\mathbf{t}[z]) \cdot LS(\alpha) = \prod_{z=0}^{N-1}(\alpha+\mathbf{a}[z]) \cdot \prod_{z=0}^{M-1}(\alpha+\mathbf{t}[z]) \cdot RS(\alpha).$$

Both $\prod_{z=0}^{N-1}(X+\mathbf{a}[z]) \cdot \prod_{z=0}^{M-1}(X+\mathbf{t}[z]) \cdot LS(X)$ and $\prod_{z=0}^{N-1}(X+\mathbf{a}[z]) \cdot \prod_{z=0}^{M-1}(X+\mathbf{t}[z]) \cdot RS(X)$ are polynomials of degree at most $N+M-1$. By the Schwartz–Zippel lemma, the probability that they agree at a random point is at most $\frac{N+M-1}{p}$. Therefore, when sampling s independent random points, the probability that all evaluations agree is at most $\frac{(N+M-1)^s}{p^s}$.

Since the Vector Lookup IOP invokes s instances of sumcheck and $2s$ instances of ZeroCheck, its overall soundness error is bounded by the union bound:

$$s\epsilon_{\mathsf{SUM}} + 2s\epsilon_{\mathsf{ZERO}} + \tfrac{(N+M-1)^s}{p^s}.$$

□

Definition 10 (l-Vector Lookup Relation over $\mathbb{F}_p$.). *The relation $\mathcal{R}_{LVLTR}$ is defined over tuples*

$$\big((Com(\mathbf{a}_0), \ldots, Com(\mathbf{a}_{l-1}), \mathbf{t}_0, \ldots, \mathbf{t}_{l-1}), (\mathbf{a}_0, \ldots, \mathbf{a}_{l-1}, \mathbf{h})\big),$$

where each $\mathbf{a}_i \in \mathbb{F}_p^N$ and $\mathbf{t}_i \in \mathbb{F}_p^M$. The relation requires that for every index $i \in [0, N-1]$, there exists some $j \in [0, M-1]$ such that

$$(\mathbf{a}_0[i], \ldots, \mathbf{a}_{l-1}[i]) = (\mathbf{t}_0[j], \ldots, \mathbf{t}_{l-1}[j]).$$

Here, $\mathbf{h} \in \mathbb{F}_p^M$ denotes a multiplicity vector certifying that every element of $(\mathbf{a}_0, \ldots, \mathbf{a}_{l-1})$ is contained in the set defined by $(\mathbf{t}_0, \ldots, \mathbf{t}_{l-1})$.

Lemma 16. *All elements of $(\mathbf{a}_0, \cdots, \mathbf{a}_{l-1}) \in \mathbb{F}_p^{l\times N}$ belong to the table $(\mathbf{t}_0, \cdots, \mathbf{t}_{l-1}) \in \mathbb{F}_p^{l\times M}$ if and only if there exists a multiplicity vector $\mathbf{h} \in \mathbb{F}_p^M$ satisfying:*

$$\sum_{j=0}^{N-1} \frac{1}{X + \sum_{i=0}^{l-1}(\mathbf{a}_i[j] \cdot Y^i)} = \sum_{e=0}^{M-1} \frac{\mathbf{h}[e]}{X + \sum_{i=0}^{l-1}(\mathbf{t}_i[e] \cdot Y^i)}.$$

Proof. We define the polynomials

$$f_j(X,Y) = X + \sum_{i=0}^{l-1} \mathbf{a}_i[j] \cdot Y^i, \qquad g_j(X,Y) = X + \sum_{i=0}^{l-1} \mathbf{t}_i[j] \cdot Y^i.$$

Let

$$LS(X,Y) = \sum_{j=0}^{N-1} \frac{1}{f_j(X,Y)}, \qquad RS(X,Y) = \sum_{j=0}^{M-1} \frac{\mathbf{h}[j]}{g_j(X,Y)}.$$

If the l-Vector Lookup Relation holds, then each $f_i(X,Y)$ must appear in the set $\{g_j(X,Y)\}_{j\in[0,M-1]}$. In this case, $\mathbf{h}[j]$ denotes the multiplicity with which $g_j(X,Y)$ occurs among $\{f_i\}_{i\in[0,N-1]}$, and it follows immediately that $LS(X,Y) = RS(X,Y)$.

Conversely, suppose there exists a vector $\mathbf{h}$ such that $LS(X,Y) = RS(X,Y)$. Then

$$\sum_{j=0}^{N-1} \frac{1}{f_j(X,Y)} - \sum_{j=0}^{M-1} \frac{\mathbf{h}[j]}{g_j(X,Y)} = 0.$$

If some $f_i(X,Y)$ does not belong to $\{g_j(X,Y)\}_{j\in[0,M-1]}$, consider

$$S(X,Y) = \left(\prod_{j=0}^{N-1} f_j(X,Y)\right)\left(\prod_{j=0}^{M-1} g_j(X,Y)\right)\big(LS(X,Y) - RS(X,Y)\big).$$

In this case, $S(X,Y)$ contains a nonzero term that cannot cancel out. Since each f_j and g_j has degree at most $(l-1)$, the degree of $S(X,Y)$ is bounded by $(l-1)(N+M-1) < p$. Hence $S(X,Y)$ is not the zero polynomial over $\mathbb{F}_p$, contradicting the assumption. Therefore, if $LS(X,Y) = RS(X,Y)$ holds for some $\mathbf{h}$, the L-Vector Lookup Relation must also hold.

l-Vector Lookup IOP Construction.

1. The verifier samples s random challenges $(r_0,\chi_0),\cdots,(r_{s-1},\chi_{s-1}) \in \mathbb{F}_p$ and sends them to the prover.
2. For each $i \in [0,s-1]$, the prover defines two new vectors:
   - $\mathbf{f}_i \in \mathbb{F}_p^N$, where $\mathbf{f}_i[j] = \frac{1}{r_i + \sum_{v=0}^{l-1} \mathbf{a}_v[j]\cdot\chi_i^v}$ for $j \in [0,N-1]$.
   - $\mathbf{g}_i \in \mathbb{F}_p^M$, where $\mathbf{g}_i[e] = \frac{\mathbf{h}_i[e]}{r_i + \sum_{v=0}^{l-1} \mathbf{t}_v[e]\cdot\chi_i^v}$ for $e \in [0,M-1]$.

   Using the vector length parameter N_c, the prover partitions and commits to $\{\mathbf{f}_i\}_{i\in[0,s-1]}$ and $\{\mathbf{g}_i\}_{i\in[0,s-1]}$.
3. The parties invoke a packed sumcheck protocol to prove that for each $i \in [0,s-1]$:
$$\sum_{j=0}^{N-1} \mathbf{f}_i[j] = \sum_{e=0}^{M-1} \mathbf{g}_i[e].$$
4. The parties invoke a packed ZeroCheck protocol to prove the following for each $i \in [0,s-1]$:
   - $\mathbf{f}_i[j] \cdot (r_i + \sum_{v=0}^{l-1} \mathbf{a}_v[j]\cdot\chi_i^v) = 1$ for all $j \in [0,N-1]$.
   - $\mathbf{g}_i[e] \cdot (r_i + \sum_{v=0}^{l-1} \mathbf{t}_v[e]\cdot\chi_i^v) = \mathbf{h}[e]$ for all $e \in [0,M-1]$.

D Supplementary Proofs for the TFHE-Friendly SNARK

D.1 Supplementary Proofs for the Verifiable Modulus Switching

Lemma 17 (Lemma 9). *Suppose* $2N_R \mid (q-1)$ *and let* $q = t \cdot (2N_R) + 1$. *For any* $\alpha \in \mathbb{Z}_q \setminus \{0\}$, *there exists a unique decomposition* $\alpha = \beta \cdot (2N_R) + \gamma$, *where* $\beta \in [0, 2N_R - 1]$ *and* $\gamma \in [1, t]$, *such that*

$$\left\lfloor \frac{\alpha \cdot (2N_R)}{q} \right\rceil = \beta.$$

Proof. We compute:

$$\begin{aligned} \left\lfloor \frac{\alpha \cdot (2N_R)}{q} \right\rceil &= \left\lfloor \frac{(\beta \cdot t + \gamma) \cdot 2N_R}{q} \right\rceil = \left\lfloor \frac{(\beta \cdot \frac{q-1}{2N_R} + \gamma) \cdot 2N_R}{q} \right\rceil \\ &= \left\lfloor \frac{\beta(q-1) + \gamma \cdot 2N_R}{q} \right\rceil = \beta + \left\lfloor \frac{\gamma \cdot 2N_R - \beta}{q} \right\rceil. \end{aligned}$$

Since $0 \le \beta \le 2N_R - 1$ and $\gamma \ge 1$, we have $(\gamma \cdot 2N_R - \beta) > 0$. Moreover, as $\gamma \le t = \frac{q-1}{2N_R}$, it follows that $(\gamma \cdot 2N_R - \beta) < q$. Hence, $\lfloor \frac{\gamma \cdot 2N_R - \beta}{q} \rceil = 0$, which yields $\lfloor \frac{\alpha \cdot (2N_R)}{q} \rceil = \beta$.

Lemma 18 (Lemma 10). *Let* $\mathbf{id}, \mathbf{r}, \mathbf{c}' \in \mathbb{Z}_q^{n+1}$ *such that every entry of* $\mathbf{id}$ *lies in* $\{0,1\}$, *every entry of* $\mathbf{r}$ *lies in* $[1,t]$, *and every entry of* $\mathbf{c}'$ *lies in* $[0, 2N_R - 1]$. *If*

$$\mathbf{c} = (t \cdot \mathbf{c}' + \mathbf{r}) \circ \mathbf{id},$$

then $\mathbf{c}'$ *is exactly the result of applying modulus switching to* $\mathbf{c}$.

Proof. Equivalently, we show that if $c = c' \cdot t + r \cdot id \pmod q$ with $c' \in [0, 2N_R - 1]$, $r \in [1,t]$, and $id \in \{0,1\}$, then $c' = \lfloor \frac{c \cdot (2N_R)}{q} \rceil$.

If $c \equiv 0 \pmod q$, then $c' = 0$, which is consistent with the formula. Otherwise, note that $0 < c' \cdot t + r \cdot id \le (2N_R - 1) \cdot t + t = q - 1 < q$, which rules out the case $c \equiv 0 \pmod q$.

For $c \not\equiv 0 \pmod q$, Lemma 9 guarantees that c admits a unique decomposition $c = c' \cdot t + \gamma$ with $c' \in [0, 2N_R - 1]$ and $\gamma \in [1,t]$. Thus,$c' = \left\lfloor \frac{c \cdot (2N_R)}{q} \right\rceil$. □

E Sumcheck Performance on Intel CPU

Table 4: Sumcheck Benchmark on Intel i9-13900H

Method	K	d	λ	Time (ms) at $l = \{14, 16, 18, 20\}$			
Ours	4	2	109	1.14 (4.01×)	4.54 (4.03×)	18.92 (3.88×)	79.25 (3.69×)
	4	3	106	2.19 (3.62×)	8.88 (3.63×)	37.59 (3.44×)	159.98 (3.25×)
	4	4	105	3.57 (3.48×)	14.47 (3.49×)	59.58 (3.44×)	251.38 (3.33×)
	4	5	103	5.18 (3.49×)	21.25 (3.49×)	85.90 (3.47×)	359.98 (3.34×)
	5	2	134	1.40 (3.05×)	5.60 (3.2×)	24.10 (3.09×)	98.60 (3.03×)
	5	3	131	2.49 (2.98×)	10.27 (3.07×)	43.43 (3.01×)	173.61 (2.98×)
	5	4	129	3.95 (2.97×)	15.84 (3.06×)	67.15 (2.93×)	273.20 (2.89×)
	5	5	128	5.69 (2.96×)	24.37 (2.82×)	96.73 (2.90×)	392.42 (2.88×)
[BDT24][5] (Alg. 1)	4	2	123	4.27	18.01	74.47	298.33
	4	3	122	7.42	31.55	130.82	517.47
	4	4	121	11.75	48.44	196.91	789.04
	4	5	121	16.86	68.68	280.91	1129.5