Batch, Pack, and Prove: More Efficient Verifiable Computation for CKKS

Ignacio Cascudo[1], Anamaria Costache[2], Daniele Cozzo[1], Dario Fiore[1], Antonio Guimarães[1], and Eduardo Soria-Vazquez[3]

[1] IMDEA Software Institute, Madrid, Spain.
firstname.lastname@imdea.org
[2] École Polytechnique, Paris, France.
lastname@lix.polytechnique.fr
[3] Technology Innovation Institute, Abu Dhabi, UAE.
firstname.lastname@tii.ae

Abstract. An important challenge in homomorphic encryption (HE) is ensuring the integrity of the encrypted computations. Although verifiable computation can address this in theory, making it practical is far from trivial. [Cascudo et al. CRYPTO 2025] recently proposed a solution to this problem for the CKKS scheme. In particular, they propose new succinct arguments that efficiently prove arithmetic circuits and range checks over the ring $R_q = \mathbb{Z}_q[Y]/(Y^N + 1)$, and combine them to prove computations on CKKS ciphertexts. Despite the important progress, their work faces scalability challenges: the size of the polynomial commitment proofs, a large $\mathcal{O}(N)$ overhead in range proofs, and the lack of an explicit method for efficient—sublinear in circuit size—verification.
In this paper we present new techniques that remove all these bottlenecks. First, we provide a more efficient arithmetization of CKKS computations. Second, we introduce BatchFold, a new polynomial commitment for R_q that supports batched evaluations and achieves shorter proofs. Third, we design packing, a novel algebraic technique that eliminates the need to commit to massive polynomials and removes the $\mathcal{O}(N)$ overhead in range proofs. Finally, we introduce SparsePack, an efficient protocol for proving the evaluations of sparse, circuit-dependent polynomials, leveraging both packing and batched evaluations. We evaluate the efficiency of our solution on a generic neural network architecture, achieving proofs several orders of magnitude smaller and faster than prior approaches.

Keywords: Verifiable Computation, Zero-knowledge proofs, FHE, homomorphic inference

Table of Contents

1 Introduction 4
2 Preliminaries 8
2.1 Background on Rings 8
2.2 Polynomial Interactive Oracle Proofs for Rings 9
2.3 Proof-friendly CKKS and its arithmetization 10
3 PIOPs for Unpacked Polynomials 12
3.1 Packing and Unpacking polynomials in R_q 12
3.2 PIOP for the Evaluation of Unpacked Polynomials on R_q points . 14
4 Our PIOP for Range Checks over Packed Polynomials 15
5 Improved Arithmetization and PIOP Π_{AC} for CKKS 18
5.1 Layer consistency equations 21
5.2 Progressing through the layers 23
5.3 Prover algorithms 24
6 Our PIOP for Evaluation of Sparse Polynomials 24
6.1 Overview of the SparsePack PIOP 25
7 BatchFold 27
7.1 Batching Openings via CRT and HyperPlonk 28
7.2 Batching BaseFold openings at a common point 28
8 Implementation and Evaluation 30
8.1 BatchFold 30
8.2 High-level applications 31
A Additional Preliminaries 38
A.1 Polynomial Commitments with Batched Evaluations 38
A.2 Compiling PIOPs into SNARKs 39
A.3 Round-by-Round Soundness 40
A.4 The Sumcheck Protocol 41
A.5 CKKS 43
A.6 A modular construction of the PIOP Π_{CKKS} 44
B Auxiliary Material on PIOP for Unpacked Polynomials 44
B.1 Proof of Theorem 1 44
C Auxiliary Material on PIOP for Range Checks 44
C.1 Proof of Lemma 4 44
C.2 Proof of Theorem 2 45
D Our Instantiation of the PIOP $\hat{\Pi}_{range}$ 47
D.1 Correctness of $\hat{\Pi}_{range}$ 48
D.2 Security of $\hat{\Pi}_{range}$ 50
D.3 Efficiency of $\hat{\Pi}_{range}$ 53
D.4 A PIOP for Range Checks over a Finite Field 54
D.5 Our Optimized PIOP $\hat{\Pi}^{*}_{range}$ 59
E Additional material for Section 5 67
E.1 The `copycon` wiring predicate in Equation (7) 67

E.2 Wiring predicates from [CCC+25] 67
E.3 Too many gates in Equation (7)? 68
E.4 Additional material for Section 5.2 69
E.5 Prover algorithms 70
F Putting everything together: The Π_{CKKS} PIOP 76
F.1 Rearranging into the final oracles and final queries 76
G Auxiliary Material on SparsePack 77
G.1 Our SparsePack PIOP for Evaluating Sparse Polynomials 77
G.2 Instantiations of SparsePack and Efficiency 79
G.3 Details on our instantiation of Π_{iptree} 80
H Details about BaseFold 81
H.1 The BaseFold Error Correcting Codes 81
H.2 The BaseFold Commitment and Evaluation 81
H.3 Improvements on the minimum distance estimates of the BaseFold codes 85
I Implementation and Evaluation additional details 90
I.1 Neural Networks 90
I.2 Additional experimental results 91

1 Introduction

Homomorphic encryption (HE) [Gen09] allows to compute directly over encrypted data, a crucial feature for data privacy in numerous applications. A notable use case is secure outsourcing of computation, allowing clients to delegate computations to untrusted servers, without revealing their inputs. For instance, in the context of Machine Learning as a Service (MLaaS), users can send encrypted queries to a server holding a model and receive encrypted results (e.g., predictions or classifications) in return.

However, *HE alone does not guarantee the integrity of the computation.* While data remains private, clients have no assurance that the encrypted results returned by the server are correct. This creates a dangerous gap in HE-based solutions that leaves users vulnerable to maliciously manipulated outputs. In fact, in certain deployment scenarios, it has been shown that malicious use of the malleability feature of HE can lead to key-recovery attacks [LM21,GNSJ24,CCP+24]. Tools to enforce the integrity of the computation can be crucial in providing solutions that counter such attacks [ABMP24].

Consequently, a line of research (see below for details on related works) addresses this problem by adding integrity properties to HE using another fundamental primitive: succinct non-interactive arguments of knowledge (SNARKs). SNARKs allow to prove the correctness of a computation in a way that can be efficiently checked. Thus one can use a SNARK to prove a homomorphic computation over HE ciphertexts, yielding what is called *verifiable HE* (vHE).

Despite the tremendous progress in SNARKs research that brought them to a practical level that currently enables numerous applications, their use for vHE is not efficient in practice. This is because HE and SNARKs typically work over algebraic structures that are incompatible with each other. SNARKs work over large finite fields and are very efficient in proving arithmetic statements in this context, whereas HE works over polynomial rings with composite moduli, e.g., $R_q = \mathbb{Z}_q[Y]/(Y^N + 1)$, where q is a product of primes, and critically uses highly non-arithmetic operations for noise management. On top of that, due to the large size of HE ciphertexts (a single R_q element can be 100-600KB), the actual homomorphic computation to be proven is much larger than its description over plaintexts. This situation makes general-purpose SNARKs non-ideal candidates for vHE due to overheads originating from emulation of ring arithmetic, non-algebraic noise-management operations, and the size of ciphertexts.

A recent work by Cascudo et al. [CCC+25] took an important step toward reconciling the aforementioned arithmetic incompatibilities by introducing a new framework for building SNARKs for verifiable HE, focusing on the CKKS scheme [CKKS17]. Their approach begins with a proof-friendly version of CKKS whose performance is only slightly degraded compared to the original scheme, and whose homomorphic computations can be described with just two relations over R_q: arithmetic circuit satisfiability and range checks. They then design a SNARK for these relations, by combining polynomial interactive oracle proofs (PIOPs) for them and a suitable polynomial commitment (PC). A key feature of their framework is that their PIOPs work over the ring R_q, not the usual prime fields,

and natively supports the HE arithmetic without emulation overheads. This approach shows promise, and [CCC+25] provides initial evidence of efficiency gains over general-purpose SNARKs. Their evaluation fully confirms the practicality of the proof-friendly CKKS HE, but targets only two of the proving building blocks (sumcheck over R_q and a variant of the Brakedown [GLS+23] polynomial commitment) and indicates application-level efficiency estimates only for a small example consisting of *one* CKKS multiplication followed by noise flooding.

In this work, we continue this line of research on building verifiable computation (VC) for CKKS with the aim of achieving solutions that can, for the first time, scale to realistic HE applications. Our starting point is the recent work of Cascudo et al. [CCC+25]. Despite the fact that their work represents significant progress over the state-of-the-art, several limitations remain that prevent their approach from scaling in practice. Most of them stem from the fact that [CCC+25] designed their framework to be efficient with respect to circuit size, but did not fully address the fact that, by themselves, HE parameters can be very large and affect concrete performance, even if we view them as constants (for a fixed security level). In more detail:

- For a circuit with $\mathcal{O}(n)$ HE operations on an RLWE scheme over the ring $R_q = \mathbb{Z}_q[Y]/(Y^N+1)$, their polynomial commitment has $\mathcal{O}(\sqrt{n}N\log(q))$-sized proofs and verification time. Although asymptotically efficient (sublinear in n), the $N\log(q)$ term already is in the order of hundreds of kilobytes – even for very small CKKS parameters. As the circuit grows, this cost quickly becomes impractical, effectively limiting [CCC+25] to tiny HE circuits.
- In dealing with noise maintenance operations, their protocol for range checks yields an $\mathcal{O}(N)$ overhead to be paid by the polynomial commitment. Crucially, this overhead comes *on top* of the cost for committing to vectors of ciphertexts that inherently have their own $\mathcal{O}(N)$ size. Considering that N is typically around 2^{13} or larger, this overhead is significant. For instance, to prove that a vector of, say, $n = 2^{10}$ elements satisfies the range check, they need to additionally commit to at least four vectors of length 2^{23}.
- Their protocol for arithmetic circuits is based on a custom version of GKR [GKR08], and the main contribution of [CCC+25] is to efficiently integrate the noise-management operations within the protocol. As typical in GKR, verifying for a circuit C either requires the verifier to do $\mathcal{O}(|C|)$ work to evaluate some polynomials that describe the circuit, or store commitments to these polynomials, and delegate their evaluation to the prover. [CCC+25] does not address the concrete efficiency of this component. While this could be solved in theory with standard techniques, these – when adapted for a large ring like R_q – would be prohibitively expensive. For instance, an approach similar to Spartan [Set20] would imply proofs in the order of GBs (see Section 8).

Our contributions. To achieve practically efficient verifiable HE for non-trivial circuits, we extend the framework of Cascudo et al. [CCC+25] with new techniques that overcome its main limitations. Our approach yields substantial effi-

ciency improvements on generic neural networks, demonstrating the feasibility of verifiable HE for MLaaS applications.

BatchFold. To reduce the $\mathcal{O}(\sqrt{n})$ proof size of Brakedown commitments used in [CCC+25], we build on BaseFold [ZCF24], which achieves $\mathcal{O}(\log^2 n)$ proofs and $\mathcal{O}(n \log n)$ proving time. Our main contribution (see Section 7) is a novel *batching technique*. For M independently committed polynomials f_i with n coefficients over a finite field, we can prove multiple evaluations $(y_i = f_i(x_i))$, reducing proof size and proving time from $O(M \log^2 n)$, using naive repetition, to $O(M \log n + \log^2 n)$. Our batching technique has a twofold impact. The first obvious one is that we can batch evaluation proofs when compiling the PIOP into a SNARK. The second advantage is improving the complexity of our R_q version of BaseFold *even for evaluating a single polynomial*. In a nutshell, a PC for R_q is built by exploiting the CRT splitting of R_q into a product of fields. Crucially, we use the fact that, for each prime factor p of q, there are $N/4$ isomorphic fields of characteristic p in the splitting, and thus we can apply batching over the $N/4$ polynomials in the CRT decomposition of the input polynomial. For a q, product of $L+1$ different primes, we have a proof size of $O(NL \log n + L \log^2 n)$ field elements (from fields of similar sizes) instead of $O(NL \log^2 n)$.

Packing and Unpacking in PIOPs. The PIOP for range checks of [CCC+25] proves that each evaluation $\tilde{v}(\boldsymbol{b})$ in the boolean hypercube of a multilinear polynomial $\tilde{v} \in R_q[\boldsymbol{X}]$ is an element of $R_q = \mathbb{Z}_q[Y]/(Y^N + 1)$ with coefficients $< B$. To do this, [CCC+25] introduces a decomposition technique that, $\forall \boldsymbol{b} \in \{0,1\}^\ell$, embeds the N coefficients of $\tilde{v}(\boldsymbol{b})$ into the evaluations $\tilde{h}(\boldsymbol{b}, \boldsymbol{k})$ of a freshly generated $\tilde{h} \in R_q[\boldsymbol{X}, \boldsymbol{Z}]$ of $\ell + \log N$ variables. This approach incurs an $\mathcal{O}(N)$ overhead, especially for committing to $\tilde{h}$, an object N times bigger than $\tilde{v}$.[4]

In our paper, we introduce *packing*, a novel algebraic technique that allows PIOPs to prove statements about "unpacked" versions of multilinear polynomials. Analogously to the above decomposition, the unpacking of a polynomial $\tilde{p} \in R_q[\boldsymbol{X}]$ is a polynomial $\tilde{u} \in R_q[\boldsymbol{X}, \boldsymbol{Z}]$ with $\log N$ more variables and whose evaluations are all in $\mathbb{Z}_q$. The crucial difference though is that: (i) *we do not commit to* $\tilde{u}$, we only store and commit to $\tilde{p}$ in its dense (aka packed) form, (ii) the PIOP can work with an efficient sparse representation of $\tilde{u}$ that has no $\mathcal{O}(N)$ artificial overhead, and (iii) queries to $\tilde{u}$ can be efficiently computed from queries to $\tilde{p}$. For this reason, we call $\tilde{u}$ a virtual unpacked oracle. We refer to Section 3 for details on the technique. We discuss here two main applications.

The first one is a new PIOP for range checks that avoids the $\mathcal{O}(N)$ overhead of [CCC+25], achieving not only an asymptotic improvement but also concrete orders of magnitude faster computing time. Specifically for the largest circuit of our experimental benchmark, we estimate [CCC+25]'s range checks to take 1h20m in 48 threads, while ours takes less than 6 minutes single-threaded.

Beyond range checks, we use packing to compress sparse polynomials over R_q, such as the multilinear extensions of $\mathbb{Z}_q$-vectors in GKR. In particular, we

[4] This decomposition and its overhead is reminiscent of the binary decomposition that creates additional (witness) bits and embeds them in a finite field.

use packing to construct SparsePack, a new protocol for the evaluation of sparse polynomials, i.e., having sublinear nonzero evaluations in the hypercube. When applied to our case (MLEs of wiring predicates), it outperforms state-of-the-art solutions [Set20] featuring proofs that, in one of our benchmarks, are 41× shorter and 10× faster to prove (see Section 8).

Better arithmetization and preprocessing. One bottleneck of [CCC+25] is handling the polynomials that describe the GKR circuit. These are as large as the circuit and hence cannot be computed by the verifier. Delegating this work to the prover is costly since, for a circuit of size 2^ℓ, these polynomials are of size $2^{3\ell}$, despite having only $\mathcal{O}(2^\ell)$ nonzero evaluations. To address this, we introduce a new arithmetization for CKKS circuits that improves upon [CCC+25] in several ways. We design closed-form polynomials for the wiring predicates related to CKKS homomorphic multiplication which the verifier can compute efficiently. We also substantially reduce the number and size of the polynomials that encode the structure of the target FHE circuit (which is arbitrary). Finally, we use SparsePack to prove the evaluations of these sparse polynomial (see above).

Experiments. The framework of [CCC+25] enabled, for the first time and with minimal emulation overhead, verifiability for complete HE multiplications with non-linear noise management operations. However, the authors only presented estimates for a single ciphertext multiplication. Our improvements can scale beyond that: we provide experiments for small Neural Networks of multiplicative depth 2, containing between 48 and 192 total ciphertext multiplications each. We measure performance for our new packed range proofs and batched polynomial commitment scheme, and estimate it for our sparse commitments, showing that these NNs can be proven in the order of minutes single-threaded. This represents a major improvement over [CCC+25], which estimated 25 seconds for a single multiplication in 48 threads. Additionally, we present execution time and proof size for commitments over R_q using Batchfold, showing a factor of 2.8 (resp. 9.5) improvement in execution time (resp. proof size) compared to Basefold.

Related work. Literature on verifiable computation for FHE can be divided in two main approaches. The first one, formalized in [FGP14], consists in designing VC schemes for proving computations on FHE ciphertexts. With the exception of recent work of Cascudo et al. [CCC+25], prior works in this line suffer from either supporting only variants of HE schemes with limited applicability [FGP14,FNP20,BCFK21], or otherwise incurring large overheads for emulating ring or noise-maintenance operations [GNS23,KVH24,VKH23]. The second approach computes proofs homomorphically over the plaintext space [GGW24,ACGS23,GBK+24]. While this approach can be faster than proving ciphertext computations, it has inherent limitations, notably: it cannot achieve public verifiability, since proof verification requires decryption[5]; it cannot sup-

[5] Public verifiability, by definition, cannot rely on secret material. Nevertheless, any designated verifier scheme can be turned into a "semi-publicly verifiable" one by making the verifier provide a publicly verifiable ZK proof of their execution of the verification algorithm. This idea is instantiated with proofs of decryption in [GBK+24].

port approximate schemes such as CKKS (due to their lack of a ring structure on the plaintext space) and it is vulnerable to verification oracle attacks.

2 Preliminaries

For a positive integer n, $[n]$ denotes the set $\{0, \ldots, n-1\}$. We denote vectors $\mathbf{v}$ in bold, start indexing at 0 and denote its i-th coefficient by $\mathbf{v}[i]$ or v_i. We denote by $\mathbf{v}_{[B]}$ the truncation of $\mathbf{v}$ to its first B terms, $\mathbf{v}_{[A,B)}$ for its terms in the range $[A, B)$ (i.e. $\mathbf{v}_{[A,B)} = (v_A, \ldots, v_{B-1})$) and $\mathbf{v}_{[A,B]}$ for its terms in the range $[A, B]$. We define $|\mathbf{v}|$ the length of $\mathbf{v}$ and $\#(\mathbf{v}) = \lceil \log_2(|\mathbf{v}|) \rceil$. Let $\mathsf{to\text{-}int} : \{0,1\}^* \to \mathbb{Z}$ be the function that, on input a binary vector $\boldsymbol{j} \in \{0,1\}^n$, for any positive integer n, returns the integer $\sum_{k \in [n]} j_k \cdot 2^k$. Let $\mathsf{bin} : \mathbb{Z} \to \{0,1\}^*$ be its inverse. Given a finite set S, we write $a \leftarrow\!\$\, S$ to mean that a is sampled uniformly at random from S and $e \leftarrow \chi$ to mean that e is sampled according to some distribution χ.

We often invoke the infinity norm $\|\cdot\|_\infty$. Given a vector $\mathbf{v}$ of elements in $R_q := \mathbb{Z}_q[Y]/(Y^N + 1)$, we apply the norm coefficient-wise: the expression $\|\mathbf{v}\|_\infty < B$ means that all the monomial coefficients of $\mathbf{v}$ are within the range $[0, B-1]$.

An *indexed relation* $\mathcal{R}$ is a set of tuples $(\mathbb{i}, \mathbb{x}; \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ the statement and $\mathbb{w}$ the witness. The language $\mathcal{L}(\mathcal{R})$ associated to a relation $\mathcal{R}$ is the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}$.

2.1 Background on Rings

We restrict ourselves to finite commutative rings with unity, which we often denote by R. Given $a \in R$, $[a]_b$ denotes the reduction of a modulo b. If $a \in R(Y)$ is an element of a polynomial ring, $a[j]$ (resp. $a[\boldsymbol{j}]$) denotes its j-th (resp. $\mathsf{to\text{-}int}(\boldsymbol{j})$-th) coefficient (in such representation) and $[a]_b$ denotes the coefficient-wise modular reduction. For a vector $\mathbf{v} \in R^n$, $[\mathbf{v}]_p$ denotes modular reduction of each component of $\mathbf{v}$.

Definition 1. *An* exceptional set *is a subset $\hat{S} \subset R$ with the property that $a - b$ is an invertible element of R for each pair of distinct elements $a, b \in \hat{S}$.*

We denote by $R[X_1, \ldots, X_\ell]$ the polynomials in ℓ variables and coefficients in R and by $R^{(\leq d)}[X_1, \ldots, X_\ell]$ the subset of polynomials of individual degree $\leq d$.

Lemma 1 (Generalized Schwartz-Zippel). *Let $f \in R[X_1, \ldots, X_\ell]$ be a non-zero polynomial of total degree d. Let $\hat{S} \subset R$ be an exceptional set. Then,* $\Pr_{(r_1,\ldots,r_\ell) \leftarrow \hat{S}^\ell}[f(r_1, \ldots, r_\ell) = 0] \leq \frac{d}{|\hat{S}|}$.

The *multilinear extension* (MLE) of a function $f : \{0,1\}^\ell \to R$ is the (unique) multilinear polynomial $\tilde{f} : R^\ell \to R$ such that $\tilde{f}(\boldsymbol{b}) = f(\boldsymbol{b})$ for all $\boldsymbol{b} \in \{0,1\}^\ell$. For any $\ell \geq 1$, let $eq : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}$ be the equality predicate (i.e., $eq(\boldsymbol{a}, \boldsymbol{b}) = 1$ iff $\boldsymbol{a} = \boldsymbol{b}$) and let $\tilde{\mathsf{eq}}$ be its MLE, which has closed-form representation $\tilde{\mathsf{eq}}(\boldsymbol{X}, \boldsymbol{Y}) = \prod_{j=1}^{\ell}(X_j Y_j + (1 - X_j)(1 - Y_j))$ computable in time $\mathcal{O}(\ell)$ [Tha13]. The MLE of any function $f : \{0,1\}^\ell \to R$ is defined by the

polynomial $\tilde{f}(X_1, \ldots, X_\ell) = \sum_{\boldsymbol{b} \in \{0,1\}^\ell} f(\boldsymbol{b}) \cdot \tilde{\mathsf{eq}}(\boldsymbol{b}, X_1, \ldots, X_\ell)$. Analogously, the MLE of a vector $\boldsymbol{v} \in R^{2^\ell}$ (which for convenience we index in binary) is the polynomial $\tilde{v}(X_1, \ldots, X_\ell) = \sum_{\boldsymbol{b} \in \{0,1\}^\ell} \boldsymbol{v}[\boldsymbol{b}] \cdot \tilde{\mathsf{eq}}(\boldsymbol{b}, X_1, \ldots, X_\ell)$.

2.2 Polynomial Interactive Oracle Proofs for Rings

We recall polynomial interactive oracle proofs (PIOP) for rings [CCC+25] using the notion of (indexed) oracle relations of [CBBZ23].

Definition 2 (Oracle Relations). *An indexed oracle relation $\mathcal{R}$ is an indexed relation in which the index $\mathtt{i}$ and the statement $\mathtt{x}$ include pointers to oracle polynomials with coefficients in a ring R, and the polynomials are part of the witness $\mathtt{w}$. A pointer to a polynomial p is denoted by $[\![p]\!]$.*

Definition 3 (Polynomial IOPs). *Let R be a ring and $\mathcal{R}$ be an indexed oracle relation for (multivariate) polynomials with coefficients in R. A polynomial IOP for $\mathcal{R}$ is an interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, compactly denoted by $\langle \mathcal{P}(\mathtt{i}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathtt{i}, \mathtt{x}) \rangle$. In each round the prover $\mathcal{P}(\mathtt{i}, \mathtt{x}, \mathtt{w})$ sends oracle polynomials and/ or ring elements, and the verifier $\mathcal{V}(\mathtt{i}, \mathtt{x})$ sends random challenges. Each oracle $[\![p]\!]$ specifies the number of variables ℓ and the degree in each variable. The verifier can query all the oracles, the ones sent by the prover and the ones included in $(\mathtt{i}, \mathtt{x})$, at arbitrary points $\boldsymbol{r} \in R^\ell$. At the end of the execution, $\mathcal{V}$ accepts or rejects. See Appendix A for the efficiency metrics.*

Correctness: For any honest execution of $\langle \mathcal{P}(\mathtt{i}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathtt{i}, \mathtt{x}) \rangle$ on $(\mathtt{i}, \mathtt{x}; \mathtt{w}) \in \mathcal{R}$, the verifier accepts.

δ-Soundness: A PIOP is δ-sound if for any unbounded adversary Adv and any $(\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R})$,

$$\Pr\left[\langle \mathsf{Adv}(\mathtt{i}, \mathtt{x}), \mathcal{V}(\mathtt{i}, \mathtt{x}) \rangle = 1\right] \leq \delta$$

δ-Knowledge Soundness: A PIOP is δ-knowledge-sound if there exists a PPT oracle machine Ext (which can query all the oracle polynomials at arbitrary points) such that for any adversary Adv and any $(\mathtt{i}, \mathtt{x})$

$$\Pr\left[b = 1 \ \wedge \ (\mathtt{i}, \mathtt{x}; \mathtt{w}) \notin \mathcal{R} : \langle \mathsf{Adv}(\mathtt{i}, \mathtt{x}), \mathcal{V}(\mathtt{i}, \mathtt{x}) \rangle = b, \ \ \mathtt{w} \leftarrow \mathsf{Ext}^{\mathsf{Adv}}(\mathtt{i}, \mathtt{x})\right] \leq \delta.$$

PIOPs can also satisfy the notion of *round-by-round (RBR) soundness* [CCH+19], which allows a tighter quantification of the loss incurred when applying the Fiat-Shamir transformation, and a better analysis of protocols with parallel repetition (see Appendix A.3 for more details).

Lemma 2 ([CBBZ23, Lemma 2.3]). *Consider a δ-sound PIOP for oracle relations $\mathcal{R}$ such that for all $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$, $\mathtt{w}$ consists only of polynomials such that the statement contains oracles to these polynomials. The PIOP has δ-knowledge-soundness, and the extractor runs in time $\mathcal{O}(|\mathtt{w}|)$.*

We generalize the notion of *virtual oracles* from [CBBZ23].

Definition 4 (Virtual oracles). *Given a function g and oracles $\{[\![p_1]\!], \ldots, [\![p_k]\!]\}$, a virtual oracle $[\![p^*]\!] = [\![g(p_1, \ldots, p_k)]\!]$ consists of the list of oracles and functions $ev_g, h_1, \ldots, h_k$ such that, for any $\boldsymbol{x}$, $ev_g(p_1(h_1(\boldsymbol{x})), \ldots, p_k(h_k(\boldsymbol{x})), \boldsymbol{x}) = p^*(\boldsymbol{x})$.*

In other words, the functions associated to the virtual oracle allow one to compute a query to the virtual oracle $[\![p^*]\!]$ at $\boldsymbol{x}$ by querying each oracle $[\![p_j]\!]$ on some $\boldsymbol{x}_j = h_j(\boldsymbol{x})$ and then computing $p^*(\boldsymbol{x})$ from their results.

Sumcheck protocol. Let R be a ring, $p : R^\ell \to R$ be an ℓ-variate polynomial of degree at most d in each variable. The sumcheck (oracle) relation $\mathcal{R}_{sum}$ is the set of tuples $(y; p)$ (resp. $((y, [\![p]\!]); p)$) such that $y = \sum_{\boldsymbol{b} \in \{0,1\}^\ell} p(\boldsymbol{b})$. The sumcheck protocol [LFKN90] is an interactive proof (resp. a PIOP) for $\mathcal{R}_{sum}$ where the verifier needs to only compute (resp. query) p at a point $(r_1, \ldots, r_\ell)$. While originally defined for finite fields, it also generalizes to rings R if the random challenges are sampled from an exceptional set $\hat{S} \subset R$. We denote the sumcheck PIOP by Π_{sum}; for completeness we present it in Appendix A.4. Also, we denote by Π^0_{sum} a variant of the PIOP Π_{sum} changed so that the verifier samples random challenges in an exceptional set $\hat{S}_0 \subset \mathbb{Z}_q$.

Polynomial commitments and compilation to SNARKs. Prior work shows how to compile PIOPs into SNARKs via polynomial commitments (PC), a cryptographic primitive that allows committing to a polynomial p and later proving that $y = p(a)$ with a succinct proof. In our work, we use an extension of this paradigm that relies on PCs with *batched evaluation proofs*. In a nutshell, this is a generalization of PCs that allow generating a *single* proof about the evaluations of multiple, say M, polynomials at multiple points, i.e., $\forall i \in [M] : y_i = p_i(a_i)$, and has the advantage that the resulting proof size is smaller than concatenating M proofs. For space reasons, see Appendix A.1 for more formal details.

2.3 Proof-friendly CKKS and its arithmetization

In this work we use the proof-friendly version of the CKKS homomorphic encryption scheme [CKKS17] introduced in [CCC+25]. For a full description of the scheme, see Appendix A.5 or [CCC+25]. Here we only discuss the main technical points that are useful to build our proof system.

Choice of the ring and exceptional sets. Fast CKKS implementations use the ciphertext ring $R_q = \mathbb{Z}_q[Y]/(Y^N+1)$, where N is a power of two and $q = \prod_{i=0}^{L} p_i$ for primes p_i. In proof-friendly CKKS, $p_i = 2aN/d + 1$ for an odd integer a and a power of two d. Thus, $Y^N + 1$ factors over $\mathbb{Z}_{p_i}$ into $k = N/d$ irreducible polynomials $Y^d - \zeta_i^{(2j+1)}$, where ζ_i is a primitive $2N/d$-th root of unity. By the CRT, $R_q \cong \prod_{i=0}^{L} \prod_{j=0}^{k-1} R_{i,j}$, with each $R_{i,j} = \mathbb{Z}_{p_i}[Y]/(Y^d - \zeta_i^{(2j+1)}) \cong \mathbb{F}_{p_i^d}$.

This structure yields *exceptional sets* large enough for soundness, bounded by the smallest field component. We use two sets, $\hat{S}_0 \subset \mathbb{Z}_q$ and $\hat{S}_N \subset R_q$, of sizes p_{min} and p_{min}^d respectively (where p_{min} is the smallest p_i). For example, setting $d = 4$ (resp. $d = 2$) for 32-bit (resp. 64-bit) primes makes $p_{min}^{-d} \approx 2^{-128}$, i.e., negligible for security parameter $\lambda = 128$; using $\hat{S}_0$ may instead require

repetitions to amplify soundness. Concretely, we take $\hat{S}_N$ diagonal: each of its elements has the same exceptional element of $\mathbb{F}_{p_i^d}$ in all the k components $R_{i,j}$ of each p_i (Appendix G.2 discusses it further).

Since R_q decomposes into copies of $\mathbb{F}_{p_i^d}$, we cannot use full NTTs to accelerate HE operations and instead use *incomplete NTTs*. We note that efficient implementations of incomplete NTTs have been widely explored in the literature, and can even lead to performance improvements in some cases [HYS+25,AHKS22,LS19]. In [CCC+25], the authors incur a slowdown of just 20% compared to "regular CKKS" due to the use of incomplete NTTs. Since we make use of their scheme unmodified, our construction also inherits their slowdown. We note this modification in the ciphertext ring has no bearing upon the underlying plaintext ring or slots. We adopt the standard plaintext encoding $\mathbb{C}^{N/2} \to \mathbb{C}[X]/(X^N + 1)$ [CKKS17], but one could also use others [JPK+24,Kim25].

Remaining in the same ring R_q. Standard levelled CKKS uses a moduli chain $q := q_L > \cdots > q_0$ (where $q_i = q_{i+1}/p_{i+1}$). In contrast, proof-friendly CKKS operates entirely over a single ring $R_{q_L} = R_q$. Operations normally altering the ring (e.g., modulus switching) have their results re-embedded back into R_{q_L} via the map $\iota_l : R_{q_l} \hookrightarrow R_{q_L}, \quad x \mapsto \tilde{x} \cdot z_l$, where $\tilde{x} \in R_{q_L}$ is a lift of x and $z_l := \sum_{i=0}^{l} Q_i \hat{Q}_i = \mathsf{CRT}(\underbrace{1, \ldots, 1}_{l+1 \text{ times}}, 0, \ldots, 0) \in R_{q_L}$ with $Q_i = q_L/p_i$ and $\hat{Q}_i = \left[(q_L/p_i)^{-1}\right]_{p_i}$ defining the CRT map. As discussed in [GNS23,CCC+25], this abstraction effectively emulates modular reductions and quotient ring arithmetic without impacting performance.

Arithmetization of proof-friendly CKKS. At the core of [CCC+25] is a procedure that translates a computation $\mathbf{C}'(\mathbf{x}) = \mathbf{y}$, where $\mathbf{C}'$ is a circuit of (proof-friendly) CKKS additions and multiplications (see sec.A.5), into a relation over R_q. Ciphertext additions are trivially described by additions over R_q. Other operations such as key switching and rounding (required for ciphertext multiplication) cannot be expressed by arithmetic operations over R_q without significant overhead. For $\mathsf{KeySwitch}$, the prover inputs $w_{\mathrm{ks},0}, \ldots, w_{\mathrm{ks},l} \in R_q$, satisfying $\|w_{\mathrm{ks},i}\|_\infty < p_i$ and such that they recompose to the original value in R_q via the CRT map (which is an arithmetic operation over R_q).

Regarding $\mathsf{Rescale}$, suppose we have an element $x \in R_{q_l}$, in its RNS representation $x = ([x]_{p_0}, [x]_{p_1}, \ldots, [x]_{p_l}, [0]_{p_l+1}, \ldots, [0]_{p_L})$. The division and rounding of x by the current level l can be proved via Euclidean division by p_l. The prover's inputs are $w_{\mathrm{rmd}} = ([w_{\mathrm{rmd}}]_{p_0}, \ldots, [w_{\mathrm{rmd}}]_{p_L})$ and $w_{\mathrm{quot}} = ([w_{\mathrm{quot}}]_{p_0}, \ldots, [w_{\mathrm{quot}}]_{p_L})$. The range proof shows that $\|w_{\mathrm{rmd}}\|_\infty < p_l$ and that $\|w_{\mathrm{quot}}\|_\infty < q_l/p_l$. Finally, recalling that $z_l = \mathsf{CRT}(\underbrace{1, \ldots, 1}_{l+1 \text{ times}}, 0, \ldots, 0) \in R_{q_L}$, we check that $x \cdot z_l = (w_{\mathrm{quot}} \cdot p_l + w_{\mathrm{rmd}}) \cdot z_l$. This shows that proof-friendly CKKS multiplications can be expressed as a sequence of arithmetic operations over R_q and a finite number of range checks relations for R_q elements.

One can apply this argument to a generic circuit $\mathbf{C}'$ organized into D multiplicative layers of width W. [6] Denoting by $\mathbf{w}^{(j)}_{\text{ks},0}, \ldots, \mathbf{w}^{(j)}_{\text{ks},j}, \mathbf{w}^{(j)}_{\text{rmd}}, \mathbf{w}^{(j)}_{\text{quot}}$ the vectors of non-deterministic inputs for level j, the whole computation can be described in terms of two oracle relations:

$$\mathcal{R}_{range} = \{(\mathbb{i}, \mathbb{x}; \mathbb{w}) := (B, [\![\tilde{\mathbf{w}}]\!]; \tilde{\mathbf{w}}) \ : \ \|\tilde{\mathbf{w}}\|_\infty < B\}$$
$$\mathcal{R}_{AC} = \{(\mathbb{i}, \mathbb{x}; \mathbb{w}) := (\mathbf{C}, ([\![\tilde{\mathbf{x}}]\!], [\![\tilde{\mathbf{y}}]\!], [\![\tilde{\mathbf{w}}]\!]); (\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{w}})) \ : \ \mathbf{C}(\mathbf{x}, \mathbf{y}, \mathbf{w}) = 0\},$$

where $\mathbf{C}$ is an arithmetic circuit over R_q of constant-depth, obtained by flattening the CKKS circuit $\mathbf{C}'$. For the details, see [CCC+25]. Importantly, since all of the RNS primes have the same bit length, one can batch all the range proofs for $\{\mathbf{w}^{(i)}_{\text{ks},j}, \mathbf{w}^{(i)}_{\text{rmd}}\}_{i,j}$ together by approximating the bounds to 2^u, their nearest power of two. Let $[\![\tilde{\mathbf{x}}]\!], [\![\tilde{\mathbf{y}}]\!], [\![\tilde{\mathbf{w}}_{2^u}]\!], [\![\tilde{\mathbf{w}}_l]\!]$ be the oracles of the MLEs of the inputs $\mathbf{x}$, outputs $\mathbf{y} := \mathbf{w}^{(D-1)}_{\text{quot}}$, prover's values for RNS decompositions and remainders $\mathbf{w}_{2^u} := \mathbf{w}^{(0)}_{\text{ks},0} \| \ldots \| \mathbf{w}^{(0)}_{\text{ks},D-1} \| \ldots \| \mathbf{w}^{(D-1)}_{\text{ks},0} \| \mathbf{w}^{(0)}_{\text{rmd}} \| \ldots \| \mathbf{w}^{(D-1)}_{\text{rmd}}$ and prover's values for quotients $\mathbf{w}_l := \mathbf{w}^{(l)}_{\text{quot}}$, for $l = 0, \ldots, D-2$. The relation that describes the computation $\mathbf{C}'$ on CKKS ciphertexts can be compactly written as:

$$\mathcal{R}_{CKKS} = \Big\{ (\mathbf{C}, ([\![\tilde{\mathbf{x}}]\!], [\![\tilde{\mathbf{w}}_{2^u}]\!], \{[\![\tilde{\mathbf{w}}_l]\!]\}_{l=0}^{D-2}, [\![\tilde{\mathbf{y}}]\!]); (\mathbf{x}, \mathbf{w}_{2^u}, \{\mathbf{w}_l\}_{l=0}^{D-2}, \mathbf{y})) \ :$$

$$\underbrace{\mathbf{C}(\mathbf{x}, \mathbf{y}, \mathbf{w}_{2^u}, \{\mathbf{w}_l\}_{l=0}^{D-2}) = \mathbf{0}}_{\mathcal{R}_{AC}} \wedge \underbrace{\|\mathbf{w}_{2^u}\|_\infty < 2^u \wedge \bigwedge_{l=0}^{D-2} \|\mathbf{w}_l\|_\infty < \frac{q_{L-l}}{p_{L-l}} \wedge \|\mathbf{y}\|_\infty < \frac{q_{L-D+1}}{p_{L-D+1}}}_{\mathcal{R}_{range}} \Big\}$$

A modular PIOP Π_{CKKS} for $\mathcal{R}_{CKKS}$ can be built from PIOPs Π_{range} for $\mathcal{R}_{range}$ and Π_{AC} for $\mathcal{R}_{AC}$ using a simple 'and' composition (see Appendix A.6). Also, notice that we include the circuit $\mathbf{C}$ as part of the index $\mathbb{i}$; this allows the verifier to preprocess $\mathbf{C}$ once and later verify proofs in time $\mathsf{poly}(\log |\mathbf{C}|)$.

3 PIOPs for Unpacked Polynomials

In this section we give a set of useful results that allow us to construct PIOPs that can prove statements about "unpacked" versions of multilinear polynomials in $R_q[\boldsymbol{X}]$, which can be seen as a way to define statements about their $\mathbb{Z}_q$-coefficients. Our techniques generalize to any ring R and polynomial ring $\hat{R} := R[Y]/I(Y)$, where $I(Y) \in R[Y]$ of degree N. For ease of exposition, we limit the presentation to our rings of interest $R = \mathbb{Z}_q$ and $\hat{R} = R_q = \mathbb{Z}_q[Y]/(Y^N + 1)$.

3.1 Packing and Unpacking polynomials in R_q

Let $R_q = \mathbb{Z}_q[Y]/(Y^N + 1)$ for $N = 2^\nu$. We consider multivariate polynomial rings $R_q[X_1, \ldots, X_\ell]$ and $R_q[X_1, \ldots, X_\ell, Z_1, \ldots, Z_\nu]$. For brevity, we denote $\boldsymbol{X} = (X_1, \ldots, X_\ell)$ and $\boldsymbol{Z} = (Z_1, \ldots, Z_\nu)$. We define two useful functions,

[6] Layers are numbered from 0 to $D-1$, with inputs at level 0 and outputs at level $D-1$. Without loss of generality, one can assume that all layers have the same width.

pack and unpack, that allow switching between two different representations of a multilinear polynomial in $R_q[\boldsymbol{X}]$. The function unpack takes a multilinear polynomial $\tilde{p} \in R_q[\boldsymbol{X}]$ and returns a multilinear polynomial $\tilde{u} \in R_q[\boldsymbol{X}, \boldsymbol{Z}]$, of degree 0 in Y, which can be naturally identified with $\mathbb{Z}_q[\boldsymbol{X}, \boldsymbol{Z}]$, and is defined as follows:

$$\begin{aligned} \mathsf{unpack} : R_q^{(\leq 1)}[\boldsymbol{X}] &\to R_q^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}] \\ \tilde{p}(\boldsymbol{X}) \quad &\mapsto \tilde{u}(\boldsymbol{X}, \boldsymbol{Z}) = \sum_{\boldsymbol{b} \in \{0,1\}^\ell} \sum_{\boldsymbol{k} \in \{0,1\}^\nu} \tilde{p}(\boldsymbol{b})[\boldsymbol{k}] \cdot \tilde{\mathsf{eq}}((\boldsymbol{b}, \boldsymbol{k}), (\boldsymbol{X}, \boldsymbol{Z})). \end{aligned} \tag{1}$$

unpack essentially extracts all the coefficients in the variable Y of the evaluations of $\tilde{p}$ in the hypercube, and encodes these coefficients under the basis $\{\tilde{\mathsf{eq}}(\boldsymbol{k}, \boldsymbol{Z})\}_{\boldsymbol{k} \in \{0,1\}^\nu}$ instead of $\{Y^k\}_{k \in [N]}$. While unpack intuitively goes from (a subset of) $\mathbb{Z}_q[\boldsymbol{X}, Y]$ to $\mathbb{Z}_q[\boldsymbol{X}, \boldsymbol{Z}]$, we stress that its outputs are embedded in $R_q[\boldsymbol{X}, \boldsymbol{Z}]$. We informally call polynomials in the image of unpack *unpacked polynomials*.

The function $\mathsf{pack} : R_q[\boldsymbol{X}, \boldsymbol{Z}] \to R_q[\boldsymbol{X}]$ instead is defined as follows:

$$u(\boldsymbol{X}, \boldsymbol{Z}) \mapsto p(\boldsymbol{X}) = \sum_{\boldsymbol{k} \in \{0,1\}^\nu} u(\boldsymbol{X}, \boldsymbol{k}) \cdot Y^{\mathsf{to\text{-}int}(\boldsymbol{k})}. \tag{2}$$

Notice that in general pack is not the inverse of unpack, yet for any multilinear $\tilde{p}$ one can see that $\tilde{p} = \mathsf{pack}(\mathsf{unpack}(\tilde{p}))$. In particular, given a multilinear polynomial $\tilde{u}(\boldsymbol{X}, \boldsymbol{Z})$ that has degree 0 in Y, pack "packs" $\tilde{u}$ into a dense polynomial with fewer variables, hence the name.

Evaluating unpacked polynomials. A key property of unpacked polynomials (formalized in the Lemma below) is that an evaluation of an unpacked polynomial on $\mathbb{Z}_q$-elements can be computed from the evaluation of its packed version.[7]

Lemma 3. *Let* uEval *be the function that on inputs* $a \in R_q$ *and* $\boldsymbol{z} \in \mathbb{Z}_q^\nu$ *returns*

$$\mathsf{uEval}(a, \boldsymbol{z}) = \sum_{\boldsymbol{k} \in \{0,1\}^\nu} a[\boldsymbol{k}] \cdot \tilde{\mathsf{eq}}(\boldsymbol{k}, \boldsymbol{z}).$$

Then for any polynomial $\tilde{p} \in R_q^{(\leq 1)}[\boldsymbol{X}]$, $\tilde{u} = \mathsf{unpack}(\tilde{p})$, *and any* $(\boldsymbol{x}, \boldsymbol{z}) \in \mathbb{Z}_q^\ell \times \mathbb{Z}_q^\nu$: $\mathsf{uEval}(\tilde{p}(\boldsymbol{x}), \boldsymbol{z}) = \tilde{u}(\boldsymbol{x}, \boldsymbol{z})$, *and* uEval *is computable with* $O(N)$ *operations in* $\mathbb{Z}_q$.

Proof. Observe that the polynomials $\tilde{\mathsf{eq}}(\boldsymbol{b}, \boldsymbol{X}) \in R_q^{(\leq 1)}[\boldsymbol{X}]$ are such that $\tilde{\mathsf{eq}}(\boldsymbol{b}, \boldsymbol{x}) \in \mathbb{Z}_q$ if $\boldsymbol{x} \in \mathbb{Z}_q^\ell$. This allows to define the coefficients $a[\boldsymbol{k}]$ of $p(\boldsymbol{x})$ as follows

$$\begin{aligned} \tilde{p}(\boldsymbol{x}) &= \sum_{\boldsymbol{b} \in \{0,1\}^\ell} \tilde{p}(\boldsymbol{b}) \tilde{\mathsf{eq}}(\boldsymbol{b}, \boldsymbol{x}) = \sum_{\boldsymbol{b} \in \{0,1\}^\ell} \sum_{\boldsymbol{k} \in \{0,1\}^\nu} \tilde{p}(\boldsymbol{b})[\boldsymbol{k}] Y^{\mathsf{to\text{-}int}(\boldsymbol{k})} \tilde{\mathsf{eq}}(\boldsymbol{b}, \boldsymbol{x}) \\ &= \sum_{\boldsymbol{k} \in \{0,1\}^\nu} \underbrace{\left(\sum_{\boldsymbol{b} \in \{0,1\}^\ell} \tilde{p}(\boldsymbol{b})[\boldsymbol{k}] \tilde{\mathsf{eq}}(\boldsymbol{b}, \boldsymbol{x}) \right)}_{= a[\boldsymbol{k}] \in \mathbb{Z}_q} Y^{\mathsf{to\text{-}int}(\boldsymbol{k})} = \sum_{k=0}^{N-1} a[k] Y^k. \end{aligned}$$

[7] Precisely, we mean evaluations on elements of R_q of degree 0 in Y; for simplicity we abuse notation and denote this subset of R_q by $\mathbb{Z}_q$.

PIOP Π_{ueval} for $((y_1, \boldsymbol{x}_1, \ldots, y_a, \boldsymbol{x}_a, [\![\tilde{p}]\!]); \tilde{p}) \in \mathcal{R}_{ueval}$

Let $\ell' = \ell + \nu$, $\boldsymbol{X} = (X_1, \ldots, X_{\ell'})$, $\boldsymbol{Z} = (Z_1, \ldots, Z_\nu)$.

1. $\mathcal{V}$ sends $\boldsymbol{\sigma} \leftarrow\!\!\$\ \hat{S}_0^\nu, \tau \leftarrow\!\!\$\ \hat{S}_0$.
2. Let $E(\boldsymbol{X}, \boldsymbol{Z}, T) = \sum_{i=1}^{a} \sum_{\boldsymbol{k}\in\{0,1\}^\nu} \tilde{\mathsf{eq}}(\boldsymbol{X}, \boldsymbol{x}_i)[\boldsymbol{k}] \cdot \tilde{\mathsf{eq}}(\boldsymbol{Z}, \boldsymbol{k}) \cdot T^{i-1} \in \mathbb{Z}_q[\boldsymbol{X}, \boldsymbol{Z}, T]$ and $\tilde{E}_{\sigma,\tau}(\boldsymbol{X}) = E(\boldsymbol{X}, \boldsymbol{\sigma}, \tau)$. Let $\tilde{y}(\boldsymbol{Z}, T) = \sum_{\boldsymbol{k}\in\{0,1\}^\nu} \tilde{\mathsf{eq}}(\boldsymbol{k}, \boldsymbol{Z}) \sum_{i=1}^{a} y_i[\boldsymbol{k}] \cdot T^{i-1}$. $\mathcal{P}$ and $\mathcal{V}$ run Π^0_{sum} for $((\tilde{y}(\boldsymbol{\sigma}, \tau), [\![f]\!]); f) \in \mathcal{R}_{sum}$ where $f = \mathsf{unpack}(\tilde{p}) \cdot \tilde{E}_{\sigma,\tau}$. Given $\Pi^0_{sum}.\mathcal{V}$'s query to $[\![f]\!]$ on $\boldsymbol{r} = (\boldsymbol{r}_1, \boldsymbol{r}_2) \in \mathbb{Z}_q^{\ell+\nu}$, the verifier $\Pi_{ueval}.\mathcal{V}$:
   - computes $\psi = \sum_{i=1}^{a} \tilde{\mathsf{eq}}(\boldsymbol{r}, \boldsymbol{x}_i)\tau^{i-1}$, parsed as $\psi = \sum_{\boldsymbol{k}} \psi[\boldsymbol{k}] Y^{\mathsf{to\text{-}int}(\boldsymbol{k})}$,
   - computes $\psi^* = \sum_{\boldsymbol{k}} \psi[\boldsymbol{k}] \tilde{\mathsf{eq}}(\boldsymbol{k}, \boldsymbol{\sigma})$,
   - makes a query to $[\![\tilde{p}]\!]$ on $\boldsymbol{r}_1$ and receives answer $v = \tilde{p}(\boldsymbol{r}_1)$,
   - returns $\mathsf{uEval}(v, \boldsymbol{r}_2) \cdot \psi^*$ to $\Pi^0_{sum}.\mathcal{V}$.

Figure 1: PIOP for $\mathcal{R}_{ueval}$

By plugging the above definition of $a[\boldsymbol{k}]$ into the definition of uEval

$$\sum_{\boldsymbol{k}\in\{0,1\}^\nu} a[\boldsymbol{k}] \cdot \tilde{\mathsf{eq}}(\boldsymbol{k}, \boldsymbol{z}) = \sum_{\boldsymbol{b}\in\{0,1\}^\ell} \sum_{\boldsymbol{k}\in\{0,1\}^\nu} \tilde{p}(\boldsymbol{b})[\boldsymbol{k}] \cdot \tilde{\mathsf{eq}}(\boldsymbol{b}, \boldsymbol{x}) \cdot \tilde{\mathsf{eq}}(\boldsymbol{k}, \boldsymbol{z}),$$

which is $= \tilde{u}(\boldsymbol{x}, \boldsymbol{z})$ by construction of unpack in Eq. (1). The computability of uEval in time $\mathcal{O}(N)$ follows by an algorithm from [VSBW13] that computes all the multilinear Lagrange polynomials in time $\mathcal{O}(2^\nu)$. □

Unpacked Virtual Oracles. A useful implication of Lemma 3 is that we can create virtual oracles (see Definition 4) of unpacked polynomials and run a PIOP on a virtual oracle $[\![g(\tilde{u}_1, \ldots, \tilde{u}_k)]\!]$, where each $\tilde{u}_i = \mathsf{unpack}(\tilde{p}_i)$ and g is a function that admits virtual oracles. This holds as long as the PIOP's verifier queries to each unpacked oracle $[\![\tilde{u}_i]\!]$ are only on points of $\mathbb{Z}_q$. This implication follows simply by seeing that for unpack, the associated functions ev_{unpack}, h for the virtual oracle (see Definition 4), are $ev_{\mathsf{unpack}} = \mathsf{uEval}$ and $h(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}$. In Appendix A.4, we give a simple example of how to use a virtual unpacked polynomial to prove a sumcheck $y = \sum_{\boldsymbol{b}\in\{0,1\}^\ell, \boldsymbol{k}\in\{0,1\}^\nu} \tilde{p}(\boldsymbol{b})[\boldsymbol{k}]$ over the $\mathbb{Z}_q$ coefficients.

3.2 PIOP for the Evaluation of Unpacked Polynomials on R_q points

The result of Lemma 3 shows how to compute the evaluation of an unpacked polynomial on $\mathbb{Z}_q$-points, given evaluations of its packed version. In this section, we address the more general problem of evaluating an unpacked polynomial on points of R_q. While these evaluations cannot be directly reduced to evaluations of the packed polynomial, here we present a protocol that allows proving such evaluation. Precisely, we present a PIOP Π_{ueval} for the following oracle relation:

$$\mathcal{R}_{ueval} = \{((y_1, \boldsymbol{x}_1, \ldots, y_a, \boldsymbol{x}_a, [\![\tilde{p}]\!]); \tilde{p}) : \forall i\, y_i = \tilde{u}(\boldsymbol{x}_i) \text{ where } \tilde{u} = \mathsf{unpack}(\tilde{p})\}.$$

The full description of Π_{ueval} is in Figure 1. Here we present the ideas in more detail and justify its correctness. For simplicity, we focus on the case of evaluating

at a single point, i.e., $a = 1$; the extension to more points is straightforward observing that the aggregation random challenge $\tau \in \hat{S}_0 \subset \mathbb{Z}_q$.

The goal is to prove that for $\tilde{u} = \mathsf{unpack}(\tilde{p})$ it holds $y = \tilde{u}(\boldsymbol{x})$, which can be written as follows (where $\ell' = \ell + \nu$).

$$\begin{aligned}\tilde{u}(\boldsymbol{x}) &= \sum_{\boldsymbol{b}\in\{0,1\}^{\ell'}} \tilde{u}(\boldsymbol{b})\tilde{\mathsf{eq}}(\boldsymbol{b},\boldsymbol{x}) = \sum_{\boldsymbol{b}\in\{0,1\}^{\ell'}} \tilde{u}(\boldsymbol{b}) \sum_{\boldsymbol{k}\in\{0,1\}^{\nu}} \underbrace{\tilde{\mathsf{eq}}(\boldsymbol{b},\boldsymbol{x})[\boldsymbol{k}]}_{\tilde{E}(\boldsymbol{b},\boldsymbol{k})} Y^{\mathsf{to\text{-}int}(\boldsymbol{k})} \\ &= \sum_{\boldsymbol{k}\in\{0,1\}^{\nu}} \sum_{\boldsymbol{b}\in\{0,1\}^{\ell'}} \tilde{u}(\boldsymbol{b})\tilde{E}(\boldsymbol{b},\boldsymbol{k}) Y^{\mathsf{to\text{-}int}(\boldsymbol{k})} = \sum_{\boldsymbol{k}\in\{0,1\}^{\nu}} y[\boldsymbol{k}] Y^{\mathsf{to\text{-}int}(\boldsymbol{k})}.\end{aligned}$$

In particular, the last equality follows from the fact that $\tilde{u}(\boldsymbol{b})$ as well as $\tilde{E}(\boldsymbol{b},\boldsymbol{k})$ are in $\mathbb{Z}_q$. Therefore, we can reduce testing $y = \tilde{u}(\boldsymbol{x})$ to checking that for all $\boldsymbol{k} \in \{0,1\}^{\nu}$, $y[\boldsymbol{k}] = \sum_{\boldsymbol{b}} \tilde{u}(\boldsymbol{b})\tilde{E}(\boldsymbol{b},\boldsymbol{k})$. These are 2^{ν} sumchecks that we can batch together by sampling a random $\boldsymbol{\sigma}$ and checking

$$\tilde{y}(\boldsymbol{\sigma}) = \sum_{\boldsymbol{k}\in\{0,1\}^{\nu}} y[\boldsymbol{k}]\tilde{\mathsf{eq}}(\boldsymbol{k},\boldsymbol{\sigma}) = \sum_{\boldsymbol{b}\in\{0,1\}^{\ell'}} \tilde{u}(\boldsymbol{b})\tilde{E}(\boldsymbol{b},\boldsymbol{\sigma}).$$

This is done by running the PIOP Π^0_{sum} (i.e., the sumcheck PIOP with the verifier's random challenges in $\hat{S}_0 \subset \mathbb{Z}_q$), which ends with the verifier querying $\tilde{u}(\boldsymbol{r})$ and $\tilde{E}(\boldsymbol{r},\boldsymbol{\sigma})$. The query to $\tilde{u}$ can be computed through a query to $\tilde{p}$ using uEval. $\tilde{E}(\boldsymbol{r},\boldsymbol{\sigma})$ can be computed by the verifier as $\psi^* = \sum_{\boldsymbol{k}} \psi[\boldsymbol{k}]\tilde{\mathsf{eq}}(\boldsymbol{k},\boldsymbol{\sigma})$, where $\psi = \tilde{\mathsf{eq}}(\boldsymbol{x},\boldsymbol{r})$, parsed as $\psi = \sum_{\boldsymbol{k}} \psi[\boldsymbol{k}] Y^{\mathsf{to\text{-}int}(\boldsymbol{k})}$. To see correctness:

$$\begin{aligned}\tilde{E}(\boldsymbol{r},\boldsymbol{\sigma}) &= \sum_{\boldsymbol{b},\boldsymbol{k}} \tilde{E}(\boldsymbol{b},\boldsymbol{k})\tilde{\mathsf{eq}}(\boldsymbol{b},\boldsymbol{r})\tilde{\mathsf{eq}}(\boldsymbol{k},\boldsymbol{\sigma}) = \sum_{\boldsymbol{k}}\sum_{\boldsymbol{b}} \tilde{\mathsf{eq}}(\boldsymbol{b},\boldsymbol{x})[\boldsymbol{k}]\tilde{\mathsf{eq}}(\boldsymbol{b},\boldsymbol{r})\tilde{\mathsf{eq}}(\boldsymbol{k},\boldsymbol{\sigma}) \\ &= \sum_{\boldsymbol{k}} \tilde{\mathsf{eq}}(\boldsymbol{x},\boldsymbol{r})[\boldsymbol{k}] \cdot \tilde{\mathsf{eq}}(\boldsymbol{k},\boldsymbol{\sigma}) = \psi^*,\end{aligned}$$

where last equality follows from $\tilde{\mathsf{eq}}(\boldsymbol{x},\boldsymbol{r}) = \sum_{\boldsymbol{b}} \tilde{\mathsf{eq}}(\boldsymbol{b},\boldsymbol{x})\tilde{\mathsf{eq}}(\boldsymbol{b},\boldsymbol{r})$ and $\tilde{\mathsf{eq}}(\boldsymbol{b},\boldsymbol{r}) \in \mathbb{Z}_q$.

Theorem 1. *Π_{ueval} is a PIOP for $\mathcal{R}_{ueval}$ with perfect completeness, prover's efficiency $\mathcal{O}\big(a2^{\ell+\nu}\big)$, verifier's efficiency $\mathcal{O}(\ell + a \cdot N)$, $3(\ell+\nu)\log q$ bits of communication, and δ_{ueval}-knowledge soundness with $\delta_{ueval} = (a-1+2\ell+3\nu)/|\hat{S}_0|$.*

Remark 1 (Parallel repetition of Π_{ueval}). If the exceptional set $\hat{S}_0$ is not large enough to make $1/|\hat{S}_0|$ negligible, we apply κ-fold parallel repetition, denoted $\Pi^{(\kappa)}_{ueval}$, that achieves soundness δ^{κ}_{ueval} and RBR-soundness $\delta^{\kappa/(\ell+\nu)}_{ueval}$ (see Theorem 4) [CCH+19]. The efficiency of $\Pi^{(\kappa)}_{ueval}$ is κ times that of Π_{ueval} and its verification requires κ queries to the (packed) oracle $[\![\tilde{p}]\!]$.

4 Our PIOP for Range Checks over Packed Polynomials

In this section, we present our new PIOP for range checks that improves over the prior solution from [CCC+25]. More precisely, we construct a PIOP for

$$\mathcal{R}_{range} = \{(\mathbb{i},\mathbb{x};\mathbb{w}) = (T_B, [\![\tilde{v}]\!]; \tilde{v}) : \forall \boldsymbol{b} \in \{0,1\}^{\ell}\ \tilde{v}(\boldsymbol{b}) \in T_B\}, \tag{3}$$

where $R_q := \mathbb{Z}_q[Y]/(Y^N + 1)$, $\tilde{v} \in R_q^{(\leq 1)}[X_1, \ldots, X_\ell]$, and T_B is the table of size B^N containing all the elements of R_q with coefficients in $[0, B-1]$.

As shown in [CCC+25], extending ideas of [STW24] from finite fields to R_q, one can reduce a lookup in T_B into $c \cdot N$ lookups in the smaller table $\boldsymbol{t}_\beta = (0, \ldots, \beta - 1)$ for $\beta = B^{1/c}$. This uses a lemma similar to Lemma 4 below with a slight difference caused by our use of unpack. See Appendix C for the proof.

Lemma 4. *Let $B = 2^b$, $N = 2^\nu$, let $c = 2^\gamma$ such that $c \mid b$ and $\beta = B^{1/c}$. Let T_B be the table defined above, and $\boldsymbol{t}_\beta = (0, \ldots, \beta - 1)$. Then for any $\tilde{v} \in R_q^{(\leq 1)}[X_1, \ldots X_\ell]$, $(T_B, [\![\tilde{v}]\!]; \tilde{v}) \in \mathcal{R}_{range}$ if and only if there exists $\tilde{\mu} \in \mathbb{Z}_q^{(\leq 1)}[X_1, \ldots, X_\ell, Z_1, \ldots, Z_\nu, Z'_1, \ldots, Z'_\gamma]$ such that, for $\tilde{u} = \mathsf{unpack}(\tilde{v})$,*

$$\forall \boldsymbol{i} \in \{0,1\}^{\ell+\nu} : \ \tilde{u}(\boldsymbol{i}) = \sum_{\boldsymbol{j} \in \{0,1\}^\gamma} \tilde{\mu}(\boldsymbol{i}, \boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})} \tag{4}$$

$$\forall \boldsymbol{b} \in \{0,1\}^{\ell+\nu+\gamma} : \ \tilde{\mu}(\boldsymbol{b}) \in \boldsymbol{t}_\beta \tag{5}$$

Based on this Lemma, one can construct a PIOP Π_{range} for proving that $(T_B, [\![\tilde{v}]\!]; \tilde{v})$ is in $\mathcal{R}_{range}$ by letting the prover send a polynomial $\tilde{\mu}$ and then showing that (i) $\tilde{\mu}$ satisfies equation (5), i.e., its evaluations are in the smaller table $\boldsymbol{t}_\beta$, and (ii) $\tilde{\mu}$ is correctly related to $\tilde{v}$ according to equation (4), i.e, it is a β-decomposition of $\mathsf{unpack}(\tilde{v})$. The protocol of [CCC+25] followed a similar approach by constructing a PIOP for each of these equations;[8] however, a large source of inefficiency in their case stems from embedding $\tilde{\mu}$ in $R_q^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z}']$ which, in spite of its sparsity, is an object that is $c \cdot N$ times bigger than $\tilde{v}$. In particular, this causes an $\mathcal{O}(N)$ overhead in the entire protocol.

In this paper, we avoid this overhead through the combination of two techniques. First, we embed $\tilde{\mu}$ over a different ring $H_q = \mathbb{Z}_q[Y]/h(Y)$ and prove that it satisfies equation (5) for the table $\boldsymbol{t}_\beta \subset \mathbb{Z}_q$ via a PIOP over H_q. Crucially, we choose H_q to be much smaller than R_q, as $d = \deg(h) \ll N$ is a small constant (e.g., $d = 2$ or 4) that ensures the existence of a large enough exceptional set $\hat{S}_d \subset H_q$ (we explain how we choose h below). With this choice, the PIOP running over H_q avoids altogether an $\mathcal{O}(N)$ overhead. In Appendix D we also give an optimized variant that takes $H_q = \mathbb{F}_p$ for a large prime p, which we use in Section 8.

To complete the picture, we need to connect the range proof about $\tilde{\mu}$ to the polynomial $\tilde{v}$, which is the end-goal of Π_{range}. To this end, we relate $\tilde{\mu} \in H_q^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z}']$ with $\tilde{v} \in R_q^{(\leq 1)}[\boldsymbol{X}]$ according to equation (4), and leverage our unpacked evaluation technique to let the verifier query $\tilde{u} = \mathsf{unpack}(\tilde{v})$ without requiring the prover to commit to $\tilde{u}$.

We remark that our technique is similar to the ring-switching used in [HMZ25], although in that work it is primarily used to transform an R1CS statement over R_q into one over a Galois ring where the entire computation is eventually done, while in our case the change of ring is used (in combination with our unpacking

[8] Precisely, equation (4) is encoded differently without the unpack encoding.

technique) for the specific task of proving ranges. Moreover, the ring-switching technique of [HMZ25] was later used in [NOZ26] to construct lattice-based multilinear polynomial commitments with concrete efficiency in verification.

In what follows we present our protocol in more detail.

Our PIOP Π_{range}. Consider the splitting of $R_q \cong \prod_{i=0}^{L} \left(\prod_{j=0}^{k-1} R_{i,j}\right)$ described in section 2.3, where each $R_{i,j} = \mathbb{Z}_{p_i}[Y]/(Y^d - \zeta_i^{(2j+1)})$ is isomorphic to the field $\mathbb{F}_{p_i^d}$. We set $H_q = \mathbb{Z}_q[Y]/h(Y)$ as a ring isomorphic to $\prod_{i=0}^{L} R_{i,0}$ by the CRT, i.e., where for all i, $h(Y) \bmod p_i = (Y^d - \zeta_i)$. This choice ensures the existence of an exceptional set $\hat{S}_d \subset H_q$ of size p_{min}^d, where $p_{min} = \min\{p_0, \ldots, p_L\}$.

Let $\hat{\mathcal{R}}_{range}$ be an oracle relation over $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu})$ defined in the same way as $\mathcal{R}_{range}$ except that its index is restricted to tables $\boldsymbol{t}_\beta \subset \mathbb{Z}_q$ and the relation is defined over multilinear polynomials $\tilde{\mu}$ with coefficients in H_q. It is easy to see that we can rephrase Lemma 4 by embedding $\tilde{\mu}$ in $H_q^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z'}]$ and replacing the condition of equation (5) with $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \hat{\mathcal{R}}_{range}$.

We construct a PIOP Π_{range} for $\mathcal{R}_{range}$ with table T_B by using a PIOP $\hat{\Pi}_{range}$ for $\hat{\mathcal{R}}_{range}$ with the smaller $\boldsymbol{t}_\beta$,[9] combined with a probabilistic test to check equation (4). The full description of Π_{range} is in Fig. 2.

The protocol proceeds as follows. The prover computes the unpacking of $\tilde{v}$ and its β-decomposition according to equation 4 and sends the oracle $[\![\tilde{\mu}]\!]$ to the verifier. Then the goal is to convince the verifier that $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \mathcal{R}_{range}$ and that $\tilde{v}$ and $\tilde{\mu}$ satisfy equation (4). The former is achieved by executing the PIOP $\hat{\Pi}_{range}$ (step 3 in Fig. 2). For the latter, the idea is to test that the polynomial $\tilde{u}(\boldsymbol{X}, \boldsymbol{Z})$ is a linear combination of partial evaluations of $\tilde{\mu}$, i.e., $\tilde{u}(\boldsymbol{X}, \boldsymbol{Z}) = \sum_{\boldsymbol{j}} \tilde{\mu}(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{j}) \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}$. Intuitively, this could be tested by evaluating the polynomials on a random point, i.e., querying $\tilde{u}(\boldsymbol{r})$ and $\tilde{\mu}(\boldsymbol{r}, \boldsymbol{j})$ for all $\boldsymbol{j}$ and testing $\tilde{u}(\boldsymbol{r}) = \sum_{\boldsymbol{j}} \tilde{\mu}(\boldsymbol{r}, \boldsymbol{j}) \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}$. However, these polynomials are over different rings with different exceptional sets. To solve this issue we take the random point $\boldsymbol{r}$ in the exceptional set $\hat{S}_0 \subset \mathbb{Z}_q$, that is common to both R_q and H_q. However, since $\hat{S}_0$ is not large enough to make $1/|\hat{S}_0| = p_{min}^{-1} \approx 2^{-\lambda}$ we repeat this test κ times. Notably, since this parallel repetition test happens in a single round we can take the smallest κ that make $p_{min}^{-\kappa} \approx 2^{-\lambda}$; e.g., for our choice of 45-bit primes (see Appendix I) we take $\kappa = 3$. The last problem to solve is that the test above requires querying a polynomial $\tilde{u}$ that is larger than $\tilde{v}$ by a factor of N. We define $\tilde{u}$ as an unpacked virtual oracle, and using the fact that the random challenges are in $\hat{S}_0 \subset \mathbb{Z}_q$, we apply our result of Lemma 3, which reduces queries to $\tilde{u} = \mathsf{unpack}(\tilde{v})$ to queries to the (dense) packed polynomial $\tilde{v}$.

The proof of the following theorem appears in Appendix C.2.

Theorem 2. *Let $\hat{S}_d \subset H_q$, and $\hat{S}_0 \subset \mathbb{Z}_q$ be exceptional sets. Let $\hat{\Pi}_{range}$ be a PIOP for $\hat{\mathcal{R}}_{range}$ with perfect completeness and RBR-knowledge error $\hat{\delta}_{range}$. Then Π_{range} is a PIOP with perfect completeness and RBR-knowledge error $\delta_{range} = \max((\frac{\ell+\nu}{|\hat{S}_0|})^\kappa, \frac{\kappa \cdot \gamma}{|\hat{S}_d|}, \hat{\delta}_{range})$.*

[9] We assume B a power of 2; see [CCC+25] for the extension to the general case.

PIOP Π_{range} for $(T_B, [\![\tilde{v}]\!]; \tilde{v}) \in \mathcal{R}_{range}$

1. $\mathcal{P}$ computes $\tilde{u} = \mathsf{unpack}(\tilde{v}) \in R_q^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}]$ and $\tilde{\mu} \in H_q^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z}']$ as the decomposition of $\tilde{u}$ according to equation (4), and sends $[\![\tilde{\mu}]\!]$.
   $\mathcal{V}$ samples $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\kappa \leftarrow\!\$\ \hat{S}_0^{\ell+\nu}$ and sends them to $\mathcal{P}$.
2. $\mathcal{P}$ sends $\tilde{\mu}_i^*(Z'_1, \ldots, Z'_\gamma) = \tilde{\mu}(\boldsymbol{r}_i, Z'_1, \ldots, Z'_\gamma)$, for $i = 1$ to κ.
   $\mathcal{V}$ samples $\boldsymbol{s} \leftarrow\!\$\ \hat{S}_d^\gamma$ and, for $i = 1$ to κ, queries the oracle $[\![\tilde{\mu}]\!]$ on $(\boldsymbol{r}_i, \boldsymbol{s})$, and the virtual oracle $[\![\tilde{u}]\!] = [\![\mathsf{unpack}(\tilde{v})]\!]$ on $\boldsymbol{r}_i$. Then $\mathcal{V}$ checks the following:
   $\forall i = 1, \ldots, \kappa$: $\tilde{u}(\boldsymbol{r}_i) = \sum_{\boldsymbol{j} \in \{0,1\}^\gamma} \tilde{\mu}_i^*(\boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}$ and $\tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{s}) = \tilde{\mu}_i^*(\boldsymbol{s})$
3. $\mathcal{P}$ and $\mathcal{V}$ run a PIOP $\hat{\Pi}_{range}$ for $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \hat{\mathcal{R}}_{range}$.

Figure 2: Construction of the PIOP Π_{range} from $\hat{\Pi}_{range}$

To obtain the final protocol Π_{range}, we need to instantiate the PIOP $\hat{\Pi}_{range}$ for the relation $\hat{\mathcal{R}}_{range}$ where the table in the index is $[0, \beta - 1] \subset \mathbb{Z}_q$ and the PIOP works over the ring H_q. We set this protocol to be the same as that in [CCC+25] with the only difference that instead of working over R_q and its exceptional set $\hat{S}_N \subset R_q$, we redefine it over the ring H_q with exceptional set $\hat{S}_d$. For completeness, we give a full description of $\hat{\Pi}_{range}$ in Appendix D, and present an optimized version in Appendix D.5. We also defer to Appendix D.5 a discussion on its efficiency, and we give a full efficiency analysis for concrete parameters in Section 8.

5 Improved Arithmetization and PIOP Π_{AC} for CKKS

We construct a PIOP for the relation $\mathcal{R}_{AC}$ (Section 2.3) which, combined with Π_{range} from the previous section, yields a PIOP for proving computations on CKKS ciphertexts, following [CCC+25]. We refer to Appendix F for an explanation about which of the polynomial oracles from this section are in common with the input oracle Π_{range}, i.e., which values require a range check.

We improve upon the arithmetization of CKKS in [CCC+25] on several fronts. First, we replace their arithmetic circuit for $\mathcal{R}_{AC}$ with a more expressive one which allows to "copy" (into a MLE $\tilde{V}_{3,copy}$) the results of plaintext-ciphertext products without going through base decomposition and rescaling. We also allow plaintext inputs from the prover, provided through an oracle MLE $\tilde{V}_{3,ptxt}$. This enables commit-and-prove SNARKs where the prover commits to $\tilde{V}_{3,ptxt}$.

Second and most importantly, by careful labeling of the wires in this new circuit, we construct MLEs for the wiring predicates that are sparse in a very structured way, and have significantly fewer variables. Beyond the obvious improvement of reducing the amount of variables, this structured sparsity speeds up the sumcheck-based layer consistency equations in different ways:

- We provide specialized linear-time prover algorithms that improve upon a generic application of the algorithms from [XZZ+19,ZLW+21];

- For wiring predicates that are independent of the structure of the HE circuit (i.e. the ones that describe base decomposition and rescaling), we provide closed form equations that the verifier can easily evaluate on their own;
- For those that are not, we provide efficient polynomial commitments in Section 6 that exploit the structured sparsity. There are up to 7 oracles of this form in our work, depending on the actual circuit structure. For example, if we only wanted to capture the less expressive relation $\mathcal{R}_{AC}$ in [CCC+25] without plaintext inputs, we would only have 3 oracles of this form.
- For a HE circuit of depth D, the PIOP of [CCC+25] had $13 + 2(D-1)$ sparse polynomials in the index and $D+1$ dense polynomials in the witness. We describe how to implement our more expressive relation with up to 7 sparse polynomials (with fewer variables) and a single dense one for the witness.

Notation: Within this section, let $\delta = \lceil \log_2(D) \rceil$ and let $\mu = \lceil \log_2(2M) \rceil + \delta$, where D is the depth of the HE circuit and M is the number of HE multiplications in its layer with the most multiplications. We denote vector of variables as $\mathbf{X} = (X_0, X_1, \ldots)$ and refer to the leftmost one as the least significant one. By abuse of notation, we use d_2 (resp. d_0, d_1) to refer to the set of wires holding the d_2 (resp. d_0, d_1) output of premultiply, according to the notation in Appendix A.5. For a boolean vector $\mathbf{x}$, $V_{i,S}(\mathbf{x})$ denotes the value on wire $\mathbf{x}$ for the set of wires S in GKR layer i (not to be confused with the CKKS level).

Due to space constraints, the predicates already introduced in [CCC+25] are given in Appendix E. Table 1 gives information about the amount of variables in the different MLEs for the intermediate values of the circuit, as well as what they are meant to index from their least to their most significant variables. Some subset of these variables can be thought of as "metadata". For example, each predicate has δ variables which, together, index one of the D layers in the HE circuit. Table 1 does not include $\tilde{V}_{3,copy}$, $\tilde{V}_{3,ptxt}$ and $\tilde{V}_{in}$, since we do not need closed form equations for the predicates that interface with them.

Notice that the overhead of adding our different forms of metadata is small. For example, without the "HE level metadata", predicates in Table 1 would need to have as many variables as $\log_2$ of the number of their relevant wires in the whole HE circuit. Given the metadata, we only need to have as many variables as the *maximum* among such relevant kind of wires on each level. For example, let $\{d_{2,j}\}_{j=0}^{D-1}$ represent the subset of d_2 wires that belong to the j-th HE layer. The MLE $\tilde{V}_{2,d_2}(\mathtt{X}_{[\#(d_2)]})$ is such that $\#(d_2) = \delta + \max(\{\#(d_{2,j})\}_{j=0}^{D-1}) = \mu - 1$.

Let us now explain how we *achieve* the closed form equations for the MLEs and in particular their structured sparsity and reduced amount of variables. Whereas the first technique originates from [ZLW+21], to the best of our knowledge we are the first ones to exploit the other two we present.

[ZLW+21] trick: In our layer consistency equations, rather than having a different set of variables for every kind of wiring predicate (as in [CCC+25]), we use the same set of variables $\mathtt{X}$ for all of them as much as possible. This results in equations of the form $\tilde{V}_{out}(\mathtt{Z}) = \sum_j \sum_{\mathbf{x} \in \{0,1\}^l} \widetilde{\mathtt{gate}}'_j(\mathtt{Z}, \mathbf{x}) \cdot \tilde{V}_{i_j}(\mathbf{x}_{[l_j]})$ where l_j is the amount of variables in $\tilde{V}_{i_j}$ and $l = \max_j(l_j)$. While it could seem that $\widetilde{\mathtt{gate}}'_j(\mathtt{Z}, \mathbf{x})$

	LSPs		Middle positions		MSPs	
	Bits	Meaning	Bits	Meaning	Bits	Meaning
$\tilde{V}_0$	$\mathtt{T}_0, \mathtt{T}_1$	Gate type	$\mathtt{Z}$	Gate number (for that type)	-	-
$\tilde{V}_1$	$\mathtt{Z}_{[\delta]}$	HE level	$\mathtt{Z}_\delta$	key-switch output type: c'_0 vs c'_1	$\mathtt{Z}_{(\delta,\mu)}$	wire on level
$\tilde{V}_{2,d_{01}}$	$\mathtt{Z}_{[\delta]}$	HE level	$\mathtt{Z}_\delta$	pre-mult output type: d_0 vs d_1	$\mathtt{Z}_{(\delta,\mu)}$	wire on level
$\tilde{V}_{2,d_2}$	$\mathtt{Z}_{[\delta]}$	HE level	$\mathtt{Z}_{[\delta,\mu-1)}$	wire carrying pre-mult output d_2	-	-
$\tilde{V}_2$	$\mathtt{Z}_0$	d_{01} vs d_2	As $\tilde{V}_{2,d_{01}}$ or $\tilde{V}_{2,d_2}$			
$\tilde{V}_{3,BD}$	$\mathtt{X}_{[\delta]}$	HE level	$\mathtt{Z}_{[\delta,\mu-1)}$	wire carrying pre-mult output d_2	$\mathtt{X}_{[\delta,2\delta)}$	specific p_i
$\tilde{V}_{3,quot}$	$\mathtt{Y}_{[\delta]}$	HE level	$\mathtt{Y}_\delta$	mult output type: c''_0 vs c''_1	$\mathtt{Y}_{(\delta,\mu)}$	wire on level
$\tilde{V}_{3,rc_3}$	$\mathtt{X}_{[\delta]}$	HE level	$\mathtt{Z}_{[\delta,\mu)}$	$\mathtt{Z}_\delta$: c''_0 vs c''_1 | $\mathtt{Z}_{(\delta,\mu)}$: wire on level	$\mathtt{X}_\delta$	rmd vs quot

Table 1: Variable information for the different intermediate-values MLEs. LSPs = Least significant positions, MSPs = Most significant positions.

is not truncated accordingly, we internally modify that MLE so as to selectively reduce the boolean hypercube, i.e. $\widetilde{\mathtt{gate}}'_j(\mathtt{Z}, \mathbf{x}) = \widetilde{\mathtt{eq}}(\mathbf{1}, \mathbf{x}_{[l_j,l)}) \cdot \widetilde{\mathtt{gate}}_j(\mathtt{Z}, \mathbf{x}_{[l_j]})$.

Metadata: Our pieces of "metadata" in the wires allow to easily apply the level-appropriate constants when describing the MLEs for the wiring predicates that correspond to the rescaling and base decomposition operations.

`X` to `Z`: Finally, a precise labelling of the wires involved in the expression

$$\tilde{V}_{out}(\mathtt{Z}) = \sum_j \sum_{\mathbf{x}\in\{0,1\}^l} \widetilde{\mathtt{gate}}'_j(\mathtt{Z}, \mathbf{x}) \cdot \tilde{V}_{i_j}(\mathbf{x}_{[l_j]})$$

results in MLEs $\widetilde{\mathtt{gate}}'_j(\mathtt{Z}, \mathbf{x})$ which internally have terms $\widetilde{\mathtt{eq}}(\mathtt{Z}_{[l'_j]}, \mathtt{X}_{[l'_j]})$ of significant length. Whenever such a factor is present, one may hope to drop the corresponding l'_j summation variables altogether and feed $\mathtt{Z}_{[l'_j]}$ directly into $\tilde{V}_{i_j}$, obtaining expressions that look as follows, where $l' = \max_j(l_j - l'_j)$:

$$\tilde{V}_{out}(\mathtt{Z}) = \sum_j \sum_{\mathbf{x}\in\{0,1\}^{l'}} \widetilde{\mathtt{gate}}'_j(\mathtt{Z}, \mathbf{x}) \cdot \tilde{V}_{i_j}(\mathtt{Z}_{[l'_j]}, \mathbf{x}_{[l_j - l'_j]}). \tag{6}$$

This reduces the amount of `X` variables, which are the ones driving the efficiency of the sumcheck protocol. The substitution, however, is not always legitimate: It may only be applied to those variables which the wiring predicate constrains *exclusively* through the equality factor $\widetilde{\mathtt{eq}}(\mathtt{Z}_{[l'_j]}, \mathtt{X}_{[l'_j]})$. If the predicate uses them for anything else, the right hand side of Eq. (6) increases its degree in those variables and stops being the multilinear polynomial $\tilde{V}_{out}$. It still agrees with $\tilde{V}_{out}$ on the boolean hypercube, but not at the random point at which the GKR verifier evaluates the layer consistency equation, so an honest prover would be rejected. As we will see in Section 5.1, the substitution is therefore *not* applied to the "HE level" metadata variables of the predicates that select level-dependent constants ($\widetilde{\mathtt{rescon}}_3, \widetilde{\mathtt{bdcon}}_3, \widetilde{\mathtt{evk}}$), which are kept in `X`.

5.1 Layer consistency equations

For a reminder of the wiring predicates in [CCC+25] see Appendix E.2. In the following, we adapt those through the application of the three techniques outlined above. We directly provide the MLEs of these new variations.

Layer 0: Let us start by looking at the MLE for the output layer $\tilde{V}_0$. We define an extra vector of variables $\mathtt{T} = (\mathtt{T}_0, \mathtt{T}_1)$ as metadata for output wires. For $\mathtt{T} = (0,0)$ it is the output of a `rescon` gate, for $\mathtt{T} = (1,0)$ output of a `bdcon` gate and for $\mathtt{T} = (0,1)$ it is a `copycon` gate. The reason to add such variables rather than simply relying on $\mathtt{Z}$ to identify a gate is that it is simpler to write the MLEs of each different gate using the `eq` predicate, all while being able to group all of them into a single sumcheck for what should be $\tilde{V}_0(\mathtt{T}, \mathtt{Z}) = 0$. Given the presence of $\mathtt{T}$, we can limit the length of the vector $\mathtt{Z}$: It only has as many variables as the *maximum* among the three kinds of gates: $\mathtt{Z} = (\mathtt{Z}_0, \ldots, \mathtt{Z}_{\max(\mu,\mu-1,\#(copy))-1})$.

Let $d = \#(V_{3,ptxt})$, let $m = \max(2\delta, d)$ and let $m' = 1 + \max(\mu, \#(in))$. We apply the $\mathtt{X} \to \mathtt{Z}$ optimization to all variables except the δ ones indexing the HE level of the layer-3 input wire, which we read from $\mathtt{X}_{[\delta]}$. Eq. (7) provides a sumcheck-style consistency check for the output layer $\tilde{V}_0$. Later on, Equations (8), (9), (10) and (30) describe the MLEs of its wiring predicates.

$$
\begin{aligned}
0 = \tilde{V}_0(\mathtt{T}, \mathtt{Z}) = {} & \widetilde{\mathtt{bdcon}}_2(\mathtt{T}) \cdot \tilde{V}_{2,d_2}(\mathtt{Z}_{[\mu-1]}) + \widetilde{\mathtt{rescon}}_1(\mathtt{T}) \cdot \tilde{V}_1(\mathtt{Z}_{[\mu]}) \\
& - \widetilde{\mathtt{copycon}}_{3,copy}(\mathtt{T}) \cdot \tilde{V}_{3,copy}(\mathtt{Z}_{[\#(copy)]}) \\
& + \sum_{\mathbf{x} \in \{0,1\}^m} \Big(\widetilde{\mathtt{bdcon}}_3(\mathtt{T}, \mathtt{Z}_{[\delta]}, \mathbf{x}) \cdot \tilde{V}_{3,BD}(\mathbf{x}_{[\delta]}, \mathtt{Z}_{[\delta,\mu-1)}, \mathbf{x}_{[\delta,2\delta)}) \\
& + \widetilde{\mathtt{rescon}}_3(\mathtt{T}, \mathtt{Z}_{[\delta]}, \mathbf{x}) \cdot \tilde{V}_{3,rc_3}(\mathbf{x}_{[\delta]}, \mathtt{Z}_{[\delta,\mu)}, x_\delta) \\
& + \sum_{\mathbf{y} \in \{0,1\}^{m'}} \Big(\widetilde{\mathtt{copycon}}(\mathtt{T}, \mathtt{Z}_{[\#(copy)]}, \mathbf{x}, \mathbf{y}) \cdot f(\mathbf{x}_{[d]}, \mathbf{y}) \Big) \Big),
\end{aligned}
\tag{7}
$$

where $f(\mathtt{X}_{[d]}, \mathtt{Y}) = \tilde{V}_{3,ptxt}(\mathtt{X}_{[d]}) \cdot g(\mathtt{Y})$ and $g(\mathtt{Y}) = (1 - \mathtt{Y}_{m'-1}) \cdot \tilde{V}_{in}(\mathtt{Y}_{[\#(in)]}) + \mathtt{Y}_{m'-1} \cdot \tilde{V}_{3,quot}(\mathtt{Y}_{[\mu]})$. One important further improvement that we do just for this layer consistency equation is that we *also truncate* $\mathtt{Z}$ *on the right hand side of Eq. (7)*. This improves efficiency without introducing any new attacks (see Appendix E.3).

Notice how the $\mathtt{X} \to \mathtt{Z}$ optimization is applied to the wire-indexing variables of $\tilde{V}_{3,BD}$ and $\tilde{V}_{3,rc_3}$ (namely $\mathtt{Z}_{[\delta,\mu-1)}$ and $\mathtt{Z}_{[\delta,\mu)}$), which the corresponding predicates constrain only through an equality factor, but *not* to their HE-level variables, which are read from $\mathtt{X}_{[\delta]}$. The three predicates that carry no $\mathtt{X}$ at all, $\widetilde{\mathtt{bdcon}}_2, \widetilde{\mathtt{rescon}}_1$ and $\widetilde{\mathtt{copycon}}_{3,copy}$, are trivially compatible with the substitution and their terms are therefore left outside the sum. Every summand of Eq. (7) is then multilinear in $\mathtt{Z}$, so the equation is an identity of polynomials and can be evaluated at the random point at which Π_{AC} invokes it. Next, we explain the MLEs of each individual wiring predicate.

Rescaling consistency $\widetilde{\mathtt{rescon}}_1(\mathtt{T})$ is trivial. The description of $\widetilde{\mathtt{rescon}}_3$ is a bit complicated, since it takes multiple values. Notice that half of the inputs in layer

3 are quotients (which we will select when $\mathtt{X}_\delta = 1$, and simply multiply by the suitable p_l) and the other half are remainders (selected when $\mathtt{X}_\delta = 0$). Since the first δ variables in $\mathtt{X}$ hold the HE level of the input wire, we use them to select whether a particular input wire to the $\mathbf{z}$-th rescon gate (which holds a quotient value, i.e. with $\mathtt{X}_\delta = 1$) has to be multiplied by $-p_l$, or $-p_2$ etc. Further multiplying $\widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{X}_{[\delta]})$ with $\widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{Z}_{[\delta]})$ forces the level of the input wire to agree with the level of the rescon gate.

$$\begin{aligned} \widetilde{\mathtt{rescon}}_1(\mathtt{T}) &= \widetilde{\mathsf{eq}}(\mathtt{T}, (0,0)) \\ \widetilde{\mathtt{rescon}}_3(\mathtt{T}, \mathtt{Z}_{[\delta]}, \mathtt{X}) &= -\widetilde{\mathsf{eq}}(\mathtt{T}, \mathbf{0}) \cdot \widetilde{\mathsf{eq}}(\mathtt{X}_{[\delta+1,m)}, \mathbf{1}) \cdot \\ &\qquad \Big(\sum_{l=0}^{D-1} \widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{Z}_{[\delta]}) \cdot \widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{X}_{[\delta]}) \cdot \big(1 - \mathtt{X}_\delta + \mathtt{X}_\delta\, p_l\big) \Big). \end{aligned} \tag{8}$$

Base decomposition consistency We define $\widetilde{\mathtt{bdcon}}_3$ similarly to $\widetilde{\mathtt{rescon}}_3$.

$$\begin{aligned} \widetilde{\mathtt{bdcon}}_2(\mathtt{T}) &= \widetilde{\mathsf{eq}}(\mathtt{T}, (1,0)) \\ \widetilde{\mathtt{bdcon}}_3(\mathtt{T}, \mathtt{Z}_{[\delta]}, \mathtt{X}) &= -\widetilde{\mathsf{eq}}(\mathtt{T}, (1,0)) \cdot \widetilde{\mathsf{eq}}(\mathtt{X}_{[2\delta,m)}, \mathbf{1}) \cdot \\ &\qquad \Big(\sum_{l=0}^{D-1} \widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{Z}_{[\delta]}) \cdot \widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{X}_{[\delta]}) \cdot \\ &\qquad\quad \big(\sum_{i=0}^{l} \widetilde{\mathsf{eq}}(\mathsf{bin}(i), \mathtt{X}_{[\delta,2\delta)}) \cdot \mathsf{PW}_{\omega_l}(1)[i] \big) \Big). \end{aligned} \tag{9}$$

Copying values consistency The $\widetilde{\mathtt{copycon}}(\mathbf{t}, \mathbf{z}, \mathbf{x}, \mathbf{y})$ MLE selects values of $\mathbf{x}, \mathbf{y}$ such that, for a given $\mathbf{z}$ it holds that $\sum_{\mathbf{x},\mathbf{y}} ptxt[\mathbf{x}] \cdot ctxt[\mathbf{y}] = ctxt'[\mathbf{z}]$, where $ctxt'[\mathbf{z}]$ is the copied result. All plaintexts are stored in $\tilde{V}_{3,ptxt}(\mathbf{x}_{[d]})$ (for which the verifier has oracle access). The variable $\mathtt{Y}_{m'-1}$ chooses whether $ctxt$ comes from $\tilde{V}_{in}(\mathbf{y}_{[\#(in)]})$ (known to the verifier) or $\tilde{V}_{3,quot}(\mathbf{y}_{[\mu]})$ (prover's oracle). $\tilde{V}_{3,copy}(\mathtt{Z}_{[\#(copy)]})$ stores $ctxt'[\mathbf{z}]$. Since there is one $ctxt'[\mathbf{z}]$ copy per check and $\mathtt{copycon}(\mathbf{t}, \mathbf{z}, \cdot, \cdot)$ is satisfied by as many pairs $(\mathbf{x}, \mathbf{y})$ as the fan-in of that copy gate, the term carrying $\tilde{V}_{3,copy}$ has to sit *outside* the sum over $\mathbf{x}, \mathbf{y}$, for which we use

$$\widetilde{\mathtt{copycon}}_{3,copy}(\mathtt{T}) = \widetilde{\mathsf{eq}}(\mathtt{T}, (0,1)). \tag{10}$$

Note that the choice between $\tilde{V}_{in}$ and $\tilde{V}_{3,quot}$ is made *inside* the sum, by $\mathtt{Y}_{m'-1}$. See Appendix E for the precise definition of the $\mathtt{copycon}$ predicate.

Layer 1: In this layer, let $\mathtt{Z} = (\mathtt{Z}_0, \ldots, \mathtt{Z}_{\mu-1})$. $\tilde{V}_1(\mathtt{Z})$ contains the outputs of key switching. Eq. (11) checks consistency between $\tilde{V}_1(\mathtt{Z})$ and the d_0, d_1 outputs of pre-multiply (stored in $\tilde{V}_{2,d_{01}}(\mathtt{Z})$, in layer 2) mixed with the inner product of the evaluation key and the base decomposition of d_2.

$$\tilde{V}_1(\mathtt{Z}) = \tilde{V}_{2,d_{01}}(\mathtt{Z}) + \sum_{\mathbf{x} \in \{0,1\}^{2\delta}} \Big(\widetilde{\mathtt{evk}}(\mathtt{Z}_{[\delta+1]}, \mathbf{x}) \cdot \tilde{V}_{3,BD}(\mathbf{x}_{[\delta]}, \mathtt{Z}_{(\delta,\mu)}, \mathbf{x}_{[\delta,2\delta)}) \Big). \tag{11}$$

Here too the HE-level variables of $\tilde{V}_{3,BD}$ are kept in $\mathtt{X}$, since $\widetilde{\mathtt{evk}}$ uses the level to pick the evaluation key digits $\mathsf{evk}_{l,b}[i]$ and hence it is not of the form required by the $\mathtt{X} \to \mathtt{Z}$ optimization. The MLE can be thus written as:

$$\widetilde{\mathtt{evk}}(\mathtt{Z},\mathtt{X}) = \sum_{b=0}^{1} \widetilde{\mathsf{eq}}(\mathtt{Z}_\delta, b) \cdot \Big(\sum_{l=0}^{D-1} \widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{Z}_{[\delta]}) \cdot \widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{X}_{[\delta]}) \cdot \Big(\sum_{i=0}^{l} \widetilde{\mathsf{eq}}(\mathsf{bin}(i), \mathtt{X}_{[\delta,2\delta)})\, \mathsf{evk}_{l,b}[i] \Big) \Big).$$

Layer 2: Finally, consistency between the values $\tilde{V}_2(\mathtt{Z})$ on layer 2 and those in layer 3 can be done through the predicate for inner products $\mathtt{Xmult}(\mathbf{z},\mathbf{x},\mathbf{y})$ from [LXZ21]. Let $s' = \max(\#(in), \mu, \#(copy))$. In Eq. (12), we limit the length of the vectors $\mathbf{x},\mathbf{y}$ to s' by having different, specific versions of $\mathtt{Xmult}$ for each kind of product that can occur. $\mathtt{Xmult}_{i,i}$ is an $\mathtt{Xmult}$ between *in* wires, $\mathtt{Xmult}_{q,q}$ between *quot* wires, $\mathtt{Xmult}_{q,c}$ between *quot* and *copy* wires, and so on. In layer 2, $\tilde{V}_2(\mathtt{Z})$ is such that $\mathtt{Z} = (\mathtt{Z}_0, \ldots, \mathtt{Z}_\mu)$ and

$$\begin{aligned}
\tilde{V}_2(\mathtt{Z}) = \sum_{\mathbf{x},\mathbf{y} \in \{0,1\}^{s'}} \Big(& \widetilde{\mathtt{Xmult}}_{i,i}(\mathtt{Z},\mathbf{x},\mathbf{y}) \cdot \tilde{V}_{in}(\mathbf{x}_{[\#(in)]}) \cdot \tilde{V}_{in}(\mathbf{y}_{[\#(in)]}) + \\
& \widetilde{\mathtt{Xmult}}_{q,q}(\mathtt{Z},\mathbf{x},\mathbf{y}) \cdot \tilde{V}_{3,quot}(\mathbf{x}_{[\mu]}) \cdot \tilde{V}_{3,quot}(\mathbf{y}_{[\mu]}) + \\
& \widetilde{\mathtt{Xmult}}_{i,q}(\mathtt{Z},\mathbf{x},\mathbf{y}) \cdot \tilde{V}_{in}(\mathbf{x}_{[\#(in)]}) \cdot \tilde{V}_{3,quot}(\mathbf{y}_{[\mu]}) + \\
& \widetilde{\mathtt{Xmult}}_{i,c}(\mathtt{Z},\mathbf{x},\mathbf{y}) \cdot \tilde{V}_{in}(\mathbf{x}_{[\#(in)]}) \cdot \tilde{V}_{3,copy}(\mathbf{y}_{[\#(copy)]}) + \\
& \widetilde{\mathtt{Xmult}}_{q,c}(\mathtt{Z},\mathbf{x},\mathbf{y}) \cdot \tilde{V}_{3,quot}(\mathbf{x}_{[\mu]}) \cdot \tilde{V}_{3,copy}(\mathbf{y}_{[\#(copy)]}) + \\
& \widetilde{\mathtt{Xmult}}_{c,c}(\mathtt{Z},\mathbf{x},\mathbf{y}) \cdot \tilde{V}_{3,copy}(\mathbf{x}_{[\#(copy)]}) \cdot \tilde{V}_{3,copy}(\mathbf{y}_{[\#(copy)]}) \Big).
\end{aligned} \tag{12}$$

Depending on the HE circuit that is captured by this equation, some of the terms of the sum can be removed if they are known to be zero.

5.2 Progressing through the layers

The Π_{AC} PIOP starts by invoking the sumcheck *protocol* on Eq. (7) and moving towards GKR-layer 3. We describe such series of reductions. For a set of wires S_j, $\tilde{V}_{S_j}(\mathbf{r}_j)$ denotes its MLE evaluations for the random sumcheck challenges $\mathbf{r}_j$.

After the execution of sumcheck for Eq. (7), and in order to reduce to a claim about $\tilde{V}_1(\mathbf{r}_1)$ together with a claim about $\tilde{V}_{2,d_2}(\mathbf{r}_2)$, the verifier needs to check the final claim from the prover by querying oracles $[\![\tilde{V}_{3,BD}]\!], [\![\tilde{V}_{3,rc_3}]\!], [\![\tilde{V}_{3,quot}]\!], [\![\tilde{V}_{3,ptxt}]\!], [\![\tilde{V}_{3,copy}]\!], [\![\widetilde{\mathtt{copycon}}]\!]$ and evaluating the closed form equations of $\widetilde{\mathtt{bdcon}}_2$, $\widetilde{\mathtt{bdcon}}_3, \widetilde{\mathtt{rescon}}_1, \widetilde{\mathtt{rescon}}_3, \widetilde{\mathtt{copycon}}_{3,copy}, \tilde{V}_{in}$ at the relevant challenge points.

The value $\tilde{V}_1(\mathbf{r}_1)$ is plugged into Eq. (11). Completing a sumcheck reduction to $\tilde{V}_{2,d_{01}}(\mathbf{r}_{01})$ requires $\mathcal{V}$ to query $[\![\tilde{V}_{3,BD}]\!]$ and evaluate $\widetilde{\mathtt{evk}}$ at the new challenges.

Claims $\tilde{V}_{2,d_{01}}(\mathbf{r}_{d_{01}})$ and $\tilde{V}_{2,d_2}(\mathbf{r}_{d_2})$ need to be combined into a claim $\tilde{V}_2(\mathbf{r}_2)$. In order to do this, we follow the approach from [CCC+25], which employs sumcheck too. Due to space constraints, we give those details in Appendix E.4. The reduction requires $\mathcal{V}$ to evaluate MLEs $\widetilde{\mathtt{C}}_{2,d_{01}}$, $\widetilde{\mathtt{C}}_{2,d_2}$ at the relevant challenges.

Finally, $\tilde{V}_2(\mathbf{r}_2)$ is substituted in Eq. (12), where for a general circuit $\mathcal{V}$ has to evaluate $\tilde{V}_{in}$ on their own (twice) and query oracles $[\![\mathtt{Xmult}_{i,i}]\!]$, $[\![\mathtt{Xmult}_{q,q}]\!]$, $[\![\mathtt{Xmult}_{i,q}]\!]$, $[\![\mathtt{Xmult}_{i,c}]\!]$, $[\![\mathtt{Xmult}_{q,c}]\!]$, $[\![\mathtt{Xmult}_{c,c}]\!]$ once and oracles $[\![\tilde{V}_{3,quot}]\!]$, $[\![\tilde{V}_{3,copy}]\!]$ twice. If all checks pass, this concludes Π_{AC}.

Notice how the whole protocol requires up to 17 oracle queries for a generic circuit. In Section 8, since we can simplify Eq. (12), we require a lot less.

5.3 Prover algorithms

Our description in Section 5.2 is focused on the verifier. In Appendix E.5, we provide linear time prover algorithms that follow the techniques in [XZZ+19,ZLW+21]. We improve upon them by exploiting our structured sparsity in different forms. One general philosophy is that we always start to challenge from the least significant variables up to the most significant ones. This has positive impacts due to the fact that the prefixes which enable sparsity are placed in the most significant position. We provide more efficient algorithms to update the function evaluation tables according to this challenge ordering rule, specialize for our specific wiring predicates and equations, show how to compute the sumcheck for the product of sparse vectors of different length, etc.

6 Our PIOP for Evaluation of Sparse Polynomials

In this section, we present a PIOP, dubbed SparsePack, for proving the evaluation of a collection of sparse polynomials with coefficients in $\mathbb{Z}_q$. More formally, let us define the indexed relation $\mathcal{R}_{speval}$ over tuples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ such that: the index $\mathtt{i} = ([\![p_0]\!], \ldots, [\![p_{a-1}]\!])$ consists of a multilinear polynomials in m variables such that each p_i evaluates to a non-zero $\mathbb{Z}_q$-value in at most 2^ℓ locations of $\{0,1\}^m$; the statement $\mathtt{x} = (\boldsymbol{x}_0, y_0, \ldots, \boldsymbol{x}_{a-1}, y_{a-1})$ consists of a evaluation points in R_q^m, and alleged results $y_i \in R_q$; the witness $\mathtt{w}$ consists of the polynomials $p_0, \ldots, p_{a-1}$. Then the relation is $\mathcal{R}_{speval} := \{(\mathtt{i}, \mathtt{x}, \mathtt{w}) : \forall i \in [a]\ p_i(\boldsymbol{x}_i) = y_i\}$.

The SparsePack PIOP is a crucial building block of the PIOP for $\mathcal{R}_{AC}$. It enables the prover to convince the verifier about the correct evaluation of the 7 polynomials $\widetilde{\mathtt{copycon}}, \widetilde{\mathtt{Xmult}}_{i,i}, \widetilde{\mathtt{Xmult}}_{q,q}, \widetilde{\mathtt{Xmult}}_{i,q}, \widetilde{\mathtt{Xmult}}_{i,c}, \widetilde{\mathtt{Xmult}}_{q,c}, \widetilde{\mathtt{Xmult}}_{c,c}$ that encode the description of the circuit (see Section 5). For a circuit of size $|C|$ such polynomials have $m \approx 3\log|C|$ variables and $\approx 2^\ell = \mathcal{O}(|C|)$ nonzero evaluations, all of them in $\mathbb{Z}_q$ and most of them being simply 1 as they encode wiring predicates. Naively using a standard polynomial commitment (PC) on these polynomials would incur $\Omega(2^{3\ell})$ proving time, i.e., cubic in the circuit size. An alternative solution is to use a PC for sparse polynomials whose state of the art is represented by Spark [Set20] (as generalized in [STW24]). Despite being designed to work over finite fields, Spark might be extended to work in R_q but

unfortunately the large size of R_q, compared to the usual size of finite fields used in standard SNARKs, reduces its concrete efficiency. We estimated the efficiency of Spark for parameters $N = 2^{13}, \log q \approx 135, \ell = 15, m = 34, a = 2$ that fit one of our neural network applications (cf. Appendix G) and compared it with the PIOP SparsePack that we present in this section.

In a nutshell, the Spark-based solution would have a proof size of about 6.5 GiB while SparsePack yields a 163 MiB proof. The main reason of this gain is that Spark requires the prover to generate at least $a\lceil m/\ell \rceil = 6$ commitments to R_q-polynomials of size 2^ℓ and later prove evaluations of 24 polynomials (6.2 GiB with two batched Basefold proofs over R_q as in [CCC+25]). In contrast, our PIOP *does not require generating any commitment at proving time* and it only requires the prover to compute a *single* evaluation proof for a polynomial commitment computed during preprocessing. Notably, by using our packing techniques, this committed preprocessed polynomial can be of size $2^{\ell+\log m+\log a+2-\log N} = 2^{11}$, much smaller than in Spark. The size of this evaluation proof is 163 MiB with BatchFold. Note that, in terms of computation time, we also eliminate the need to commit to polynomials at proving time, which in Spark would take about 10 minutes of work. Finally, we also obtain another gain in the size of the information-theoretic part of the proof, which with Spark would be 235 MiB, while in SparsePack it can be as low as 25 KiB. We provide more details of this comparison in Section 8 and Appendix G.

6.1 Overview of the SparsePack PIOP

In this section we explain the main ideas of our SparsePack PIOP, and refer to Appendix G for its full description, security proof and efficiency analysis.

Sparse encoding Let $S = \{s_0, \ldots, s_{2^\ell-1}\} \subset \{0,1\}^m$ be a subset of cardinality 2^ℓ such that $\{\boldsymbol{k} \in \{0,1\}^m : p(\boldsymbol{k}) \neq 0\} \subseteq S$. From now on we assume that $m = 2^\mu$ is a power of 2 (otherwise we can generalize the arguments below by appropriate padding of zeros). Let $\mathbf{S} \in \{0,1\}^{2^\ell \times 2^\mu}$ be the matrix with rows indexed by elements in $\{0,1\}^\ell$ and whose $\boldsymbol{b}$-th row contains $s_{\mathsf{to\text{-}int}(\boldsymbol{b})} \in S \subset \{0,1\}^m$. Then a sparse p can be represented with two multilinear polynomials val_p and ind_p where ind_p is the MLE of the above matrix $\mathbf{S}$ and val_p is such that

$$\forall \boldsymbol{b} \in \{0,1\}^\ell : \mathsf{val}_p(\boldsymbol{b}) = p(\mathsf{ind}_p(\boldsymbol{b}, \mathbf{0}), \ldots, \mathsf{ind}_p(\boldsymbol{b}, \mathbf{1})).$$

For a polynomials $p_0, \ldots, p_{a-1}$, each assumed to have up to 2^ℓ nonzero evaluations, we can define (larger) polynomials $\mathsf{val}, \mathsf{ind}$ containing all their sparse encodings such that, denoting $\alpha = \lceil \log a \rceil$ and setting $p_i := 0$ for $a \leq i < 2^\alpha$,

$$\forall (\boldsymbol{b}, \boldsymbol{i}) \in \{0,1\}^{\ell+\alpha} : \mathsf{val}(\boldsymbol{b}, \boldsymbol{i}) = p_i(\mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \mathbf{0}), \ldots, \mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \mathbf{1})).$$

Ideas of the proof. To prove $(\forall i \in [a],\ p_i(\boldsymbol{x}_i) = y_i)$, let $\tilde{y} \in R_q^{(\leq 1)}[Z_1, \ldots, Z_\alpha]$ be the MLE of the vector $(y_0, \ldots, y_{a-1})$ and let $\tilde{x} \in R_q^{(\leq 1)}[Z_1, \ldots, Z_{\alpha+\mu}]$ be the

MLE of the vector of inputs $(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{a-1})$, i.e., such that $\tilde{x}(\boldsymbol{i}, \boldsymbol{j}) = (\boldsymbol{x}_i)_j$. Then we can express the statement to be proven as

$$\begin{aligned}\tilde{y}(\boldsymbol{Z}) &= \sum_{\boldsymbol{i}\in\{0,1\}^{\alpha}} p_i(\boldsymbol{x}_i)\tilde{\mathsf{eq}}(\boldsymbol{i}, \boldsymbol{Z}) = \sum_{\boldsymbol{i}\in\{0,1\}^{\alpha}, \boldsymbol{k}\in\{0,1\}^{m}} p_i(\boldsymbol{k})\tilde{\mathsf{eq}}(\boldsymbol{i}, \boldsymbol{Z})\tilde{\mathsf{eq}}(\boldsymbol{k}, \boldsymbol{x}_i) \\ &= \sum_{(\boldsymbol{b},\boldsymbol{i})\in\{0,1\}^{\ell+\alpha}} \mathsf{val}(\boldsymbol{b}, \boldsymbol{i})\tilde{\mathsf{eq}}(\boldsymbol{i}, \boldsymbol{Z}) \prod_{\boldsymbol{j}\in\{0,1\}^{\mu}} \tilde{\mathsf{eq}}(\mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}), \tilde{x}(\boldsymbol{i}, \boldsymbol{j})). \qquad (13)\end{aligned}$$

Using standard techniques, we can test this polynomial equation on a verifier's random challenge $\boldsymbol{t} \leftarrow\$ \hat{S}_N^{\alpha}$, reducing the problem to proving

$$\tilde{y}(\boldsymbol{t}) = \sum_{(\boldsymbol{b},\boldsymbol{i})\in\{0,1\}^{\ell+\alpha}} \mathsf{val}(\boldsymbol{b}, \boldsymbol{i})\tilde{\mathsf{eq}}(\boldsymbol{i}, \boldsymbol{t}) \prod_{\boldsymbol{j}\in\{0,1\}^{\mu}} \tilde{\mathsf{eq}}(\mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}), \tilde{x}(\boldsymbol{i}, \boldsymbol{j})). \qquad (14)$$

The statement above is a highly structured computation which involves an inner product and $2^{\ell+\alpha}$ trees of multiplications of depth μ, performed in parallel. Therefore, we present SparsePack generically as a family of protocols parametrized by a PIOP for proving a statement like that of Equation (14). To instantiate this PIOP we considered two options—Thaler's multiplication protocol [Tha13] and a parallel version of Quark's grand product [SL20]. In SparsePack, we adopted the former as it avoids creating fresh commitments during the proof. A straightforward use would be expensive due to computing μ sequential sumchecks and due to the need, in the end, to perform an evaluation proof on the committed polynomials describing the sparse encoding, of size $am2^{\ell}$. In SparsePack we leverage the power of our packing technique to reduce these costs.

More formally, for an integer β, let Π_{iptree} be a PIOP for the oracle relation

$$\mathcal{R}_{iptree} = \{(v, [\![f_1]\!], [\![f_2]\!], [\![E]\!]) : v = \sum_{(\boldsymbol{b},\boldsymbol{i})\in\{0,1\}^{\ell+\alpha}} f_1(\boldsymbol{b}, \boldsymbol{i})f_2(\boldsymbol{i}) \prod_{\boldsymbol{j}\in\{0,1\}^{\beta}} E(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j})\},$$

and in which the verifier makes one query to each oracle polynomial: $f_1(\boldsymbol{r}_1, \boldsymbol{r}'_1)$, $f_2(\boldsymbol{r}_2)$, $E(\boldsymbol{r}, \boldsymbol{s})$. It is easy to see how to use Π_{iptree} to prove Equation (14) by running it on $(\tilde{y}(\boldsymbol{t}), [\![\mathsf{val}]\!], [\![\tilde{\mathsf{eq}}(\boldsymbol{X}, \boldsymbol{t})]\!], [\![E_\mu]\!])$ where, for $\boldsymbol{Z} = (Z_1, \ldots, Z_{\alpha+\mu})$, $E_\mu(\boldsymbol{X}, \boldsymbol{Z}) = \tilde{\mathsf{eq}}(\mathsf{ind}(\boldsymbol{X}, \boldsymbol{Z}), \tilde{x}(\boldsymbol{Z}))$, which is multilinear in $\boldsymbol{X}$ and of degree 2 in $\boldsymbol{Z}$.

We improve on this solution through the following techniques.

Packing and Preprocessing. The polynomials $\mathsf{val}, \mathsf{ind}$ have all coefficients in $\mathbb{Z}_q$. Thus, assuming that $\alpha+\mu+2 \leq \nu = \log N$, which for our values of $\alpha = 3$ and $\log N \approx 13$ would be satisfied for circuits of size up to $\approx 2^{2^{\nu-5}/3} \approx 2^{85}$, we can pack $\mathsf{val}, \mathsf{ind}$ into a single polynomial $\overline{\mathsf{pol}}$ of size 2^{ℓ}. However, since we have space to pack even more information in $\overline{\mathsf{pol}}$, we leverage that the p_i polynomials are part of the index (thus they are not adversarially generated) to create additional polynomials in a deterministic preprocessing that allows us to reduce the depth of the multiplication tree in (14).

We do this as follows: we observe that the product in Equation (14) can be also written as $\prod_{\boldsymbol{j}\in\{0,1\}^{\mu-1}} E_\mu(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 0)E_\mu(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 1)$. Let $\boldsymbol{Z} = (Z_1, \ldots, Z_{\alpha+\mu-1})$.

While running Π_{iptree} with the polynomial $\hat{E}_\mu(\boldsymbol{X}, \boldsymbol{Z}) = E_\mu(\boldsymbol{X}, \boldsymbol{Z}, 0) \cdot E_\mu(\boldsymbol{X}, \boldsymbol{Z}, 1)$ would make the tree of smaller depth, the degree of $\hat{E}_\mu$ would double in all variables and proving its evaluation would vanish the gain in depth.

Instead, we use preprocessing to create an additional polynomial that allows us to replace the product of (14) by $\prod_{\boldsymbol{j} \in \{0,1\}^{\mu-1}} E^*_{\mu-1}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j})$, where $E^*_{\mu-1}$ has degree 1 in $\boldsymbol{X}$ and degree 2 in $\boldsymbol{Z}$.

To do this, let ind^* be the MLE of $(\mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 0) \cdot \mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 1) : \forall \boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j})$, and let $\tilde{x}^*$ be the MLE of $\tilde{x}(\boldsymbol{i}, \boldsymbol{j}, 0)\tilde{x}(\boldsymbol{i}, \boldsymbol{j}, 1)$. Let us define the following polynomials

$$\begin{aligned}
\tilde{x}^*_0(\boldsymbol{Z}) &= -2 \cdot \tilde{x}^*(\boldsymbol{Z}) + 2 \cdot \tilde{x}(\boldsymbol{Z}, 0) + \tilde{x}(\boldsymbol{Z}, 1) - 1 \\
\tilde{x}^*_1(\boldsymbol{Z}) &= -2 \cdot \tilde{x}^*(\boldsymbol{Z}) + \tilde{x}(\boldsymbol{Z}, 0) + 2 \cdot \tilde{x}(\boldsymbol{Z}, 1) - 1 \\
\tilde{x}^*_2(\boldsymbol{Z}) &= 4 \cdot \tilde{x}^*(\boldsymbol{Z}) - 2 \cdot \tilde{x}(\boldsymbol{Z}, 0) - 2 \cdot \tilde{x}(\boldsymbol{Z}, 1) + 1 \\
\tilde{x}^*_3(\boldsymbol{Z}) &= \tilde{x}^*(\boldsymbol{Z}) - \tilde{x}(\boldsymbol{Z}, 0) - \tilde{x}(\boldsymbol{Z}, 1) + 1 \\
E^*_{\mu-1}(\boldsymbol{X}, \boldsymbol{Z}) &= \mathsf{ind}(\boldsymbol{X}, \boldsymbol{Z}, 0)\tilde{x}^*_0(\boldsymbol{Z}) + \mathsf{ind}(\boldsymbol{X}, \boldsymbol{Z}, 1)\tilde{x}^*_1(\boldsymbol{Z}) \\
&\qquad + \mathsf{ind}^*(\boldsymbol{X}, \boldsymbol{Z})\tilde{x}^*_2(\boldsymbol{Z}) + \tilde{x}^*_3(\boldsymbol{Z}).
\end{aligned} \tag{15}$$

By construction, it holds $\forall \boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}$: $E^*_{\mu-1}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}) = E_\mu(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 0)E_\mu(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 1)$. Therefore, in $\mathsf{SparsePack}$ we run Π_{iptree} with the virtual oracle $[\![E^*_{\mu-1}]\!]$; this ends with a verifier's query $v = E^*_{\mu-1}(\boldsymbol{r}, \boldsymbol{s})$, with $\boldsymbol{r} \in \hat{S}^\ell_N, \boldsymbol{s} \in \hat{S}^{\alpha+\mu-1}_N$. Interestingly, answering this query requires no additional sumcheck: by Equation (15), $E^*_{\mu-1}$ is a linear combination of $\mathsf{ind}(\boldsymbol{X}, \boldsymbol{Z}, 0)$, $\mathsf{ind}(\boldsymbol{X}, \boldsymbol{Z}, 1)$ and $\mathsf{ind}^*(\boldsymbol{X}, \boldsymbol{Z})$ whose coefficients $\tilde{x}^*_i(\boldsymbol{Z})$ depend only on the public inputs. Hence the verifier computes the values $\tilde{x}^*_i(\boldsymbol{s})$ on its own and obtains v from the three queries $\mathsf{ind}(\boldsymbol{r}, \boldsymbol{s}, 0)$, $\mathsf{ind}(\boldsymbol{r}, \boldsymbol{s}, 1)$, and $\mathsf{ind}^*(\boldsymbol{r}, \boldsymbol{s})$.

In $\mathsf{SparsePack}$, we create a polynomial pol of $\ell + \alpha + \mu + 2$ variables that puts together $\mathsf{val}, \mathsf{ind}$ and ind^*, but then we pack pol into a polynomial $\overline{\mathsf{pol}}$ with only $n = (\ell + \alpha + \mu + 2 - \nu)$ variables. In particular, the three queries above become evaluations of pol on points that differ only in two coordinates, and we batch them into a single query to pol using two random challenges. We refer to Appendix G for the details.

Evaluating the (un)packed polynomial. The solution sketched above requires querying the unpacked $[\![\mathsf{pol}]\!]$ on points in R_q, so we cannot use Lemma 3 to compute them from queries to $[\![\overline{\mathsf{pol}}]\!]$. The last piece of the protocol, which solves this issue, is again a tool of our packing technique, namely our PIOP Π_{ueval} of Section 3.2 for proving evaluations of unpacked polynomials on R_q points.

7 BatchFold

In this section, we describe our polynomial commitment (PC) scheme to elements in $R_q[\boldsymbol{X}]$, which instantiate the PIOP oracles. For efficiency, our goal is to allow for batched openings of the form $f_m(\boldsymbol{z}_m) = y_m$, for $m \in [M]$, at a smaller cost sublinear in M. We do this in two steps: we first reduce this question to constructing PC schemes over *finite fields* with batched openings of polynomials f_m *at a common point* $\boldsymbol{z}$. Then we instantiate the latter from BaseFold [ZCF24].

7.1 Batching Openings via CRT and HyperPlonk

Recall that $R_q \cong \prod_{i=0}^{L} \prod_{j=0}^{k-1} R_{i,j}$, where each $R_{i,j}$ is a finite field of size p_i^d and $d = N/k$ (in practice, $d = 4$). Let $\Phi : R_q \to \prod_{i=0}^{L} \prod_{j=0}^{k-1} R_{i,j}$ denote the aforementioned isomorphism and $\Phi_{ij} : R_q \to R_{i,j}$ its components. We extend Φ and Φ_{ij} to R_q^ℓ by applying them coordinate-wise, and to $R_q[\boldsymbol{X}]$ coefficient-wise.

In [CCC+25], given polynomial commitments $\mathsf{PC}_{R_{i,j}}$ for each $R_{i,j}[X_1, \ldots, X_\ell]$, the authors construct a unified PC_{R_q} for $R_q[X_1, \ldots, X_\ell]$. This boils down to committing to each CRT component, i.e., $\mathsf{PC}_{R_q}.\mathsf{Com}(f) := (\mathsf{PC}_{R_{i,j}}.\mathsf{Com}(\Phi_{ij}(f)))_{i,j}$, and the evaluation function (for single openings) $\mathsf{PC}_{R_q}.\mathsf{Eval}$ shows $f(\boldsymbol{z}) = y$ by running each $\mathsf{PC}_{R_{i,j}}.\mathsf{Eval}$ on the claim $\Phi_{ij}(f)(\Phi_{ij}(\boldsymbol{z})) = \Phi_{ij}(y)$.

Our first (elementary) observation is that this can be extended to more general Eval_M protocols for batches of M evaluations: to prove $\{f_m(\boldsymbol{z}_m) = y_m : m \in [M]\}$, apply each $\mathsf{PC}_{R_{i,j}}.\mathsf{Eval}_M$ on $\{\Phi_{ij}(f_m)(\Phi_{ij}(z_m)) = \Phi_{ij}(y_m) : m \in [M]\}$.

Batching on isomorphic components. More interestingly, we can use the observation that for every $i \in [L+1]$, the fields $\{R_{i,j}\}_{j=0}^{k-1}$ are isomorphic to each other. Therefore, all $\{\mathsf{PC}_{R_{i,j}}\}_{j=0}^{k-1}$ can be captured by a single scheme PC_i. Then, if each PC_i is endowed with a $\mathsf{PC}_i.\mathsf{Eval}_{kM}$ (batching kM openings) we can construct $\mathsf{PC}_{R_q}.\mathsf{Eval}_M$ (batching M openings) that only uses each $\mathsf{PC}_i.\mathsf{Eval}_{kM}$ *once*. Indeed, to prove $\{f_m(\boldsymbol{z}_m) = y_m : m \in [M]\}$, apply $\mathsf{PC}_i.\mathsf{Eval}_{kM}$ to prove the claim $\{\Phi_{ij}(f)(\Phi_{ij}(\boldsymbol{z}_m)) = \Phi_{ij}(y_m) : j \in [k], m \in [M]\}$, for every $i \in [L+1]$. This amounts to $L+1$ batched evaluations proofs, rather than $k(L+1)$ (if we were to apply $\mathsf{PC}_i.\mathsf{Eval}_M$ k times) and is beneficial given that one execution $\mathsf{PC}_i.\mathsf{Eval}_{kM}$ is cheaper than k executions of $\mathsf{PC}_i.\mathsf{Eval}_M$, as we will see.

Reducing to evaluations on the same point. We now reduce the task of constructing Eval_M to creating a protocol $\mathsf{Eval}_M^{\mathsf{com}}$ that proves batched opening *at a common evaluation point*, i.e. $\mathsf{Eval}_M^{\mathsf{com}}$ proves claims of the form $\{f_m(\boldsymbol{z}') = y'_m : m \in [M]\}$ for a common $\boldsymbol{z}'$. In Figure 3, we construct $\mathsf{PC}.\mathsf{Eval}_M$ from $\mathsf{PC}.\mathsf{Eval}_M^{\mathsf{com}}$ by using an idea from HyperPlonk [CBBZ23]. The overhead of this protocol is an additional sumcheck over $\{0,1\}^\ell$. The soundness error of Eval_M is at most $\frac{2\ell+M-1}{|\mathbb{F}|} + \mathsf{error}_{\mathsf{Eval}_M^{\mathsf{com}}}$ where $\mathsf{error}_{\mathsf{Eval}_M^{\mathsf{com}}}$ is the soundness error of $\mathsf{Eval}_M^{\mathsf{com}}$, and $\mathbb{F}$ is the field over which PC is defined (in our case, for each PC_i, $\mathbb{F}$ is of size p_i^d).

7.2 Batching BaseFold openings at a common point

BaseFold. We summarize the relevant features of BaseFold. A detailed description can be found in Appendix H. The commitment to a polynomial f is given by an oracle π_d (in practice, a vector commitment) to an encoding of its coefficient vector with certain linear error-correcting code C_d, of dimension 2^ℓ (the number of coefficients of f). The protocol $\mathsf{PC}.\mathsf{Eval}$ for a single opening $f(\boldsymbol{z}) = y$ consists of a sumcheck for $\sum_{\boldsymbol{b} \in \{0,1\}^\ell} f(\boldsymbol{b})\tilde{\mathsf{eq}}(\boldsymbol{b}, \boldsymbol{z}) = y$, and a consistency check for the fact that π_d is indeed an oracle to an element in C_d and is consistent with the sumcheck. The soundness of the evaluation protocol of PC depends on

Construction of $\mathsf{PC.Eval}_M$ from $\mathsf{PC.Eval}^{\mathsf{com}}_M$

1. The verifier sends $s \leftarrow\!\$\ \mathbb{F}$ to the prover. For every $\boldsymbol{b} \in \{0,1\}^\ell$ the prover defines the value $h_{\boldsymbol{b}} := \sum_{m=0}^{M-1} s^m \cdot \tilde{\mathsf{eq}}(\boldsymbol{b}, \boldsymbol{z}_m) \cdot f_m(\boldsymbol{b})$. Let $\tilde{h}$ be the multilinear extension of the $h_{\boldsymbol{b}}$, i.e. the unique $\tilde{h} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\ell]$ with $\tilde{h}(\boldsymbol{b}) = h_{\boldsymbol{b}}\ \forall \boldsymbol{b} \in \{0,1\}^\ell$.
2. Prover and verifier now engage in a standard sumcheck protocol for the statement $\sum_{\boldsymbol{b}\in\{0,1\}^\ell} \tilde{h}(\boldsymbol{b}) = S$ for $S := \sum_{m=0}^{M-1} s^m \cdot y_m$ where, when the verifier queries $\tilde{h}(\boldsymbol{r})$ for some $\boldsymbol{r} \leftarrow\!\$\ \mathbb{F}^\ell$:
   (a) The prover opens $\{q_m = f_m(\boldsymbol{r}), m \in [M]\}$ with $\mathsf{PC.Eval}^{\mathsf{com}}_M$.
   (b) If this is correct, the verifier computes $\tilde{h}(\boldsymbol{r}) = \sum_{m=0}^{M-1} s^m \cdot \tilde{\mathsf{eq}}(\boldsymbol{r}, \boldsymbol{z}_m) \cdot q_m$

   The verifier accepts if the sumcheck (and the $\mathsf{PC.Eval}^{\mathsf{com}}_M$ opening) passes.

Figure 3: Construction of $\mathsf{PC.Eval}_M$ from $\mathsf{PC.Eval}^{\mathsf{com}}_M$

$\mathsf{PC.Eval}^{\mathsf{com}}_M$: Batched opening proof at common point $\boldsymbol{z}$

1. The verifier sends $s \leftarrow\!\$\ \mathbb{F}$. The prover sets $u(\boldsymbol{X}) := \sum_{m=0}^{M-1} s^m f_m(\boldsymbol{X})$. This implicitly defines $\pi_d := \pi_{d,u} = \sum_{m=0}^{M-1} s^m \pi_{d,f_m}$.
2. The prover and verifier compute $S = \sum_{m=0}^{M-1} s^m y_m$ and engage in $\mathsf{PC.Eval}$ for the claim: "π_d oracle to $\mathsf{Enc}_d(\boldsymbol{u})$ such that $u(\boldsymbol{z}) = S$". When the verifier queries π_u at a coordinate μ, this is answered with the answers a_m to queries of π_{d,f_m} at μ, for all $m \in [M]$. The verifier can then compute the answer as $\sum_{m=0}^{M-1} s^m a_m$.

Figure 4: Batched opening of M evaluations at common point.

the minimum distance of the code C_d and can be found in Appendix H, where we show an slight improvement with respect to the results in [ZCF24].

Batching at the same point. Our construction for $\mathsf{PC.Eval}^{\mathsf{com}}_M$ is given in Figure 4 where we leave the $\mathsf{PC.Eval}$ of Basefold implicit. In Appendix H, Figure 12 we provide the full protocol $\mathsf{PC.Eval}^{\mathsf{com}}_M$ including $\mathsf{PC.Eval}$. Here we point out that we reduce $\mathsf{PC.Eval}^{\mathsf{com}}_M$ to applying $\mathsf{PC.Eval}$ on a linear combination $u = \sum_{m=0}^{M-1} s^m f_m$, where queries to u are obtained by querying all f_m's; since other than making these queries and computing the linear combination of the results, the rest of $\mathsf{PC.Eval}^{\mathsf{com}}_M$ is independent of M (it is essentially a single application of $\mathsf{PC.Eval}$), we obtain an advantage over repeating $\mathsf{PC.Eval}$ M times.

We have the following soundness (see Appendix H, Lemma 9).

Lemma 5. *The soundness error of* $\mathsf{PC.Eval}^{\mathsf{com}}_M$ *is at most* $\frac{M-1}{|\mathbb{F}|} + \mathsf{error}_{\mathsf{PC.Eval}}$ *where* $\mathsf{error}_{\mathsf{PC.Eval}}$ *denotes the soundness error of* $\mathsf{PC.Eval}$.

As for communication, $\mathsf{PC.Eval}^{\mathsf{com}}_M$ has size $\mathcal{O}_\lambda(M\ell + \ell^2)$ (for security parameter λ), while naively repeating M times $\mathsf{PC.Eval}$ would yield $\mathcal{O}_\lambda(M\ell^2)$ communication. See Appendix H for more precise statements about proof sizes.

Putting everything together. We have shown that the BaseFold commitment PC with individual openings Eval can be compiled into a protocol PC_{R_q} for polynomials in $R_q[\boldsymbol{X}]$. The commitment itself consists in committing to each component $\Phi_{ij}(f)$ with Basefold over R_{ij}. A batched opening Eval_M of M evaluations

at arbitrary points in R_q can be obtained from $L+1$ executions of $\mathsf{PC.Eval}^{\mathsf{com}}_{Mk}$ (each in a field R_{ij}) and $L+1$ sumchecks needed in the reduction from $\mathsf{Eval}^{\mathsf{com}}_{Mk}$ to Eval_{Mk}, as shown in the diagram below (where the second arrow is Figure 3 and the first one was explained in *Batching on isomorphic components*):

$$\boxed{\mathsf{PC}_{R_q}.\mathsf{Eval}_M} \leftarrow \boxed{\{\mathsf{PC}_i.\mathsf{Eval}_{Mk}\}_{i\in[L+1]}} \leftarrow \boxed{\{\mathsf{PC}_i.\mathsf{Eval}^{com}_{Mk} + 1 \text{ sumcheck}\}_{i\in[L+1]}}$$

The soundness error is $(L+1)(\frac{2\ell+Mk-1}{\min\{p_i\}^d} + \mathsf{error}_{\mathsf{PC.Eval}})$, where $\mathsf{error}_{\mathsf{PC.Eval}}$ is an upper bound for the soundness error of Basefold. The communication complexity is $\mathcal{O}_\lambda(MkL\ell + L\ell^2)$. In contrast, using $\mathsf{PC.Eval}$ openings for individual statements in each of the Lk components would yield $\mathcal{O}_\lambda(MkL\ell^2)$ communication.

8 Implementation and Evaluation

We first implemented our proposals over the verifiable HE framework of [CCC+25][10], which we later integrated into the `vfhe` library [CCC+26a] for better performance and reproducibility. This full version of the paper considers the updated implementation, and the code reproducing our results is available at `https://github.com/vfhe/Batch-Pack-and-Prove`. Results for the previous implementation are presented in the printed version [CCC+26b].

We follow a similar approach to [CCC+25] when analyzing performance. In particular, the performance of their approach is dominated by common proof operations, such as the sumcheck and polynomial commitment procedures, which they benchmark over their "proof-friendly" CKKS implementation. For fair comparison, we perform a similar analysis, in the same ring, with the same HE parameters. We run all experiments on an `r7i.metal-48xl` instance on AWS (Intel Xeon 8488C at 3.2GHz).

8.1 BatchFold

Our first improvement over [CCC+25] is the change of PC scheme from Brakedown to Basefold, and in particular to our new BatchFold variant. Basefold commitments are asymptotically more expensive than Brakedown (quasilinear instead of linear) but they have logarithmic-time verification (instead of $\mathcal{O}(\sqrt{2^\ell})$). Thanks to our new techniques (notably packing and the use of H_q), many polynomials we commit to in our protocols are a factor $\mathcal{O}(N)$ smaller than in [CCC+25], allowing the prover to bear the extra costs of quasilinear proving time, in exchange for significantly faster verification. BatchFold improves performance over Basefold not only for the verifier, but also for the prover computing evaluation proofs, as Figure 5 shows. These measurements include the Merkle commitments, where BatchFold also helps, since it hashes significantly less information. Comparisons between plain Basefold and Brakedown can be found in [ZCF24]. As additional contribution, we implement an estimator for computing optimal parameters for BaseFold and BatchFold.

[10] Available at `https://github.com/vfhe/proof-friendly-CKKS` (commit `e44d520`)

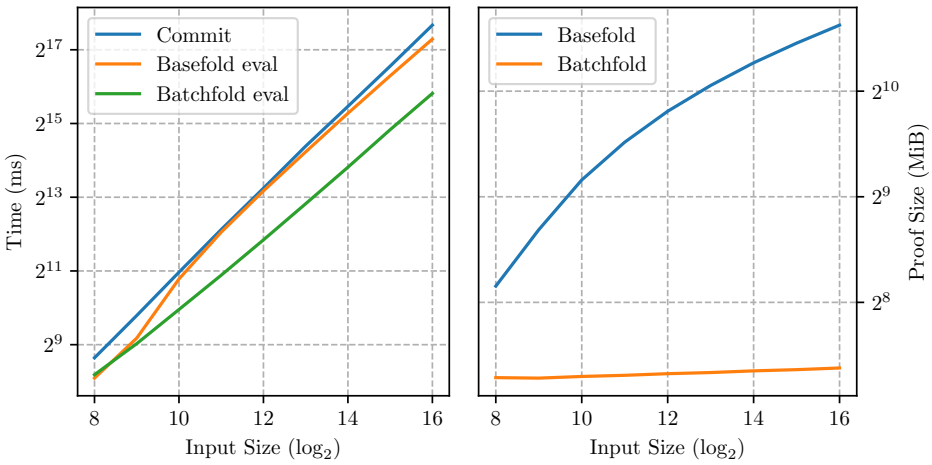


Fig. 5: Basefold [ZCF24] and **Batchfold** (our work) for R_q with $N = 2^{13}$ and 3 RNS components. Time is single-threaded and includes the Merkle hashing. We consider only the batching of $N/4$ components in R_q.

We achieve an up to 2.8× performance improvement upon Basefold for evaluation proofs, while proof size is reduced by up to a factor of 9.5×. Our impact to high level protocols could, however, be much greater than this, as our results are only considering the batching of the $N/4 = 2^{11}$ CRT components of R_q while high level protocols could also batch evaluations of multiple different multivariate polynomials. Compared to [CCC+25], which takes 8.5 seconds per CRT component of R_q (i.e., for each $R_{i,j}$) for input size 2^{16} using the original implementation of Brakedown, we achieve improvements of up to 302×, taking just 57.7 seconds for the same instance. [CCC+25] also presents an improved version of Brakedown, but they do not report proof size and proving time for it. Although potential improvements from the optimized version in [CCC+25] should be taken into consideration, we are only measuring single-threaded time, while [CCC+25] uses up to 48 threads. Therefore, we expect our results and improvements to hold up, improvements to [CCC+25] notwithstanding.

8.2 High-level applications

For the polynomial commitment, we provided a direct comparison between ours and [ZCF24,CCC+25] based on a generic polynomial; however, demonstrating our broader contributions to verifiable computation requires a less generic benchmark. In particular, while [CCC+25] showcases a microbenchmark based on a single CKKS multiplication, we go significantly beyond this and consider small HE applications. We consider 3 small generic neural network models, each composed of 2 fully-connected hidden layers using $f(x) = x^2$ as the activation function. We denote our models by tuples (n_0, n_1), representing the number of neurons in the hidden layers. All models take 30 inputs, and have 2 outputs – thus being suitable in small classification problems (*e.g.*, [WMSS93,K+96]).

We consider batched evaluations using CKKS's SIMD packing [CKKS17] (*i.e.*, $N/2 = 2^{12}$ samples are classified at once) and present the detailed architecture in Appendix I. We note that our goal is to demonstrate the feasibility of our vHE approach, and that machine learning optimization aspects are out of our scope. We compile these small HE applications into our $\mathcal{R}_{CKKS}$ relations – composed of GKR circuits and range checks – and compute precisely the number of wires, gates and commitments needed to evaluate and prove them. From those, we measure the range proofs end to end, and estimate the other components from measurements of their building blocks: we consider the execution times for polynomial commitment operations given in Section 8.1, and we implement and benchmark SumChecks, following [CCC+25].

Arithmetic circuit. In [CCC+25] the performance analysis focuses on the range proof protocol Π_{range}, under the assumption that Π_{AC} and the actual HE and GKR evaluations add negligibly to the overall cost of the protocol. Our estimates confirm that the GKR evaluation costs are indeed very small, taking 21.9, 5.9, and 3.0 seconds for our three circuits, respectively. On the other hand, Π_{AC} should also prove the evaluations of large sparse polynomials that describe the circuit, an aspect overlooked by [CCC+25] and left to be handled through generic techniques. In this work, we improve Π_{AC} in two ways. First, thanks to our refined arithmetization we minimize the numbers of polynomials encoding the circuit structure that have to be proven. Second, we propose the SparsePack PIOP specialized to prove the evaluation of these sparse polynomials. Therefore, in our evaluation below we focus on comparing these two components, sparse commitments and range proofs.

Sparse commitments. We compare the performance of SparsePack with that of using Spark [Set20,STW24]. Our benchmark are the evaluations of the sparse polynomials arising from the HE circuits corresponding to the three neural networks mentioned earlier. Table 2 presents our results. We are able to reduce proof sizes by more than an order of magnitude. Execution time is also improved by factors of 10–23×. A large benefit of our construction is that it does not require any commitments at run time – all our commitments depend only on the circuit and not on the inputs, enabling them to be precomputed. Notice that most of the cost of our solution comes from SumChecks, which could be trivially parallelized. Appendix G further discusses and details the operations required by each solution.

We emphasize that here we are not comparing our solution with an application of Spark [Set20,STW24] to handle the sparse polynomials encoding the circuit wiring in [CCC+25]. Such a comparison would likely show even further improvements for us since the arithmetization of [CCC+25] requires more sparse polynomials (and with more variables) than in our case (see Section 5).

Range proofs. Compared to [CCC+25], our strategy of packing and working over H_q enables improvements of $\mathcal{O}(N/d)$ times in both proof size and performance. We measure execution times of 5m54s, 2m47s, and 1m24s, respectively for each of our three circuits above, with the optimized variant $\hat{\Pi}^*_{range}$ of our range

Table 2: Performance of our SparsePack and comparison with Spark. Notice that our solution does not require execution-time commitments.

Model (n_0, n_1)	Spark					SparsePack			
	`SumCheck`		`Comm.`	`Eval`		`SumCheck`		`Eval`	
	Size	Time	Time	Size	Time	Size	Time	Size	Time
$(128, 64)$	235.3 MiB	6m14s	9m38s	6.2 GiB	6m35s	25.4 KiB	1s	163.0 MiB	2m6s
$(64, 16)$	176.5 MiB	1m33s	2m8s	5.5 GiB	1m37s	17.7 KiB	1s	159.4 MiB	13s
$(32, 16)$	150.3 MiB	47s	58s	5.2 GiB	48s	16.7 KiB	1s	159.4 MiB	13s

proof that takes $H_q = \mathbb{F}_p$ (Fig. 10; parameters $\sigma = 2^{11}$, $m = 15$, $\kappa = 20$, Appendix D.5). Appendix I.2 details our results. It is difficult to precisely estimate the costs for [CCC+25], since their protocol is unlikely to handle the circuits of our benchmark without unrealistic amounts of memory (given the $\mathcal{O}(N)$ overhead of their range proof, discussed in Section 4). Even if possible, [CCC+25] estimates 25 seconds per multiplication, leading to a proof time of at least 1 hour 20 minutes for our largest circuit, which contains 192 multiplications. For the same circuit, our range proof takes less than 6 minutes. Additionally, their results are for multithreaded execution with up to 48 threads, whereas we are reporting only single-threaded execution time. Considering this, our wall-clock improvement of more than 13× would likely translate to more than 100× times in computational time.

Discussion on concrete verifier costs. Our experimental evaluation focuses on the concrete performance of the prover, which is often regarded as the major challenge in verifiable HE with sublinear verification, and leaves the implementation-level analysis of the verifier as future work. The `vfhe` library provides timings for the verification of our building blocks. For example, a Basefold (resp. BatchFold) evaluation proof over R_q verifies in 0.4–1.7 s (resp. 0.2–0.3 s) for 2^8 to 2^{16} entries, and the range proof of Table 3 at its largest size, 2^{24} digits, takes 1.8 s, single-threaded. The full verification of our IOPs, however, would require procedures beyond the verification of the building blocks, and extrapolating these timings is difficult.

We nevertheless remark that, asymptotically, our protocol achieves polylogarithmic verification time in the circuit size, improving over the square-root verification complexity of [CCC+25]. This asymptotic improvement suggests that our approach is fundamentally more scalable and that, for sufficiently large computations, verification costs can be amortized and be faster than trivial solutions, such as recomputing the circuit. Moreover, recomputation is not always the appropriate baseline in applications of verifiable FHE. In some scenarios, the verifier may not have access to the prover's inputs and therefore cannot independently recompute. An example is privacy-preserving ML, where a server holding a proprietary model evaluates encrypted client queries. As a future direction, our protocols could be extended to this scenario by using circuit privacy [Gen09] and a zero-knowledge version of our proof system. We expect this to work based on standard techniques. For the former, one may either rely on noise flood-

ing [Gen09] or recent proposals, such as Differential-Privacy-based circuit privacy for CKKS [HMSS25,KS25], which are concretely more efficient. Regarding ZK, the conversion can be achieved through the zero-knowledge sumcheck of Libra [XZZ+19] and PCs with ZK evaluation arguments.

Acknowledgements We would like to thank the anonymous reviewers of Asiacrypt 2026 for the insightful comments and discussions that allowed us to improve the presentation in Section 2.3 and our results in Section 5. This work is partially funded by the grants PID2022-142290OB-I00, JDC2022-049711-I, JDC2024-055789-I, and CEX2024-001471-M, funded by MCIN/AEI/10.13039/501100011033/ FEDER, UE and ESF+, and it is part of the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001283), and of the VERIFHE project funded by the European Union (GA: 101287502). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

ABMP24. Andreea Alexandru, Ahmad Al Badawi, Daniele Micciancio, and Yuriy Polyakov. Application-aware approximate homomorphic encryption: Configuring FHE for practical use. Cryptology ePrint Archive, Paper 2024/203, 2024.

ACGS23. Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. Cryptology ePrint Archive, Paper 2023/1949, 2023.

AHKS22. Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Amber Sprenkels. Faster Kyber and Dilithium on the Cortex-M4. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security*, pages 853–871, Cham, 2022. Springer International Publishing.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BCFK21. Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12711 of *Lecture Notes in Computer Science*, pages 528–558, Virtual Event, May 10–13, 2021. Springer, Cham, Switzerland.

BFS20. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

CCC+25. Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. Verifiable computation for approximate homomorphic encryption schemes. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part VII*, volume 16006 of *Lecture Notes in Computer Science*, pages 643–677, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

CCC+26a. Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Adrien Dubois, Dario Fiore, Antonio Guimarães, Robin Köstler, Michał Osadnik, Alin-Petru Roșu, and Eduardo Soria-Vazquez. The `vfhe` library. `https://github.com/vfhe/vfhe`, 2026. Commit `769198d`.

CCC+26b. Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. Batch, pack, and prove: More efficient verifiable computation for ckks. In *ASIACRYPT 2026*, 2026.

CCH+19. Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1082–1090, Phoenix, AZ, USA, June 23–26, 2019. ACM Press.

CCKP19. Shuo Chen, Jung Hee Cheon, Dongwoo Kim, and Daejun Park. Verifiable computing for approximate computation. Cryptology ePrint Archive, Report 2019/762, 2019.

CCP+24. Jung Hee Cheon, Hyeongmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. Attacks against the IND-CPA$^{\mathrm{D}}$ security of exact FHE schemes. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 2505–2519, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.

CMS19. Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 1–29, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.

FGP14. Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 844–855, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

FNP20. Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 124–154, Edinburgh, UK, May 4–7, 2020. Springer, Cham, Switzerland.

GBK+24. Mariana Gama, Emad Heydari Beni, Jiayi Kang, Jannik Spiessens, and Frederik Vercauteren. Blind zkSNARKs for private proof delegation and verifiable computation over encrypted data. Cryptology ePrint Archive, Report 2024/1684, 2024.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

GGW24. Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 449–487, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

GKR08. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 113–122, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

GLS+23. Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

GNS23. Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. *Journal of Cryptology*, 36(4):41, October 2023.

GNSJ24. Qian Guo, Denis Nabokov, Elias Suvanto, and Thomas Johansson. Key recovery attacks on approximate homomorphic encryption with non-worst-case noise flooding countermeasures. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

HMSS25. Intak Hwang, Seonhong Min, Jinyeong Seo, and Yongsoo Song. On the security and privacy of CKKS-based homomorphic evaluation protocols. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025, Part VII*, volume 16251 of *Lecture Notes in Computer Science*, pages 296–330, Melbourne, VIC, Australia, December 8–12, 2025. Springer, Singapore, Singapore.

HMZ25. Mi-Ying Miryam Huang, Xinyu Mao, and Jiapeng Zhang. Sublinear proofs over polynomial rings. Cryptology ePrint Archive, Report 2025/199, 2025.

HYS+25. Syed Mahbub Hafiz, Bahattin Yildiz, Marcos A. Simplício Jr., Thales B. Paiva, Henrique S. Ogawa, Gabrielle De Micheli, and Eduardo Lopes Cominetti. Incompleteness in number-theoretic transforms: New tradeoffs and faster lattice-based cryptographic applications. In *2024 IEEE Euro-*

pean Symposium on Security and Privacy, pages 565–584, Venice, Italy, June 30 – July 4, 2025. IEEE Computer Society Press.

JPK+24. Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Minsik Kang, Donghwan Kim, Jung Hee Cheon, and Jung Ho Ahn. NeuJeans: Private neural network inference with joint optimization of convolution and FHE bootstrapping. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 4361–4375, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

K+96. Ron Kohavi et al. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Kdd*, volume 96, pages 202–207, 1996.

Kim25. Jaehyung Kim. Efficient homomorphic integer computer from CKKS. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(4):873–898, 2025.

KS25. Kamil Kluczniak and Giacomo Santato. On circuit private, multikey and threshold approximate homomorphic encryption. *IACR Communications in Cryptology (CiC)*, 2(1):9, 2025.

KVH24. Christian Knabenhans, Alexander Viand, and Anwar Hithnawi. Towards robust fhe for the real world. Real World Crypto 2024, 2024.

LFKN90. Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st Annual Symposium on Foundations of Computer Science*, pages 2–10, St. Louis, MO, USA, October 22–24, 1990. IEEE Computer Society Press.

LM21. Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 648–677, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.

LS19. Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):180–201, 2019.

LXZ21. Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2968–2985, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

NOZ26. Ngoc Khanh Nguyen, George O'Rourke, and Jiapeng Zhang. Hachi: Efficient lattice-based multilinear polynomial commitments over extension fields. Cryptology ePrint Archive, Paper 2026/156, 2026.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

SL20. Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020.

STW24. Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with Lasso. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 180–209, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

Tha13. Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

VKH23. Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption, 2023.

VSBW13. Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 223–237, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

WMSS93. William Wolberg, Olvi Mangasarian, Nick Street, and W. Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5DW2B.

XZZ+19. Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

ZCF24. Hadas Zeilberger, Binyi Chen, and Ben Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 138–169, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

ZLW+21. Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 159–177, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

A Additional Preliminaries

PIOP Efficiency. We consider the following efficiency metrics: prover's (resp. verifier's) running time $\mathsf{T}_{\mathcal{P}}$ (resp. $\mathsf{T}_{\mathcal{V}}$); number of rounds r; number of oracles sent by the prover o; size s of the proof oracles (i.e., the sum of length of all the polynomials sent by the prover); number of verifier's queries q to the oracles.

A.1 Polynomial Commitments with Batched Evaluations

Definition 5 (Multilinear Polynomial Commitment). *A polynomial commitment scheme for multilinear polynomials over a ring R is a tuple* $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, , \mathsf{ProveEval}, \mathsf{VerEval})$

$\mathsf{Setup}(1^\lambda, \ell) \to (\mathsf{ck}, \mathsf{vk})$ *takes the security parameter and (possibly) an upper bound on the number of variables ℓ, and outputs a commitment key* ck *and a verification key* vk.

$\mathsf{Com}(\mathsf{ck}, f) \to (\mathsf{cm}_f, \mathsf{opn}_f)$ *takes a multilinear polynomial* $f \in R^{(\leq 1)}[X_1, \ldots, X_\ell]$*, and outputs a commitment* cm_f *and an opening* opn_f*.*

$\mathsf{VerCom}(\mathsf{vk}, \mathsf{cm}_f, f, \mathsf{opn}_f) \to b$ *checks if* opn_f *is a valid opening for the commitment* cm_f *to a polynomial* f*, by accepting* ($b = 1$) *or not* ($b = 0$)*.*

$\langle \mathsf{ProveEval}(\mathsf{ck}, \mathsf{cm}_f, \boldsymbol{a}, y, \ell; f, \mathsf{opn}_f), \mathsf{VerEval}(\mathsf{vk}, \mathsf{cm}_f, \boldsymbol{a}, y, \ell) \rangle \to b$ *is a protocol in which the prover* ProveEval *aims to convince the verifier* VerEval *that* $y = f(\boldsymbol{a})$ *for the polynomial* $f \in R^{(\leq 1)}[X_1, \ldots, X_\ell]$ *committed in* cm_f*. As a shorthand we write* Eval *to refer to* $\langle \mathsf{ProveEval}, \mathsf{VerEval} \rangle$*.*

Correctness. PC *is correct if for any multilinear* ℓ*-variate polynomial* f

$$\Pr\left[\mathsf{VerCom}(\mathsf{vk}, \mathsf{cm}_f, f, \mathsf{opn}_f) = 1 \;\middle|\; \begin{array}{l}(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, \ell) \\ (\mathsf{cm}_f, \mathsf{opn}_f) \leftarrow \mathsf{Com}(\mathsf{ck}, f)\end{array}\right] = 1.$$

Binding. PC *is computationally binding if for all PPT adversaries* Adv *the following probability is* $\mathsf{negl}(\lambda)$*:*

$$\Pr\left[\begin{array}{c}\mathsf{VerCom}(\mathsf{vk}, \mathsf{cm}, f, \mathsf{opn}_f) = 1 \\ \wedge \mathsf{VerCom}(\mathsf{vk}, \mathsf{cm}, f', \mathsf{opn}'_f) = 1 \\ \wedge f \neq f'\end{array} \;\middle|\; \begin{array}{l}(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda, \ell) \\ (\mathsf{cm}, f, \mathsf{opn}_f, f', \mathsf{opn}'_f) \leftarrow \mathsf{Adv}(\mathsf{ck})\end{array}\right]$$

Knowledge Soundness. *We say that* PC *is knowledge-sound if* $\langle \mathsf{ProveEval}, \mathsf{VerEval} \rangle$ *is a complete and knowledge-sound argument of knowledge for the NP relation* $\mathcal{R}_{\mathsf{Eval}} = \{(\mathbb{x} = (\mathsf{cm}_f, \boldsymbol{a}, y, \ell); \mathbb{w} = (f, \mathsf{opn}_f))\}$ *of tuples such that*

$$f \in R^{(\leq 1)}[X_1, \ldots, X_\ell] \wedge y = f(\boldsymbol{a}) \wedge \mathsf{VerCom}(\mathsf{vk}, \mathsf{cm}_f, f, \mathsf{opn}_f) = 1$$

Succinctness. *We say that* PC *is succinct if the commitments produced by* Com *have size sublinear in the size of a polynomial and if* $\langle \mathsf{ProveEval}, \mathsf{VerEval} \rangle$ *is a succinct argument.*

A.2 Compiling PIOPs into SNARKs

Given a multilinear polynomial commitment it is possible to turn a PIOP for a relation $\mathcal{R}$ into a public-coin interactive argument of knowledge for the same relation. The compilation consists in two steps: replace every oracle sent by the prover with a commitment to the underlying polynomial, and replace every verifier's query to an oracle $[\![p]\!]$ at a point $\boldsymbol{a}$ with sending the actual value $p(\boldsymbol{a})$ and proving its correctness w.r.t. the commitment of p by using the $\langle \mathsf{ProveEval}, \mathsf{VerEval} \rangle$ protocol. To this standard compilation technique we add the observation that if we start from a PIOP for an oracle relation $\mathcal{R}$ then we can easily build a (preprocessing) commit-and-prove argument. Finally, if the PIOP is public-coin so is the resulting argument system and thus it can be further compiled into a non-interactive argument using the Fiat-Shamir transform.

We summarize the compiler in the following theorem whose proof follows from previous work, e.g., [BFS20].

Theorem 3. *Let* PC *be a multilinear polynomial commitment scheme and let* Π *be a PIOP for an oracle relation* $\mathcal{R}$ *over tuples* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = (\{[\![p_j^{\mathbb{i}}]\!]\}_j, \{[\![p_i^{\mathbb{x}}]\!]\}_i, (\{p_j^{\mathbb{i}}\}_j, \{p_i^{\mathbb{x}}\}_i))$ *with negligible soundness error. There exists a public-coin interactive argument for the commit-and-prove relation*

$$\begin{aligned}
&\mathcal{R}' = \{(\mathbb{i}', \mathbb{x}', \mathbb{w}') = \\
&(\{\mathsf{cm}_j^{\mathbb{i}}\}_j, \{\mathsf{cm}_i^{\mathbb{x}}\}_i, (\{p_i^{\mathbb{x}}, \mathsf{opn}_i^{\mathbb{x}}\}_i \{[\![p_j^{\mathbb{i}}]\!]\}_j, \{[\![p_i^{\mathbb{x}}]\!]\}_i, (\{p_j^{\mathbb{i}}\}_j, \{p_i^{\mathbb{x}}\}_i))) \in \mathcal{R} \\
&\bigwedge_i \mathsf{VerCom}(\mathsf{vk}, \mathsf{cm}_i^{\mathbb{x}}, p_i^{\mathbb{x}}, \mathsf{opn}_i^{\mathbb{x}}) = 1\}
\end{aligned}$$

where all the commitments $\mathsf{cm}_j^{\mathbb{i}} = \mathsf{Com}(\mathsf{ck}, p_j^{\mathbb{i}})$ *are honestly computed in the preprocessing phase.*

The efficiency of the resulting argument depends on the efficiency of both Π and PC as follows. For Π, let $\mathsf{T}_{\mathcal{P}}$ and $\mathsf{T}_{\mathcal{V}}$ be its prover and verifier running times, and let $\mathsf{r}, \mathsf{q}, \mathsf{o}, \mathsf{s}$ be, respectively, its round complexity, number of verifier's queries, number of prover's oracles, and total size of the oracles. For PC, let $\mathsf{T}_{\mathsf{Com}}, \mathsf{T}_{\mathsf{ProveEval}}, \mathsf{T}_{\mathsf{VerEval}}$ be the running times of the corresponding algorithms, and let $|\mathsf{cm}|$ and $|\pi_{\mathsf{PC}}|$ be the size of a commitment and a proof respectively.

- Proving time is $\mathsf{T}_{\mathcal{P}} + \mathsf{T}_{\mathsf{Com}} \cdot \mathsf{s} + \mathsf{T}_{\mathsf{ProveEval}} \cdot \mathsf{q}$
- Verifier time is $\mathsf{T}_{\mathcal{V}} + \mathsf{T}_{\mathsf{VerEval}} \cdot \mathsf{q}$
- Proof size is $|\mathsf{cm}| \cdot \mathsf{o} + |\pi_{\mathsf{PC}}| \cdot \mathsf{q}$

In our work, we use a straightforward extension of this compilation that relies on PCs with *batched evaluation proofs* in order to prove all the evaluation proofs involve in the PIOP with single proof. Assume that the proving time (resp. proof size) of batching is a function $\mathsf{T}_{\mathsf{ProveEval}}(\cdot)$ (resp. $\mathsf{S}_{\pi}(\cdot)$) of the batch size. Then we get the following efficiency:

- Proving time is $\mathsf{T}_{\mathcal{P}} + \mathsf{T}_{\mathsf{Com}} \cdot \mathsf{s} + \mathsf{T}_{\mathsf{ProveEval}}(\mathsf{q})$
- Verifier time is $\mathsf{T}_{\mathcal{V}} + \mathsf{T}_{\mathsf{VerEval}} \cdot \mathsf{q}$
- Proof size is $|\mathsf{cm}| \cdot \mathsf{o} + \mathsf{S}_{\pi}(\mathsf{q})$

A.3 Round-by-Round Soundness

We recall the notions of round-by-round (RBR) soundness [CCH+19] and round-by-round knowledge [CMS19] for PIOPs. These notions are useful to precisely argue the knowledge-soundness error of the SNARK obtained after applying the Fiat-Shamir transformation.

Informally, a RBR sound protocol has a well-defined state (accept or reject) at each round and, if it starts with a false statement, at each round there should be a negligible probability of turning from a reject to an accept state.

Definition 6 (Round-by-Round Soundness [CCH+19]). *A PIOP* Π *for an indexed oracle relation* $\mathcal{R}$ *satisfies round-by-round (RBR) soundness with error* δ *if there exists a state function* $\mathsf{state}(\mathbb{i}, \mathbb{x}, \mathsf{tr})$ *such that:*

- *If* $(\mathtt{i}, \mathbb{x}) \notin \mathcal{L}_{\mathcal{R}}$ *then* $\mathsf{state}(\mathtt{i}, \mathbb{x}, \emptyset) = 0$,
- *For any* $(\mathtt{i}, \mathbb{x})$ *and any* complete *transcript* $(\mathtt{i}, \mathbb{x}, \mathsf{tr})$, *if* $\mathsf{state}(\mathtt{i}, \mathbb{x}, \mathsf{tr}) = 0$ *then* $\mathcal{V}(\mathtt{i}, \mathbb{x}, \mathsf{tr})$ *rejects.*
- *If* $(\mathtt{i}, \mathbb{x}, \mathsf{tr})$ *is a partial transcript such that* $\mathsf{state}(\mathtt{i}, \mathbb{x}, \mathsf{tr}) = 0$, *then for every potential prover's message* π, *holds that* $\Pr[\mathsf{state}(\mathtt{i}, \mathbb{x}, \mathsf{tr}|\pi|c) = 1 : c \leftarrow\$ \mathcal{V}(\mathtt{i}, \mathbb{x}, \mathsf{tr}|\pi)] \leq \delta$.

Definition 7 (Round-by-Round Knowledge [CMS19]). *A PIOP* Π *for an indexed oracle relation* $\mathcal{R}$ *satisfies round-by-round (RBR) knowledge with error* δ *if there exists a polynomial-time extractor* Ext *and a state function* $\mathsf{state}(\mathtt{i}, \mathbb{x}, \mathsf{tr})$ *such that:*

- $\mathsf{state}(\mathtt{i}, \mathbb{x}, \emptyset) = 0$,
- *For any* $(\mathtt{i}, \mathbb{x})$ *and any* complete *transcript* $(\mathtt{i}, \mathbb{x}, \mathsf{tr})$, *if* $\mathsf{state}(\mathtt{i}, \mathbb{x}, \mathsf{tr}) = 0$ *then* $\mathcal{V}(\mathtt{i}, \mathbb{x}, \mathsf{tr})$ *rejects.*
- *If* $(\mathtt{i}, \mathbb{x}, \mathsf{tr})$ *is a partial transcript such that* $\mathsf{state}(\mathtt{i}, \mathbb{x}, \mathsf{tr}) = 0$, *if* $\Pr[\mathsf{state}(\mathtt{i}, \mathbb{x}, \mathsf{tr}|\pi|c) = 1 : c \leftarrow\$ \mathcal{V}(\mathtt{i}, \mathbb{x}, \mathsf{tr}|\pi)] > \delta$, *then* $(\mathtt{i}, \mathbb{x}, \mathsf{Ext}(\mathtt{i}, \mathbb{x}, \mathsf{tr}|\pi)) \in \mathcal{R}$.

It is possible to see that an analogous version of Lemma 2 can be stated for RBR soundness and RBR knowledge.

Lemma 6. *Consider a* δ*-RBR-sound PIOP for oracle relations* $\mathcal{R}$ *such that for all* $(\mathtt{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ $\mathbb{w}$ *consists only of polynomials such that the statement contains oracles to these polynomials. The PIOP has* δ*-RBR-knowledge and the extractor runs in time* $O(|\mathbb{w}|)$.

The proof, in particular the construction of the extractor, is essentially the same as in the proof of Lemma 2 (see [CBBZ23]). The only difference is that we have to define the state function for round-by-round knowledge, which can be define equal to the state function of RBR-soundness, except that on empty transcripts it always rejects, regardless of whether the statement is in the language.

RBR soundness is particularly useful to understand the (RBR) soundness guarantees of protocols that need parallel repetitions. This is summarized by the result [CCH+19, Corollary 5.7], restated here for PIOPs.

Theorem 4. *Suppose that* Π *is a* $2r$*-round PIOP with* δ*-soundness. Then, its* κ*-parallel repetition* Π^{κ} *is* δ^{κ}*-sound and* $\delta^{\kappa/r}$*-RBR-sound.*

A.4 The Sumcheck Protocol

Below we describe Π_{sum} as either an interactive proof or a PIOP. In our protocols, $\mathcal{V}$ might not be given (oracle access to) $p(r_1, \ldots, r_\ell)$ directly, but rather enough information to compute that on its own. For example, for $p = fg + h$, where $\mathcal{V}$ knows f, it might be given (oracle access to) $g(r_1, \ldots, r_\ell)$ and $h(r_1, \ldots, r_\ell)$.

PIOP Π_{usum} for $((y, [\![\tilde{p}]\!]); \tilde{p}) \in \mathcal{R}_{usum}$

$\mathcal{P}$ and $\mathcal{V}$ run the PIOP Π^0_{sum} for $((y, [\![\tilde{u}]\!]); \tilde{u}) \in \mathcal{R}_{usum}$ with $\tilde{u} = \mathsf{unpack}(\tilde{p})$.
Given $\Pi^0_{sum}.\mathcal{V}$'s query to $[\![\tilde{u}]\!]$ on $\boldsymbol{r} = (\boldsymbol{r}_1, \boldsymbol{r}_2)$, $\Pi_{usum}.\mathcal{V}$ makes a query to $[\![\tilde{p}]\!]$ on $\boldsymbol{r}_1$ and given its answer $v = \tilde{p}(\boldsymbol{r}_1)$, returns $\mathsf{uEval}(v, \boldsymbol{r}_2)$ to $\Pi^0_{sum}.\mathcal{V}$.

Figure 6: PIOP for $\mathcal{R}_{usum}$

The sum-check protocol/PIOP Π_{sum} for $\mathcal{R}_{sum}$

for $j = 1$ to ℓ

$\mathcal{P}$ computes and sends $g_j(X) \leftarrow \sum_{\boldsymbol{i} \in \{0,1\}^{\ell - j}} p(r_1, \ldots, r_{j-1}, X, \boldsymbol{i})$.

$\mathcal{V}$ sends $r_j \leftarrow\!\!\$\ \hat{S}_N$.

$\mathcal{P}$ sends $p(r_1, \ldots, r_\ell)$. (In the PIOP version: $\mathcal{V}$ queries $[\![p]\!]$ on $(r_1, \ldots, r_\ell)$).

$\mathcal{V}$ accepts if the following checks are satisfied:

$\forall j \in [\ell] : g_{j-1}(r_{j-1}) = g_j(0) + g_j(1)$, where $g_0(r_0) = y$

$g_\ell(r_\ell) = p(r_1, \ldots, r_\ell)$

Theorem 5. *Let $\hat{S}_N \subseteq R$ be an exceptional set. The PIOP for $\mathcal{R}_{sum}$ described above has perfect completeness and δ-knowledge-soundness with $\delta = d\ell/|\hat{S}_N|$.*

Proof. It follows from the soundness analysis in [CCKP19] and Lemma 2. □

Unpacked Sumcheck We define the *unpacked sumcheck* oracle relation $\mathcal{R}_{usum}$ as the set of tuples $((y, [\![\tilde{p}]\!]); \tilde{p})$ such that $y = \sum_{\boldsymbol{b} \in \{0,1\}^\ell, \boldsymbol{k} \in \{0,1\}^\nu} \tilde{p}(\boldsymbol{b})[\boldsymbol{k}]$, or equivalently,

$$\mathcal{R}_{usum} = \{((y, [\![\tilde{p}]\!]); \tilde{p}) : ((y, [\![\mathsf{unpack}(\tilde{p})]\!]); \mathsf{unpack}(\tilde{p})) \in \mathcal{R}_{sum}\}.$$

Given the above reduction of $\mathcal{R}_{usum}$ to $\mathcal{R}_{sum}$ we can construct a PIOP Π_{usum} for $\mathcal{R}_{usum}$ using the PIOP Π_{sum} for $\mathcal{R}_{sum}$, changed so that the verifier samples random challenges in an exceptional set $\hat{S}_0 \subset \mathbb{Z}_q$. Let us call Π^0_{sum} this variant of Π_{sum}. Π_{usum} simply runs Π^0_{sum} on the virtual oracle $\mathsf{unpack}(\tilde{p})$ and uses uEval to simulate queries to the unpacked oracle.

Theorem 6. *Π_{usum} is a PIOP with perfect completeness, prover's efficiency $O(2^{\ell+\nu})$, verifier's efficiency $O(\ell+2^\nu)$, δ_{usum}-knowledge soundness with $\delta_{usum} = (\ell + \nu)/|\hat{S}_0|$, and RBR-knowledge with error $1/|\hat{S}_0|$.*

The theorem follows by specializing known results for the sumcheck protocol to the case of challenges in the smaller set $\hat{S}_0$. When $\hat{S}_0$ is not large enough to have $1/|\hat{S}_0|$ negligible, we repeat Π_{usum} $\kappa(\ell + \nu)$ times in parallel, obtaining RBR-knowledge error $1/|\hat{S}_0|^\kappa$ which is negligible for large enough κ [CCH+19].

A.5 CKKS

We recall the proof-friendly CKKS construction, as presented in [CCC+25]. Note that the secret and public key generation algorithms, as well as the encryption, addition and multiplication algorithms are unchanged. For a full description of the "regular" CKKS scheme, we point the reader to [CKKS17]. For ease of exposition, we mostly follow the notation of [CCC+25].

Let $q_0 < q_1 < \cdots < q_L$ be the chain of moduli as outlined in the preliminaries and $(\omega_0, \omega_1, \ldots, \omega_L)$ their respective CRT bases, χ_{key} be the secret key distribution, and $\chi_{\text{err}}, \chi_{\text{enc}}$ be the error and encryption distributions, respectively.

CRT Let l denote $l = L - j$, and $\omega_j = (p_0, p_1, \ldots, p_l)$ be a CRT base for q_j. Given $a \in R_{q_0}$, the inverse CRT map is defined as follows:

$$\mathsf{CRT}^{-1}_{\omega_j}(a) \coloneqq \left([a]_{p_0}, [a]_{p_1}, \ldots, [a]_{p_l}\right) \in R^{l+1}_{q_0}. \tag{16}$$

Let $Q_i = q_0/p_i$ and $\hat{Q}_i = \left[(q_0/p_i)^{-1}\right]_{p_i}$ be the CRT recomposition constants. The generator of the ideal defining the embedding is defined as $z_l \coloneqq \sum_{i=0}^{l} Q_i \hat{Q}_i = \mathsf{CRT}(\underbrace{1, \ldots, 1}_{l+1 \text{ times}}, 0, \ldots, 0) \in R_{q_0}$.

For each $0 \le j \le L$, the CRT recomposition vector for a given $a \in R_{q_0}$ is:

$$\mathsf{PW}_{\omega_l}(a) \coloneqq \left(\left[aQ_0\hat{Q}_0\right]_{q_0}, \left[aQ_1\hat{Q}_1\right]_{q_0}, \ldots, \left[aQ_l\hat{Q}_l\right]_{q_0}\right). \tag{17}$$

Proof-friendly CKKS

$\mathsf{evk} \leftarrow \mathsf{KeySwitchGen}(s, s^2)$**:** For $\mathsf{sk} = (1, s)$, for $i = 0, \ldots, L$, sample $a_i \xleftarrow{\$} R_{q_0}$, sample $e_i \leftarrow \chi_{\text{err}}$. Compute $b_i = -a_i \cdot s + e_i + \mathsf{PW}_{\omega_L}(s^2)[i] \pmod{q_0}$. Recall that $l = L - j$, for each level $j \in \{0, \ldots, L\}$, compute

$$\mathsf{evk} \coloneqq (\mathsf{evk}_{j,0}, \mathsf{evk}_{j,1}) \leftarrow ((z_l b_i)_{i=0,\ldots,l}, (z_l a_i)_{i=0,\ldots,l}) \in \left(R^2_{q_0}\right)^{l+1}.$$

$\mu + e' \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$**:** Given a ciphertext $\mathsf{ct} \in R_{q_0}$ for some level j, parse $(c_0, c_1) = \mathsf{ct}$. Output the CRT recomposition of the first $l = L - j$ components of $\langle \mathsf{ct}, \mathsf{sk} \rangle$, namely $\mathsf{CRT}_{\omega_j}(\langle \mathsf{ct}, \mathsf{sk} \rangle)$.

$(d_0, d_1, d_2) \leftarrow \mathsf{PreMult}(\mathsf{ct}_0, \mathsf{ct}_1, j)$**:** For some level $j \in \{0, \ldots, L\}$, compute

$$(d_0, d_1, d_2) \coloneqq (\mathsf{ct}_0[0] \cdot \mathsf{ct}_1[0], \mathsf{ct}_0[0] \cdot \mathsf{ct}_1[1] + \mathsf{ct}_1[0] \cdot \mathsf{ct}_0[1], \mathsf{ct}_0[1] \cdot \mathsf{ct}_1[1]), \tag{18}$$

as an element of $R^3_{q_0}$. Perform the key switching as follows.

$\mathsf{ct}' \leftarrow \mathsf{KeySwitch}(\mathsf{evk}, \mathsf{ct} = (d_0, d_1, d_2))$**:** output $\mathsf{ct}' = (c'_0, c'_1)$ where

$$c'_i \coloneqq d_i + \langle \mathsf{CRT}^{-1}_{\omega_j}(d_2), \mathsf{evk}_{j,i} \rangle \in R_{q_0}, \quad i = 0, 1. \tag{19}$$

$\mathsf{ct}'' \leftarrow \mathsf{Rescale}(\mathsf{ct}', j)$**:** Let $\mathsf{ct}' = (c'_0, c'_1)$, and $\mathsf{p}_l^{-1} = [q_{j+1}/q_j]_{q_{j+1}} \cdot z_l \in R_{q_0}$. Rescale is done by computing $\mathsf{ct}'' \coloneqq (c''_0, c''_1) \in R^2_{q_0}$ as

$$c''_i \coloneqq \left(c'_i - [c'_i]_{p_l}\right) \mathsf{p}_l^{-1} \in R_{q_0}, \quad i = 0, 1. \tag{20}$$

A.6 A modular construction of the PIOP Π_{CKKS}

Belowe recall how we modularly build a PIOP Π_{CKKS} for the relation $\mathcal{R}_{CKKS}$ from PIOPs Π_{range} for $\mathcal{R}_{range}$ and Π_{AC} for $\mathcal{R}_{AC}$.

PIOP Π_{CKKS} for $(\mathtt{i}, \mathtt{x}; \mathtt{w}) \in \mathcal{R}_{CKKS}$

$\mathcal{P}$ and $\mathcal{V}$ run Π_{AC} for $(\mathbf{C}, ([\![\tilde{\mathbf{x}}]\!], [\![\tilde{\mathbf{y}}]\!], [\![\tilde{\mathbf{w}}_{2^u}]\!], \{[\![\tilde{\mathbf{w}}_l]\!]\}_{l=0}^{D-2}); \mathbf{x}, \mathbf{y}, \mathbf{w}_{2^u}, \{\mathbf{w}_l\}_{l=1}^{D-2}) \in \mathcal{R}_{AC}$
$\mathcal{P}$ and $\mathcal{V}$ run Π_{range} for $(2^u, [\![\tilde{\mathbf{w}}_{2^u}]\!]; \mathbf{w}_{2^u}) \in \mathcal{R}_{range}$ and $(q_{L-D+1}/p_{L-D+1}, [\![\tilde{\mathbf{y}}]\!]; \mathbf{y}) \in \mathcal{R}_{range}$
For $l = 0, \ldots, D-2$: $\mathcal{P}$ and $\mathcal{V}$ run Π_{range} for $(q_{L-l}/p_{L-l}, [\![\tilde{\mathbf{w}}_l]\!]; \mathbf{w}_l) \in \mathcal{R}_{range}$.

B Auxiliary Material on PIOP for Unpacked Polynomials

B.1 Proof of Theorem 1

Proof. For correctness, we refer to the explanation in Section 3. Prover efficiency follows from the runtime of sumcheck, while Verifier efficiency is the sumcheck's verification plus the time to compute ψ, ψ^* and uEval.

For soundness, if the statement is false, i.e., $\exists i : y_i \neq \tilde{u}(\boldsymbol{x}_i)$, then $\tilde{y}(\boldsymbol{Z}, T) - \sum_{\boldsymbol{b} \in \{0,1\}^{\ell'}} \tilde{u}(\boldsymbol{b}) E(\boldsymbol{b}, \boldsymbol{Z}, T)$ is a nonzero polynomial, and by generalized Schwartz-Zippel the probability that $\tilde{y}(\boldsymbol{\sigma}, \tau) = \sum_{\boldsymbol{b} \in \{0,1\}^{\ell'}} \tilde{u}(\boldsymbol{b}) E(\boldsymbol{b}, \boldsymbol{\sigma}, \tau)$, for random $\boldsymbol{\sigma} \leftarrow\!\!\$\ \hat{S}_0^\nu, \tau \leftarrow\!\!\$\ \hat{S}_0$, is $\leq (\nu + a - 1)/|\hat{S}_0|$. On the other hand, by the soundness of Π^0_{sum}, the probability that the verifier accepts a tuple $((\tilde{y}(\boldsymbol{\sigma}), [\![f]\!]); f) \notin \mathcal{R}_{sum}$, with $f = \tilde{u}(\boldsymbol{X}) E(\boldsymbol{X}, \boldsymbol{\sigma}, \tau)$ of individual degree 2, is $\leq 2(\ell + \nu)/|\hat{S}_0|$. □

C Auxiliary Material on PIOP for Range Checks

C.1 Proof of Lemma 4

Proof. Let us index T_B by tuples $(\mathsf{idy}_1, \ldots, \mathsf{idy}_N) \in [0, B-1]^N$ such that

$$T_B[\mathsf{idy}_1, \ldots, \mathsf{idy}_N] = \sum_{k=1}^{N} \mathsf{idy}_k \cdot Y^{k-1} \in R_q.$$

Let $\mathsf{dig}_j^{(\beta)}(\mathsf{idy})$ be the function that on input an integer idy, returns the j-th digit of its base-β decomposition, i.e., idy_j such that $\mathsf{idy} = \sum_j \mathsf{idy}_j \beta^{j-1}$. Then we can express each element of T_B in terms of elements of $\boldsymbol{t}_\beta$ as follows.

$$\begin{aligned} T_B[\mathsf{idy}_1, \ldots, \mathsf{idy}_N] &= \sum_{k=1}^{N} \mathsf{idy}_k \cdot Y^{k-1} = \sum_{k=1}^{N} \sum_{j=1}^{c} \mathsf{dig}_j^{(\beta)}(\mathsf{idy}_k) \cdot \beta^{j-1} \cdot Y^{k-1} \\ &= \sum_{k=1}^{N} \sum_{j=1}^{c} \boldsymbol{t}_\beta[\mathsf{dig}_j^{(\beta)}(\mathsf{idy}_k)] \cdot Y^{k-1} \cdot \beta^{j-1}. \end{aligned}$$

The above equation implies that $\tilde{v}(\boldsymbol{i}) \in T_B$ holds $\forall \boldsymbol{i} \in \{0,1\}^\ell$ if and only if $\exists \tilde{\mu} \in \mathbb{Z}_q^{(\leq 1)}[X_1, \ldots, X_\ell, Z_1, \ldots, Z_\nu, Z'_1, \ldots, Z'_\gamma]$ such that, for $\tilde{u} = \mathsf{unpack}(\tilde{v})$,

$$\forall \boldsymbol{i} \in \{0,1\}^\ell, \boldsymbol{k} \in \{0,1\}^\nu : \tilde{u}(\boldsymbol{i}, \boldsymbol{k}) = \sum_{\boldsymbol{j} \in \{0,1\}^\gamma} \tilde{\mu}(\boldsymbol{i}, \boldsymbol{k}, \boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}, \tag{21}$$

$$\forall \boldsymbol{i} \in \{0,1\}^\ell, \boldsymbol{j} \in \{0,1\}^\gamma, \boldsymbol{k} \in \{0,1\}^\nu : \tilde{u}(\boldsymbol{i}, \boldsymbol{k}, \boldsymbol{j}) \in \boldsymbol{t}_\beta. \tag{22}$$

Concretely, $\tilde{\mu}$ is determined by the values $\tilde{\mu}(\boldsymbol{i}, \boldsymbol{k}, \boldsymbol{j}) = \mathsf{dig}_j^{(\beta)}(\tilde{v}(\boldsymbol{i})[\boldsymbol{k}])$, $\forall \boldsymbol{i} \in \{0,1\}^\ell$, $\boldsymbol{j} \in \{0,1\}^\gamma$, $\boldsymbol{k} \in \{0,1\}^\nu$. This completes the proof by definition of unpack and the equations (4), (5). □

C.2 Proof of Theorem 2

Proof. To prove RBR-soundness, we start by defining a $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_i)$ function on every (partial) transcript tr_i, from round $i = 0$ to $i = 2 + \mathsf{r}_{range}$, where r_{range} denotes the number of rounds of the $\hat{\Pi}_{range}$ protocol:

- For $\mathsf{tr}_0 = \emptyset$, $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0$ iff $(T_B, [\![\tilde{v}]\!]) \notin \mathcal{L}_{\mathcal{R}_{range}}$.
- For $\mathsf{tr}_1 = [\![\tilde{\mu}]\!] | (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\kappa)$, $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 0$ iff $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \notin \mathcal{R}_{range}$ or $\exists i \in [\kappa]$: $f(\boldsymbol{r}_i) \neq 0$, where

$$f(\boldsymbol{X}, \boldsymbol{Z}) = \tilde{u}(\boldsymbol{X}, \boldsymbol{Z}) - \sum_{\boldsymbol{j} \in \{0,1\}^\gamma} \tilde{\mu}(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}.$$

- For $\mathsf{tr}_2 = \mathsf{tr}_1 | (\tilde{\mu}_1^*, \ldots, \tilde{\mu}_\kappa^*) | \boldsymbol{s}$: $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 0$ iff $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \notin \mathcal{R}_{range}$ or $\exists i \in [\kappa]$ such that $\tilde{u}(\boldsymbol{r}_i) \neq \sum_{\boldsymbol{j} \in \{0,1\}^\gamma} \tilde{\mu}_i^*(\boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}$ or $\mu_i^*(\boldsymbol{s}) \neq \tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{s})$.
- For $\mathsf{tr}_{2+k} = \mathsf{tr}_2 | \mathsf{tr}'_k$ where, for $1 \leq k \leq \mathsf{r}_{range}$, tr'_k is the partial transcript of $\hat{\Pi}_{range}$ at the k-th round: $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_{2+k}) = 0$ iff $\mathsf{state}'(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}'_k) = 0$ or $\exists i \in [\kappa]$ such that $\tilde{u}(\boldsymbol{r}_i) \neq \sum_{\boldsymbol{j} \in \{0,1\}^\gamma} \tilde{\mu}_i^*(\boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}$ or $\mu_i^*(\boldsymbol{s}) \neq \tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{s})$.

Let us now show that this state function satisfies the three properties of RBR-soundness.

First, it is easy to see that by construction, if $(T_B, [\![\tilde{v}]\!]; \tilde{v}) \notin \mathcal{R}_{range}$ then $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0$.

Second, for any *complete* transcript $\mathsf{tr} = \mathsf{tr}_{2+\mathsf{r}_{range}} = \mathsf{tr}_2 | \mathsf{tr}'_{\mathsf{r}_{range}}$, by construction we have that if $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}) = 0$, then either $\mathsf{state}'(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}'_k) = 0$, which means that the verifier of $\hat{\Pi}_{range}$ rejects (by the RBR-soundness of $\hat{\Pi}_{range}$), or at least one of the verifier's checks in step 3 fails. Therefore, $\Pi_{range}.\mathcal{V}$ rejects.

Third, we show that if state rejects on a partial transcript of round i, for any prover's message of the round $i + 1$, we have a concrete (negligible) bound on the probability that state accepts on the next partial transcript. We show this property round by round.

Claim. If $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0$, then for any prover's message $[\![\tilde{\mu}]\!]$ we have that

$$\Pr_{\boldsymbol{r}_1,\ldots,\boldsymbol{r}_\kappa \leftarrow\$ \hat{S}_0^{\ell+\gamma}}[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1 = ([\![\tilde{\mu}]\!](\boldsymbol{r}_1, \ldots, \boldsymbol{r}_\kappa))) = 1] \leq \left(\frac{\ell+\nu}{|\hat{S}_0|}\right)^\kappa.$$

By definition, $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0$ iff $(T_B, [\![\tilde{v}]\!]) \notin \mathcal{L}_{\mathcal{R}_{range}}$ and by Lemma 4 this means that for every $\tilde{\mu}$ we have that the polynomial $f(\boldsymbol{X}, \boldsymbol{Z}) = \tilde{u}(\boldsymbol{X}, \boldsymbol{Z}) - \sum_{\boldsymbol{j}\in\{0,1\}^\gamma} \tilde{\mu}(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})} \neq 0$ or $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]) \notin \mathcal{L}_{\hat{\mathcal{R}}_{range}}$. Instead, state accepting on tr_1 means that $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \mathcal{R}_{range}$ and $\forall i \in [\kappa]$: $f(\boldsymbol{r}_i) = 0$.

$$\begin{aligned}
&\Pr[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 1 \mid \mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0] \\
&= \Pr[(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \mathcal{R}_{range} \bigwedge_{i\in[\kappa]} f(\boldsymbol{r}_i) = 0 \mid f(\boldsymbol{X}, \boldsymbol{Z}) \neq 0 \vee (\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \notin \mathcal{R}_{range}] \\
&\leq \Pr[(\bigwedge_{i\in[\kappa]} f(\boldsymbol{r}_i) = 0 \mid f(\boldsymbol{X}, \boldsymbol{Z}) \neq 0] \\
&= \prod_{i\in[\kappa]} \Pr_{\boldsymbol{r}_i \leftarrow\$ \hat{S}_0^{\ell+\gamma}}[(f(\boldsymbol{r}_i) = 0 \mid f(\boldsymbol{X}, \boldsymbol{Z}) \neq 0 \wedge (\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \mathcal{R}_{range}].
\end{aligned}$$

By generalized Schwartz-Zippel, $\Pr_{\boldsymbol{r}_i \leftarrow\$ \hat{S}_0}[f(\boldsymbol{r}_i) = 0 \mid f(\boldsymbol{X}, \boldsymbol{Z}) \neq 0] \leq \frac{\ell+\gamma}{|\hat{S}_0|}$, and the claimed bound follows by considering the probability that this occurs for κ independent tests.

Claim. If $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 0$, then for any prover's message $(\tilde{\mu}_1^*, \ldots, \tilde{\mu}_\kappa^*)$ we have that

$$\Pr_{\boldsymbol{s} \leftarrow\$ \hat{S}_d^\gamma}[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2 = (\mathsf{tr}_1|(\tilde{\mu}_1^*, \ldots, \tilde{\mu}_\kappa^*)|\boldsymbol{s})) = 1] \leq \frac{\gamma}{|\hat{S}_d|}.$$

By definition, $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 0$ implies that either $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \notin \mathcal{R}_{range}$ or $\exists i \in [\kappa]$: $f(\boldsymbol{r}_i) \neq 0$. On the other hand, if $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 1$ then $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \mathcal{R}_{range}$ and $\forall i \in [\kappa]$ $\tilde{u}(\boldsymbol{r}_i) = \sum_{\boldsymbol{j}\in\{0,1\}^\gamma} \tilde{\mu}_i^*(\boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}$ and $\mu_i^*(\boldsymbol{s}) = \tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{s})$.

Consider the specific index i for which $f(\boldsymbol{r}_i) \neq 0$, then we have that

$$\begin{aligned}
&\Pr[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 1 \mid \mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 0] \\
&\leq \Pr[\bigwedge_{i=1}^{\kappa} \tilde{u}(\boldsymbol{r}_i) = \sum_{\boldsymbol{j}\in\{0,1\}^\gamma} \tilde{\mu}_i^*(\boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})} \wedge \mu_i^*(\boldsymbol{s}) = \tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{s}) \mid \exists i \in [\kappa]\ f(\boldsymbol{r}_i) \neq 0] \\
&\leq \Pr[\tilde{u}(\boldsymbol{r}_i) = \sum_{\boldsymbol{j}\in\{0,1\}^\gamma} \tilde{\mu}_i^*(\boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})} \wedge \mu_i^*(\boldsymbol{s}) = \tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{s}) \mid f(\boldsymbol{r}_i) \neq 0] \\
&\leq \Pr[\mu_i^*(\boldsymbol{s}) = \tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{s}) \mid \tilde{u}(\boldsymbol{r}_i) = \sum_{\boldsymbol{j}\in\{0,1\}^\gamma} \tilde{\mu}_i^*(\boldsymbol{j}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})} \wedge f(\boldsymbol{r}_i) \neq 0] \\
&= \Pr[\mu_i^*(\boldsymbol{s}) = \tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{s}) \mid \tilde{\mu}(\boldsymbol{r}_i, \boldsymbol{Z}') \neq \tilde{\mu}_i^*(\boldsymbol{Z}')].
\end{aligned}$$

To see the last equality, observe that by definition of f we get that $\sum_{\boldsymbol{j}\in\{0,1\}^\gamma} \tilde{\mu}(\boldsymbol{r}_i,\boldsymbol{j})\cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{j})} \neq \sum_{\boldsymbol{j}\in\{0,1\}^\gamma} \tilde{\mu}_i^*(\boldsymbol{j})\cdot\beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}$ and since $\beta\neq 0$ and $\beta^{2^\gamma-1}<q$ this means that there is a $\boldsymbol{j}$ such that $\mu_i^*(\boldsymbol{j})\neq\tilde{\mu}(\boldsymbol{r}_i,\boldsymbol{j})$, and thus $g(\boldsymbol{Z}')=\tilde{\mu}(\boldsymbol{r}_i,\boldsymbol{Z}')-\sum_{\boldsymbol{j}\in\{0,1\}^\gamma}\tilde{\mu}_i^*(\boldsymbol{j})\cdot\widetilde{\mathsf{eq}}(\boldsymbol{Z}',\boldsymbol{j})\neq 0$. However, by generalized Schwartz-Zippel, $\Pr_{\boldsymbol{s}\leftarrow\$\hat{S}_d}[g(\boldsymbol{s})=0\mid g(\boldsymbol{Z}')\neq 0]\leq\frac{\gamma}{|\hat{S}_d|}$, which proves the claim.

Claim. For every $k=0$ to $\mathsf{r}_{range}-1$, if $\mathsf{state}(T_B,[\![\tilde{v}]\!],\mathsf{tr}_{2+k})=0$, then for any prover's message π and every verifier's random challenge ρ we have that

$$\Pr_\rho[\mathsf{state}(T_B,[\![\tilde{v}]\!],\mathsf{tr}_{2+k+1}=(\mathsf{tr}_2|\mathsf{tr}'_k|\pi|\rho)=1]\leq\hat{\delta}_{range}.$$

For $k=0$, if $\mathsf{state}(T_B,[\![\tilde{v}]\!],\mathsf{tr}_2)=0$ then $(\boldsymbol{t}_\beta,[\![\tilde{\mu}]\!];\tilde{\mu})\notin\mathcal{R}_{range}$ and, by the RBR-soundness of $\hat{\Pi}_{range}$, $\mathsf{state}'(\boldsymbol{t}_\beta,[\![\tilde{\mu}]\!],\emptyset)=0$. Let E be the event that $\exists i\in[\kappa]$ such that $\tilde{u}(\boldsymbol{r}_i)\neq\sum_{\boldsymbol{j}\in\{0,1\}^\gamma}\tilde{\mu}_i^*(\boldsymbol{j})\cdot\beta^{\mathsf{to\text{-}int}(\boldsymbol{j})}$ or $\mu_i^*(\boldsymbol{s})\neq\tilde{\mu}(\boldsymbol{r}_i,\boldsymbol{s})$. Then we have

$$\begin{aligned}
&\Pr_\rho[\mathsf{state}(T_B,[\![\tilde{v}]\!],\mathsf{tr}_{2+k+1})=1\mid\mathsf{state}(T_B,[\![\tilde{v}]\!],\mathsf{tr}_{2+k})=0]\\
&=\Pr_\rho[\mathsf{state}'(\boldsymbol{t}_\beta,[\![\tilde{\mu}]\!],\mathsf{tr}'_k|\pi|\rho)=1\wedge\bar{E}\mid\mathsf{state}'(\boldsymbol{t}_\beta,[\![\tilde{\mu}]\!],\mathsf{tr}'_k)=0\vee E]\\
&\leq\Pr_\rho[\mathsf{state}'(\boldsymbol{t}_\beta,[\![\tilde{\mu}]\!],\mathsf{tr}'_k|\pi|\rho)=1\mid\mathsf{state}'(\boldsymbol{t}_\beta,[\![\tilde{\mu}]\!],\mathsf{tr}'_k)=0]\leq\hat{\delta}_{range}.
\end{aligned}$$

where the last bound follows by the RBR-soundness of $\hat{\Pi}_{range}$. □

D Our Instantiation of the PIOP $\hat{\Pi}_{range}$

In this section, we present an instantiation of the PIOP $\hat{\Pi}_{range}$ for the relation

$$\hat{\mathcal{R}}_{range}=\{(\mathbb{i},\mathbb{x};\mathbb{w})=(\boldsymbol{t}_\beta,[\![\tilde{\mu}]\!];\tilde{\mu}):\forall\boldsymbol{b}\in\{0,1\}^{\ell^*}\ \tilde{\mu}(\boldsymbol{b})\in\boldsymbol{t}_\beta\},$$

where the index $\boldsymbol{t}_\beta=\{0,\ldots,\beta-1\}\subset\mathbb{Z}$ and $\tilde{\mu}\in H_q^{(\leq 1)}[X_1,\ldots,X_{\ell^*}]$.

A building block of our $\hat{\Pi}_{range}$ is a PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$ for the oracle relation $\hat{\mathcal{R}}_{range}$ over a finite field $\mathbb{F}=\mathbb{Z}_p$ for an odd prime number p that is sufficiently larger than β, following a bound that we specify below.

Preliminaries. Let us denote $\boldsymbol{X}=(X_1,\ldots,X_{\ell^*})$, let $q=p_0\cdot p_1\cdots p_L$ be the product of $L+1$ distinct prime numbers and let p_{min} denote the smallest of them.

Let p be an odd prime number and $I_p=\{-\lfloor p/2\rfloor,\ldots,\lfloor p/2\rfloor\}\subset\mathbb{Z}$. For a positive integer a, let $\pi_a:\mathbb{Z}\to\mathbb{Z}_a$ be the canonical quotient map $\pi_a(x)=x\pmod a$. Let π_{I_p} be the restriction of π_p to I_p, which is a bijection, and let $\iota=\pi_{I_p}^{-1}:\mathbb{Z}_p\to I_p$ be its inverse.

Abusing notation, we extend the definition of these maps to vectors by applying them coordinate-wise and to polynomials by applying them on all their coefficients. For example, given a polynomial $f\in\mathbb{Z}[X_1,\ldots,X_n]$ we denote by $\pi_p(f)\in\mathbb{Z}_p[X_1,\ldots,X_n]$ the polynomial obtained by applying π_p on all the coefficients of f.

We rely on the following proposition, which intuitively formalizes the fact that $\mathbb{Z}_p$ can emulate computations over signed integers, as long as no overflow occurs.

Proposition 1. *Let $f \in \mathbb{Z}[X_1, \ldots, X_n]$ and $\boldsymbol{x} \in \mathbb{Z}^n$ such that $|f(\boldsymbol{x})| < p/2$. Then*

$$\iota(\pi_p(f)(\pi_p(\boldsymbol{x}))) = f(\boldsymbol{x}).$$

Proof. Since π_p is a ring homomorphism we have $\pi_p(f(\boldsymbol{x})) = \pi_p(f)(\pi_p(\boldsymbol{x}))$. Applying ι on both sides $\iota(\pi_p(f)(\pi_p(\boldsymbol{x}))) = \iota(\pi_p(f(\boldsymbol{x}))) = f(\boldsymbol{x})$ since $f(\boldsymbol{x}) \in I_p$ and the restriction of $\iota \circ \pi_p$ to I_p acts as the identity function.

Parameters. Let $\Sigma = \{0, \ldots, \sigma - 1\} \subseteq \mathbb{Z}$ with $\sigma \leq p_{\mathsf{min}}$, and notice that $\pi_q(\Sigma)$ is a subset of the exceptional set $\hat{S}_0 \subset \mathbb{Z}_q$. Let $0 \leq m \leq \ell^*$ be an integer that represents an efficiency tradeoff parameter, and $\kappa \geq 1$ be an integer that represents the number of repetitions.

The set of parameters must satisfy the following conditions:

$$p > 2\beta \cdot (2\sigma)^m \tag{23}$$

$$\left(\frac{m}{\sigma}\right)^{\kappa} \approx 2^{-\lambda} \tag{24}$$

$$m \cdot 2^{\ell^* - m} = \mathsf{poly}(\ell^*) \tag{25}$$

Intuitively, the first condition ensures correctness of the protocol, the second one ensures soundness, and the third one succinctness.

The $\hat{\Pi}_{range}$ protocol. We describe the full protocol in Figure 7.

Theorem 7. *Consider the parameters $q, p, \beta, \sigma, m, \kappa, \ell^*$ satisfying equations (23), (24),(25). Let $\hat{S}_d \subset H_q$ be an exceptional set. Let $\boldsymbol{t}_\beta = \{0, \ldots, \beta - 1\} \subset \mathbb{Z}$ and let $\hat{\Pi}^{\mathbb{F}}_{range}$ be a PIOP for relation $\hat{\mathcal{R}}_{range}$ over the finite field $\mathbb{F} = \mathbb{Z}_p$ that is perfectly complete and has RBR-knowledge error δ'_{range}. Then the $\hat{\Pi}_{range}$ protocol of Figure 7 is a PIOP for $\hat{\mathcal{R}}_{range}$ with perfect completeness and RBR-knowledge error* $\max\left\{\left(\frac{m}{\sigma}\right)^{\kappa}; \frac{\ell^* - m}{|\hat{S}_d|} + \frac{\ell^* - m}{p}; \delta'_{range}\right\}$.

We prove correctness in Section D.1 and RBR-knowledge in Section D.2. When instantiated with the PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$ from Section D.4, p should additionally be $> 2^{\ell^*}$.

D.1 Correctness of $\hat{\Pi}_{range}$

Let $\tilde{\mu}$ be a polynomial such that $\forall \boldsymbol{b} \in \{0,1\}^{\ell^*}$ it holds $\tilde{\mu}(\boldsymbol{b}) \in \boldsymbol{t}_\beta$. By construction, $\tilde{\mu}''$ is such that $\pi_q(\tilde{\mu}''(\boldsymbol{b})) = \tilde{\mu}(\boldsymbol{b})$, $\tilde{\mu}(\boldsymbol{b}) \in \boldsymbol{t}_\beta$, and for $\tilde{\mu}'$ it holds $\tilde{\mu}'(\boldsymbol{b}) \in \boldsymbol{t}'_\beta$ $\forall \boldsymbol{b} \in \{0,1\}^{\ell^*}$ since

$$\tilde{\mu}'(\boldsymbol{b}) = \pi_p(\tilde{\mu}'')(\boldsymbol{b}) = \pi_p(\tilde{\mu}''(\boldsymbol{b})) \in \pi_p(\boldsymbol{t}_\beta) = \boldsymbol{t}'_\beta.$$

PIOP $\hat{\Pi}_{range}$ for $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \hat{\mathcal{R}}_{range}$

1. $\mathcal{P}$ computes $\tilde{\mu}' = \pi_p(\tilde{\mu}'') \in \mathbb{Z}_p[\boldsymbol{X}]$ where $\tilde{\mu}'' \in \mathbb{Z}^{(\leq 1)}[\boldsymbol{X}]$ is such that, for all $\boldsymbol{b}$, $\tilde{\mu}''(\boldsymbol{b}) \in \boldsymbol{t}_\beta$ and $\pi_q(\tilde{\mu}''(\boldsymbol{b})) = \tilde{\mu}(\boldsymbol{b})$, and sends $[\![\tilde{\mu}']\!]$.
   $\mathcal{V}$ samples $\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{\kappa-1} \leftarrow\!\!\$\ \Sigma^m$ and sends them to $\mathcal{P}$.
2. $\mathcal{P}$ sends $(\tilde{y}_0, \tilde{y}'_0, \ldots, \tilde{y}_{\kappa-1}, \tilde{y}'_{\kappa-1})$, where, for $\eta \in [\kappa]$, $\hat{\boldsymbol{r}}_\eta = \pi_q(\boldsymbol{r}_\eta)$, $\boldsymbol{r}'_\eta = \pi_p(\boldsymbol{r}_\eta)$ and:

$$\tilde{y}_\eta(X_{m+1}, \ldots, X_{\ell^*}) = \tilde{\mu}(\hat{\boldsymbol{r}}_\eta, X_{m+1}, \ldots, X_{\ell^*}) \in H_q[X_{m+1}, \ldots, X_{\ell^*}]$$

$$\tilde{y}'_\eta(X_{m+1}, \ldots, X_{\ell^*}) = \tilde{\mu}'(\boldsymbol{r}'_\eta, X_{m+1}, \ldots, X_{\ell^*}) \in \mathbb{Z}_p[X_{m+1}, \ldots, X_{\ell^*}]$$

   $\mathcal{V}$ samples random points $\hat{\boldsymbol{s}} \leftarrow\!\!\$\ \hat{S}_d^{\ell^*-m}$ and $\boldsymbol{s}' \leftarrow\!\!\$\ \mathbb{Z}_p^{\ell^*-m}$.
   For all $\eta \in [\kappa]$, $\mathcal{V}$ queries $[\![\tilde{\mu}]\!]$ and $[\![\tilde{\mu}']\!]$ on $(\hat{\boldsymbol{r}}_\eta, \hat{\boldsymbol{s}}) \in H_q^{\ell^*}$ and $(\boldsymbol{r}'_\eta, \boldsymbol{s}') \in \mathbb{Z}_p^{\ell^*}$ respectively, and checks if

$$\tilde{y}_\eta(\hat{\boldsymbol{s}}) = \tilde{\mu}(\hat{\boldsymbol{r}}_\eta, \hat{\boldsymbol{s}}) \in H_q \quad \text{and} \quad \tilde{y}'_\eta(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{s}') \in \mathbb{Z}_p$$

   Then $\mathcal{V}$ checks if for all $\eta \in [\kappa]$, $\boldsymbol{\delta} \in \{0,1\}^{\ell^*-m}$, and $l \in [0, L]$:

$$\tilde{y}_\eta(\boldsymbol{\delta}) \bmod p_l = \iota(\tilde{y}'_\eta(\boldsymbol{\delta})) \bmod p_l$$

3. $\mathcal{P}$ and $\mathcal{V}$ run a PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$ for $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range}$, where $\boldsymbol{t}'_\beta = \pi_p(\boldsymbol{t}_\beta)$.

Figure 7: Construction of the PIOP $\hat{\Pi}_{range}$ for $\hat{\mathcal{R}}_{range}$ over H_q from $\hat{\Pi}^{\mathbb{F}}_{range}$ for $\hat{\mathcal{R}}_{range}$ over $\mathbb{Z}_p$

Therefore, by its correctness, the protocol $\hat{\Pi}^{\mathbb{F}}_{range}$ accepts.

Next, we have to show that the verifier's checks of step 2 are satisfied.

It is straightforward to see that the checks $\tilde{y}_\eta(\hat{\boldsymbol{s}}) = \tilde{\mu}(\hat{\boldsymbol{r}}_\eta, \hat{\boldsymbol{s}})$ and $\tilde{y}'_\eta(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{s}')$ are satisfied by construction.

Let us now focus on the last check $\tilde{y}_\eta(\boldsymbol{\delta}) \bmod p_l = \iota(\tilde{y}'_\eta(\boldsymbol{\delta})) \bmod p_l$.

By construction of $\tilde{y}_\eta$, $\tilde{y}_\eta(\boldsymbol{\delta}) = \tilde{\mu}(\hat{\boldsymbol{r}}_\eta, \boldsymbol{\delta})$, and since $\hat{\boldsymbol{r}}_\eta = \pi_q(\boldsymbol{r}_\eta)$ and $\pi_q(\tilde{\mu}'') = \tilde{\mu}$, $\tilde{\mu}(\hat{\boldsymbol{r}}_\eta, \boldsymbol{\delta}) = \pi_q(\tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta}))$. Therefore, $\tilde{y}_\eta(\boldsymbol{\delta}) \bmod p_l = \pi_q(\tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta})) \bmod p_l = \tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta}) \bmod p_l$.

To conclude the correctness proof, we need to argue that, for every l, $\tilde{y}_\eta(\boldsymbol{\delta}) \bmod p_l = \tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta}) \bmod p_l = \iota(\tilde{y}'_\eta(\boldsymbol{\delta})) \bmod p_l$. The last equality follows by construction of $\tilde{y}'_\eta$ – by which $\tilde{y}'_\eta(\boldsymbol{\delta}) = \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{\delta})$ – and by applying Proposition 1 along with the bound on the value of $\tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta})$, which we claim below.

Claim. If $\tilde{\mu}''(\boldsymbol{b}) \in \boldsymbol{t}_\beta$ for all $\boldsymbol{b} \in \{0,1\}^{\ell^*}$ then $|\tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta})| < 2^m \beta \sigma^m < p/2$.

Proof. By the multi-linearity of the polynomial we have that for all $\boldsymbol{\delta} \in \{0,1\}^{\ell^*-m}$:

$$\tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta}) = \sum_{\boldsymbol{\phi} \in \{0,1\}^m} \tilde{\mu}''(\boldsymbol{\phi}, \boldsymbol{\delta}) \tilde{\rho}_\eta(\boldsymbol{\phi})$$

where $\tilde{\rho}_\eta(\boldsymbol{\phi}) = \widetilde{\mathsf{eq}}(\boldsymbol{\phi}; \boldsymbol{r}_\eta) = \prod_{i=1}^m (\phi_i r_{\eta,i} + (1-\phi_i)(1-r_{\eta,i}))$. Given that $0 \leq r_{\eta,i} < \sigma$, we have that $-\sigma^m < \tilde{\rho}_\eta(\boldsymbol{\phi}) < \sigma^m$; thus by combining it with the assumption that $\tilde{\mu}''(\boldsymbol{\phi}, \boldsymbol{\delta}) < \beta$ we obtain $|\tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta})| < 2^m \beta \sigma^m < p/2$.

This concludes the correctness proof.

D.2 Security of $\hat{\Pi}_{range}$

We prove the RBR-soundness, from which we get RBR-knowledge by then applying Lemma 6.

To prove RBR-soundness, we start by defining a $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_i)$ function on every (partial) transcript tr_i, from round $i = 0$ to $i = 2 + \mathsf{r}_{range}$, where r_{range} denotes the number of rounds of the $\hat{\Pi}^{\mathbb{F}}_{range}$ protocol:

- For $\mathsf{tr}_0 = \emptyset$, $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0$ iff $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]) \notin \mathcal{L}_{\hat{\mathcal{R}}_{range}}$.
- For $\mathsf{tr}_1 = [\![\tilde{\mu}']\!]|(\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{\kappa-1})$, $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0$ iff $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \notin \hat{\mathcal{R}}_{range}$ or there exist $\eta \in [\kappa]$, $\boldsymbol{\delta} \in \{0,1\}^{\ell^* - m}$ such that $f_{\boldsymbol{\delta}}(\pi_q(\boldsymbol{r}_\eta)) \neq 0 \bmod q$, where $f_{\boldsymbol{\delta}}(\boldsymbol{X}) \in H_q[\boldsymbol{X}]$ is the polynomial

$$f_{\boldsymbol{\delta}}(\boldsymbol{X}) = \tilde{\mu}(\boldsymbol{X}, \boldsymbol{\delta}) - (\tilde{\mu}''(\boldsymbol{X}, \boldsymbol{\delta}) \bmod q)$$

  and $\tilde{\mu}''$ is the multilinear polynomial such that $\tilde{\mu}''(\boldsymbol{b}) = \iota(\tilde{\mu}'(\boldsymbol{b}))$ for all $\boldsymbol{b} \in \{0,1\}^{\ell^*}$.
- For $\mathsf{tr}_2 = \mathsf{tr}_1|(\tilde{y}_0, \tilde{y}'_0, \ldots, \tilde{y}_{\kappa-1}, \tilde{y}'_{\kappa-1})|(\hat{\boldsymbol{s}}, \boldsymbol{s}')$: $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 0$ iff
  - $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \notin \hat{\mathcal{R}}_{range}$
  - or there exist $\eta \in [\kappa]$, $\boldsymbol{\delta} \in \{0,1\}^{\ell^* - m}$, $l \in [0, L]$ such that

$$\tilde{y}_\eta(\boldsymbol{\delta}) \bmod p_l \neq \iota(\tilde{y}'_\eta(\boldsymbol{\delta})) \bmod p_l,$$

  - or $\exists \eta \in [\kappa]$ such that $\tilde{y}_\eta(\hat{\boldsymbol{s}}) \neq \tilde{\mu}(\hat{\boldsymbol{r}}_\eta, \hat{\boldsymbol{s}}) \vee \tilde{y}'_\eta(\boldsymbol{s}') \neq \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{s}')$.
- For $\mathsf{tr}_{2+k} = \mathsf{tr}_2|\mathsf{tr}'_k$ where, for $1 \leq k \leq \mathsf{r}_{range}$, tr'_k is the partial transcript of $\hat{\Pi}^{\mathbb{F}}_{range}$ at the k-th round: $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k}) = 0$ iff
  - $\mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_k) = 0$
  - or there exist $\eta \in [\kappa]$, $\boldsymbol{\delta} \in \{0,1\}^{\ell^* - m}$, $l \in [0, L]$ such that

$$\tilde{y}_\eta(\boldsymbol{\delta}) \bmod p_l \neq \iota(\tilde{y}'_\eta(\boldsymbol{\delta})) \bmod p_l,$$

  - or $\exists \eta \in [\kappa]$ such that $\tilde{y}_\eta(\hat{\boldsymbol{s}}) \neq \tilde{\mu}(\hat{\boldsymbol{r}}_\eta, \hat{\boldsymbol{s}}) \vee \tilde{y}'_\eta(\boldsymbol{s}') \neq \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{s}')$.

Let us now show that this state function satisfies the three properties of RBR-soundness.

First, it is easy to see that by construction, if $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \notin \hat{\mathcal{R}}_{range}$ then $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0$.

Second, for any *complete* transcript $\mathsf{tr} = \mathsf{tr}_{2+\mathsf{r}_{range}} = \mathsf{tr}_2|\mathsf{tr}'_{\mathsf{r}_{range}}$, by construction we have that if $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}) = 0$, then either $\mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_{\mathsf{r}_{range}}) = 0$, which means that the verifier of $\hat{\Pi}^{\mathbb{F}}_{range}$ rejects (by the RBR-soundness of $\hat{\Pi}^{\mathbb{F}}_{range}$), or at least one of the verifier's checks fails. In any of these cases, $\hat{\Pi}_{range}.\mathcal{V}$ rejects.

Third, we show that if state rejects on a partial transcript of round i, for any prover's message of the round $i + 1$, we have a concrete (negligible) bound on the probability that state accepts on the next partial transcript. We show this property round by round.

Claim. If $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0$, then for any prover's message $[\![\tilde{\mu}']\!]$ we have that

$$\Pr_{\boldsymbol{r}_0,\ldots,\boldsymbol{r}_{\kappa-1} \leftarrow\$ \Sigma^m}[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1 = ([\![\tilde{\mu}']\!]|(\boldsymbol{r}_0,\ldots,\boldsymbol{r}_{\kappa-1}))) = 1] \leq \left(\frac{m}{\sigma}\right)^\kappa.$$

Proof. By definition, $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0$ iff $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]) \notin \mathcal{L}_{\hat{\mathcal{R}}_{range}}$, that is there exist $\boldsymbol{\delta}^* \in \{0,1\}^{\ell^*-m}, \boldsymbol{\phi}^* \in \{0,1\}^m$ such that $\tilde{\mu}(\boldsymbol{\phi}^*, \boldsymbol{\delta}^*) \notin \boldsymbol{t}_\beta$. On the other hand, state accepting on tr_1 means that:
(i) $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range}$, i.e., for all $\boldsymbol{\delta} \in \{0,1\}^{\ell^*-m}, \boldsymbol{\phi} \in \{0,1\}^m$, $\tilde{\mu}'(\boldsymbol{\phi}, \boldsymbol{\delta}) \in \boldsymbol{t}'_\beta$;
(ii) for all $\eta \in [\kappa]$, $\boldsymbol{\delta} \in \{0,1\}^{\ell^*-m}$: $f_{\boldsymbol{\delta}}(\pi_q(\boldsymbol{r}_\eta)) = 0 \bmod q$.

From (i) and the definition of $\tilde{\mu}''$, we have $\tilde{\mu}''(\boldsymbol{\phi}, \boldsymbol{\delta}) = \iota(\tilde{\mu}'(\pi_p(\boldsymbol{\phi}, \boldsymbol{\delta}))) \in \iota(\boldsymbol{t}'_\beta)$. Since $\boldsymbol{t}'_\beta = \pi_p(\boldsymbol{t}_\beta)$, $\beta < p/2$ (from equation (23)), and using ι's definition, we have $\iota(\pi_p(\boldsymbol{t}_\beta)) = \boldsymbol{t}_\beta$, and thus $\tilde{\mu}''(\boldsymbol{\phi}, \boldsymbol{\delta}) \in \boldsymbol{t}_\beta$ and the polynomial $f_{\boldsymbol{\delta}^*}(\boldsymbol{X})$ is nonzero.

Therefore,

$$\begin{aligned}
&\Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 1 \mid \mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0] \\
&= \Pr[(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range} \bigwedge_{\substack{\eta\in[\kappa] \\ \boldsymbol{\delta}\in\{0,1\}^{\ell^*-m}}} f_{\boldsymbol{\delta}}(\pi_q(\boldsymbol{r}_\eta)) = 0 \mid (\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \notin \hat{\mathcal{R}}_{range}] \\
&\leq \Pr[\bigwedge_{\substack{\eta\in[\kappa] \\ \boldsymbol{\delta}\in\{0,1\}^{\ell^*-m}}} f_{\boldsymbol{\delta}}(\pi_q(\boldsymbol{r}_\eta)) = 0 \mid (\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range} \wedge (\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \notin \hat{\mathcal{R}}_{range}] \\
&\leq \Pr[\bigwedge_{\eta\in[\kappa]} f_{\boldsymbol{\delta}^*}(\pi_q(\boldsymbol{r}_\eta)) = 0 \mid \exists \boldsymbol{\delta}^* \in \{0,1\}^{\ell^*-m} : f_{\boldsymbol{\delta}^*}(\boldsymbol{X}) \neq 0] \\
&= \prod_{\eta\in[\kappa]} \Pr_{\boldsymbol{r}_\eta \leftarrow\$ \Sigma^m}[f_{\boldsymbol{\delta}^*}(\pi_q(\boldsymbol{r}_\eta)) = 0 \mid f_{\boldsymbol{\delta}^*}(\boldsymbol{X}) \neq 0].
\end{aligned}$$

By generalized Schwartz-Zippel 1, $\Pr_{\boldsymbol{r}_\eta \leftarrow\$ \Sigma^m}[f_{\boldsymbol{\delta}^*}(\pi_q(\boldsymbol{r}_\eta)) = 0 \mid f_{\boldsymbol{\delta}^*}(\boldsymbol{X}) \neq 0] \leq \frac{m}{\sigma}$, and the claimed bound follows by considering the probability that this occurs for κ independent samples; by equation (24) this bound is $\approx 2^{-\lambda}$. □

Claim. If $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0$, then for any prover's message $(\tilde{y}_0, \tilde{y}'_0, \ldots, \tilde{y}_{\kappa-1}, \tilde{y}'_{\kappa-1})$ we have that

$$\Pr_{\hat{\boldsymbol{s}}, \boldsymbol{s}'}[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2 = (\tilde{y}_0, \tilde{y}'_0, \ldots, \tilde{y}_{\kappa-1}, \tilde{y}'_{\kappa-1})|\hat{\boldsymbol{s}}, \boldsymbol{s}')) = 1] \leq \frac{\ell^* - m}{|\hat{S}_d|} + \frac{\ell^* - m}{p}.$$

Proof. The event $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1$ is the intersection of the following events:

$$(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range} \bigwedge_{\eta\in[\kappa]} \mathsf{V}_{3,\eta} \bigwedge_{\substack{\eta\in[\kappa], l\in[0,L] \\ \boldsymbol{\delta}\in\{0,1\}^{\ell^*-m}}} \mathsf{V}_{4,\eta,\boldsymbol{\delta},l}$$

where

$$\begin{aligned}
\mathsf{V}_{3,\eta} &:= (\tilde{y}_\eta(\hat{\boldsymbol{s}}) = \tilde{\mu}(\hat{\boldsymbol{r}}_\eta, \hat{\boldsymbol{s}})) \wedge (\tilde{y}'_\eta(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{s}')) \\
\mathsf{V}_{4,\eta,\boldsymbol{\delta},l} &:= (\tilde{y}_\eta(\boldsymbol{\delta}) \bmod p_l = \iota(\tilde{y}'_\eta(\boldsymbol{\delta})) \bmod p_l)
\end{aligned}$$

Let $\mathcal{E}_2$ be the event "$\bigwedge_{\eta\in[\kappa]}(\tilde{y}_\eta(X_{m+1},\ldots,X_{\ell^*}) = \tilde{\mu}(\hat{\boldsymbol{r}}_\eta, X_{m+1},\ldots,X_{\ell^*})) \wedge (\tilde{y}'_\eta(X_{m+1},\ldots,X_{\ell^*}) = \tilde{\mu}'(\boldsymbol{r}'_\eta, X_{m+1},\ldots,X_{\ell^*}))$". By partitioning over $\mathcal{E}_2$, we upper bound $\Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1 \mid \mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0]$ by the sum $\Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1 \mid \mathcal{E}_2, \mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0] + \Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1 \mid \bar{\mathcal{E}}_2, \mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0]$

The second term can be upper bounded as follows (where we use $\boldsymbol{X}$ to compactly denote $X_{m+1},\ldots,X_{\ell^*}$):

$$
\begin{aligned}
&\Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1 \mid \bar{\mathcal{E}}_2, \mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0] \\
&\leq \Pr[\bigwedge_{\eta\in[\kappa]} \mathsf{V}_{3,\eta} \mid \exists \bar{\eta} : \tilde{y}_{\bar{\eta}}(\boldsymbol{X}) \neq \tilde{\mu}(\hat{\boldsymbol{r}}_{\bar{\eta}}, \boldsymbol{X}) \vee \tilde{y}'_{\bar{\eta}}(\boldsymbol{X}) \neq \tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \boldsymbol{X})] \\
&\leq \Pr[\tilde{y}_{\bar{\eta}}(\hat{\boldsymbol{s}}) = \tilde{\mu}(\hat{\boldsymbol{r}}_{\bar{\eta}}, \hat{\boldsymbol{s}}) \wedge \tilde{y}'_{\bar{\eta}}(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \boldsymbol{s}') \mid \exists \bar{\eta} : \tilde{y}_{\bar{\eta}}(\boldsymbol{X}) \neq \tilde{\mu}(\hat{\boldsymbol{r}}_{\bar{\eta}}, \boldsymbol{X}) \vee \tilde{y}'_{\bar{\eta}}(\boldsymbol{X}) \neq \tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \boldsymbol{X})] \\
&\leq \Pr[\tilde{y}_{\bar{\eta}}(\hat{\boldsymbol{s}}) = \tilde{\mu}(\hat{\boldsymbol{r}}_{\bar{\eta}}, \hat{\boldsymbol{s}}) \mid \tilde{y}_{\bar{\eta}}(\boldsymbol{X}) \neq \tilde{\mu}(\hat{\boldsymbol{r}}_{\bar{\eta}}, \boldsymbol{X})] + \Pr[\tilde{y}'_{\bar{\eta}}(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \boldsymbol{s}') \mid \tilde{y}'_{\bar{\eta}}(\boldsymbol{X}) \neq \tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \boldsymbol{X})] \\
&\leq \frac{\ell^* - m}{|\hat{S}_d|} + \frac{\ell^* - m}{p}
\end{aligned}
$$

where the latter inequality follows by applying Schwartz-Zippel over H_q (resp. $\mathbb{Z}_p$) with the random $\hat{s} \leftarrow\!\$\ \hat{S}_d$ (resp. $s' \leftarrow\!\$\ \mathbb{Z}_p$).

For the first term, we argue that $\Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1 \mid \mathcal{E}_2, \mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0] = 0$ as follows.

$$
\begin{aligned}
&\Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1 \mid \mathcal{E}_2, \mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0] \\
&\leq \Pr\left[\begin{array}{l|l} (\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range} & \\ \bigwedge_{\eta\in[\kappa], l\in[0,L]} \mathsf{V}_{4,\eta,\boldsymbol{\delta},l} & \mathcal{E}_2 \wedge ((\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \notin \hat{\mathcal{R}}_{range} \vee \exists \bar{\eta}, \bar{l}, \bar{\boldsymbol{\delta}} : f_{\bar{\boldsymbol{\delta}}}(\hat{\boldsymbol{r}}_{\bar{\eta}}) \neq 0 \bmod p_{\bar{l}}) \\ \boldsymbol{\delta}\in\{0,1\}^{\ell^*-m} & \end{array}\right] \\
&\leq \Pr\left[\begin{array}{l|l} \bigwedge_{\substack{\eta\in[\kappa], l\in[0,L] \\ \boldsymbol{\delta}\in\{0,1\}^{\ell^*-m}}} \mathsf{V}_{4,\eta,\boldsymbol{\delta},l} & \mathcal{E}_2 \wedge (\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range} \wedge \exists \bar{\eta}, \bar{l}, \bar{\boldsymbol{\delta}} : f_{\bar{\boldsymbol{\delta}}}(\hat{\boldsymbol{r}}_{\bar{\eta}}) \neq 0 \bmod p_{\bar{l}} \end{array}\right] \\
&\leq \Pr\left[\tilde{\mu}(\hat{\boldsymbol{r}}_{\bar{\eta}}, \bar{\boldsymbol{\delta}}) \bmod p_{\bar{l}} = \iota(\tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \bar{\boldsymbol{\delta}})) \bmod p_{\bar{l}} \;\middle|\; \begin{array}{l} (\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range} \wedge \\ \exists \bar{\eta}, \bar{l}, \bar{\boldsymbol{\delta}} : f_{\bar{\boldsymbol{\delta}}}(\boldsymbol{r}_{\bar{\eta}}) \neq 0 \bmod p_{\bar{l}} \end{array}\right] \\
&\leq \Pr\left[\tilde{\mu}(\hat{\boldsymbol{r}}_{\bar{\eta}}, \bar{\boldsymbol{\delta}}) \bmod p_{\bar{l}} = \tilde{\mu}''(\boldsymbol{r}_{\bar{\eta}}, \bar{\boldsymbol{\delta}}) \bmod p_{\bar{l}} \;\middle|\; \begin{array}{l} (\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range} \wedge \\ \exists \bar{\eta}, \bar{l}, \bar{\boldsymbol{\delta}} : f_{\bar{\boldsymbol{\delta}}}(\boldsymbol{r}_{\bar{\eta}}) \neq 0 \bmod p_{\bar{l}} \end{array}\right] = 0
\end{aligned}
$$

where the third inequality follows by applying $\mathcal{E}_2$ to the values in the event $\mathsf{V}_{4,\eta,\boldsymbol{\delta},l}$ and then considering only the event for the tuple $\bar{\eta}, \bar{l}, \bar{\boldsymbol{\delta}}$ for which $f_{\bar{\boldsymbol{\delta}}}(\hat{\boldsymbol{r}}_{\bar{\eta}}) \neq 0 \bmod p_{\bar{l}}$); the fourth inequality follows by applying Proposition 1 and that $|\tilde{\mu}''(\boldsymbol{r}_{\bar{\eta}}, \bar{\boldsymbol{\delta}})| < p/2$ (which is implied by the fact that $\boldsymbol{r}_\eta \in \Sigma$ and $0 \leq \tilde{\mu}''(\boldsymbol{b}) < \beta$). Finally, the last probability is 0, since we are conditioning on $f_{\bar{\boldsymbol{\delta}}}(\boldsymbol{r}_{\bar{\eta}}) \neq 0 \bmod p_{\bar{l}}$). □

To prove the last case, recall that, once tr_2 is fixed, so is the event

$$E := \exists\, \eta \in [\kappa],\, \boldsymbol{\delta} \in \{0,1\}^{\ell^*-m},\, l \in [0, L] : \neg\mathsf{V}_{3,\eta} \vee \neg\mathsf{V}_{4,\eta,\boldsymbol{\delta},l},$$

that is, the event that at least one of the verifier's checks of step 2 fails. We also let $\mathsf{tr}'_0 = \emptyset$.

Claim. For every $k = 0, \ldots, \mathsf{r}_{range} - 1$, if $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k}) = 0$, then for any prover's message π of the $(k+1)$-th round of $\hat{\Pi}^{\mathbb{F}}_{range}$ we have that

$$\Pr_{\rho}[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k+1}) = 1] \leq \delta'_{range}, \quad \text{where } \mathsf{tr}_{2+k+1} = \mathsf{tr}_2 | \mathsf{tr}'_k | \pi | \rho,$$

and where the probability is over the $(k+1)$-th verifier's challenge ρ of $\hat{\Pi}^{\mathbb{F}}_{range}$.

Proof. Since $k + 1 \geq 1$, the definition of state on tr_{2+k+1} gives

$$\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k+1}) = 1 \iff \mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_k | \pi | \rho) = 1 \ \wedge \ \bar{E}.$$

Moreover, E is determined by tr_2, hence it is fixed and independent of ρ. We distinguish two cases.

If E occurs, then $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k+1}) = 1$ never occurs and the probability of the claim is 0.

If E does not occur, then the event $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k}) = 0$ of the precondition implies $\mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_k) = 0$: for $k \geq 1$ this is immediate from the definition of state on tr_{2+k}, whereas for $k = 0$ the definition of state on tr_2 gives $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]) \notin \mathcal{L}_{\hat{\mathcal{R}}_{range}}$, which in turn implies $\mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_0) = 0$ by the first property of RBR-soundness (Definition 6) applied to $\hat{\Pi}^{\mathbb{F}}_{range}$. Therefore, for all $0 \leq k < \mathsf{r}_{range}$,

$$\begin{aligned}
&\Pr_{\rho}[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k+1}) = 1] \\
&= \Pr_{\rho}[\mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_k | \pi | \rho) = 1 \wedge \bar{E}] \\
&\leq \Pr_{\rho}[\mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_k | \pi | \rho) = 1] \ \leq \ \delta'_{range},
\end{aligned}$$

where the last inequality follows by the third property of RBR-soundness applied to $\hat{\Pi}^{\mathbb{F}}_{range}$ and to the partial transcript tr'_k, on which state' evaluates to 0. □

The final RBR-soundness error is the max of the three per-round bounds, namely $\max\left\{\left(\frac{m}{\sigma}\right)^{\kappa}; \frac{\ell^*-m}{|\hat{S}_d|} + \frac{\ell^*-m}{p}; \delta'_{range}\right\}$, where the first term is $\approx 2^{-\lambda}$ by equation (24).

D.3 Efficiency of $\hat{\Pi}_{range}$

In addition to the cost of running the PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$, the efficiency of this protocol is as follows.

- The prover sends 1 oracle, $[\![\tilde{\mu}']\!]$, whose size is 2^{ℓ^*} elements of $\mathbb{Z}_p$; in addition the prover sends $\kappa 2^{\ell^*-m}$ elements of $\mathbb{Z}_q$ and as many elements of $\mathbb{Z}_p$.
- The proving time is $O(\kappa \cdot 2^{\ell^*})$, which covers the computation of the partial evaluations of $\tilde{\mu}$ and $\tilde{\mu}'$ sent in round 2.

- The verifier time is $O(\kappa \cdot 2^{\ell^* - m})$.
- The number of verifier's queries is 2κ, which includes κ queries to $[\![\tilde{\mu}]\!]$ and κ queries to $[\![\tilde{\mu}']\!]$.

Instantiating the concrete parameters. If we target the use of a prime p of 200 bits, and assume $\ell = 20$ and $\beta = 2^{20}$, then we can take for example $m = \ell - \lceil \log \ell \rceil = 15$, $\sigma = 2^{11}$, $\kappa = 18$. This gives $\tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta}) < 2^m \cdot \beta \cdot \sigma^m < p/2 \approx 2^{199}$.

Looking specifically at the parameters that fit our use of range proofs for the largest circuit in our experiments, then we have: $\ell = 9$, $\beta = 2^{11}$, $\gamma = 2$. To fit a 200-bits prime p, then we can set the parameters of the PIOP as $m = \ell$, $\sigma = 2^{19}$, $\kappa = 8$. These choices satisfy the correctness and soundness bounds since

$$\tilde{\mu}''(\boldsymbol{r}_\eta, \boldsymbol{\delta}) < 2^m \cdot \beta \cdot \sigma^m = 2^{191} < p/2 \approx 2^{199}$$

$$\left(\frac{m}{\sigma}\right)^\kappa = \left(\frac{9}{2^{19}}\right)^8 \leq 2^{-120}$$

and $m = \ell$ clearly ensures the succinctness bound.

D.4 A PIOP for Range Checks over a Finite Field

The protocol relies on the following lemma from [CCC+25], which simplifies Spartan's memory checking technique [Set20] to the case of lookups in $\boldsymbol{t}_\beta$.

Lemma 7 ([CCC+25]). *Let p, ℓ^* be positive integers such that $p > 2^{\ell^*}$. Let $\mathbb{F}$ be a field of characteristic p, let $\tilde{\mu}$ be an ℓ^*-variate multilinear polynomial over $\mathbb{F}$, and let $\tilde{t}_\beta$ be the MLE of a table of size $\beta = 2^{b^*}$ whose elements are all distinct. Then we have that $\forall \boldsymbol{i} \in \{0,1\}^{\ell^*}$ $\tilde{\mu}(\boldsymbol{i}) \in \{\tilde{t}_\beta(\boldsymbol{j}) : \boldsymbol{j} \in \{0,1\}^{b^*}\}$ if and only if there exist polynomials $\mathsf{read_ts} \in \mathbb{F}[X_1, \ldots, X_{\ell^*}], \mathsf{final_cts} \in \mathbb{F}[X_1, \ldots, X_{b^*}]$ such that $\mathsf{WS} = \mathsf{RS} \cup \mathsf{S}$ where*

$$\begin{aligned}
\mathsf{WS} &= \{(\tilde{t}_\beta(\boldsymbol{j}), 0) : \boldsymbol{j} \in \{0,1\}^{b^*}\} \cup \{(\tilde{\mu}(\boldsymbol{i}), \mathsf{read_ts}(\boldsymbol{i}) + 1) : \boldsymbol{i} \in \{0,1\}^{\ell^*}\} \\
\mathsf{RS} &= \{(\tilde{\mu}(\boldsymbol{i}), \mathsf{read_ts}(\boldsymbol{i})) : \boldsymbol{i} \in \{0,1\}^{\ell^*}\} \\
\mathsf{S} &= \{(\tilde{t}_\beta(\boldsymbol{j}), \mathsf{final_cts}(\boldsymbol{j})) : \boldsymbol{j} \in \{0,1\}^{b^*}\}.
\end{aligned}$$

Following the lemma, the PIOP prover builds the multilinear polynomials $\mathsf{read_ts}, \mathsf{final_cts}$ and aims to convince the verifier that $\mathsf{WS} = \mathsf{RS} \cup \mathsf{S}$.

To prove this multiset equality, the verifier sends random challenges $\sigma, \tau \leftarrow\!\!\$\ \mathbb{F}$, the prover computes $\overline{\mathsf{WS}}_t, \overline{\mathsf{WS}}_h, \overline{\mathsf{RS}}, \overline{\mathsf{S}}$ as follows

$$\overline{\mathsf{WS}} = \overbrace{\prod_{\boldsymbol{j} \in \{0,1\}^{b^*}} (\tilde{t}_\beta(\boldsymbol{j}) - \tau)}^{\overline{\mathsf{WS}}_t} \overbrace{\prod_{\boldsymbol{i} \in \{0,1\}^{\ell^*}} (\tilde{\mu}(\boldsymbol{i}) + \sigma(\mathsf{read_ts}(\boldsymbol{i}) + 1) - \tau)}^{\overline{\mathsf{WS}}_\mu} \tag{26}$$

$$\overline{\mathsf{RS}} = \prod_{\boldsymbol{i} \in \{0,1\}^{\ell^*}} (\tilde{\mu}(\boldsymbol{i}) + \sigma \cdot \mathsf{read_ts}(\boldsymbol{i}) - \tau) \tag{27}$$

$$\overline{\mathsf{S}} = \prod_{\boldsymbol{j} \in \{0,1\}^{b^*}} (\tilde{t}_\beta(\boldsymbol{j}) + \sigma \cdot \mathsf{final_cts}(\boldsymbol{j}) - \tau), \tag{28}$$

PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$ for $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]; \tilde{\mu}) \in \hat{\mathcal{R}}_{range}$

1. Let $\tilde{\mu} \in \mathbb{F}^{(\leq 1)}[X_1, \ldots, X_{\ell^*}]$. $\mathcal{P}$ constructs $\mathsf{read_ts} \in \mathbb{F}^{(\leq 1)}[X_1, \ldots, X_{\ell^*}]$, $\mathsf{final_cts} \in \mathbb{F}^{(\leq 1)}[X_1, \ldots, X_{b^*}]$ following Lemma 7 applied to $\tilde{\mu}$ and $\tilde{t}_\beta$.
   Send $[\![\mathsf{read_ts}]\!], [\![\mathsf{final_cts}]\!]$.
   $\mathcal{V}$ replies with random $\sigma, \tau \leftarrow\!\!\$\ \mathbb{F}$.
2. $\mathcal{P}$ constructs multilinear polynomials $f^{(1)}_{\mathsf{WS}_\mu}, f^{(1)}_{\mathsf{RS}}, f^{(1)}_{\mathsf{WS}_t}, f^{(1)}_{\mathsf{S}}$ such that
   for $\alpha = \mathsf{WS}_\mu, \mathsf{RS}$: $f^{(1)}_\alpha(\boldsymbol{i}) = f^{(i_1)}_\alpha(\boldsymbol{i}', 0) \cdot f^{(i_1)}_\alpha(\boldsymbol{i}', 1) \ \ \forall \boldsymbol{i} \in \{0,1\}^{\ell^*}$
   for $\alpha = \mathsf{WS}_t, \mathsf{S}$: $f^{(1)}_\alpha(\boldsymbol{j}) = f^{(j_1)}_\alpha(\boldsymbol{j}', 0) \cdot f^{(j_1)}_\alpha(\boldsymbol{j}', 1) \ \ \forall \boldsymbol{j} \in \{0,1\}^{b^*}$
   where for a given $\boldsymbol{i} \in \{0,1\}^m$, $\boldsymbol{i}'$ denotes $(i_2, \ldots, i_m)$, and the polynomials $f^{(0)}_\alpha$ are

$$f^{(0)}_{\mathsf{WS}_\mu} = \tilde{\mu} + \sigma \cdot (\mathsf{read_ts} + 1) - \tau, \qquad f^{(0)}_{\mathsf{RS}} = \tilde{\mu} + \sigma \cdot \mathsf{read_ts} - \tau,$$
$$f^{(0)}_{\mathsf{WS}_t} = \tilde{t}_\beta - \tau, \qquad f^{(0)}_{\mathsf{S}} = \tilde{t}_\beta + \sigma \cdot \mathsf{final_cts} - \tau$$

   $\mathcal{P}$ sends $[\![f^{(1)}_{\mathsf{WS}_\mu}]\!], [\![f^{(1)}_{\mathsf{RS}}]\!], [\![f^{(1)}_{\mathsf{WS}_t}]\!], [\![f^{(1)}_{\mathsf{S}}]\!]$.
   $\mathcal{V}$ replies with random $\chi \leftarrow\!\!\$\ \mathbb{F}$, $\boldsymbol{\rho} \leftarrow\!\!\$\ \mathbb{F}^{\ell^*}, \boldsymbol{\xi} \leftarrow\!\!\$\ \mathbb{F}^{b^*}$.
3. $\mathcal{P}$ and $\mathcal{V}$ run Π_{sum} for $((0, [\![\mu^*]\!]); \mu^*) \in \mathcal{R}_{sum}$ and $((0, [\![t^*]\!]); t^*) \in \mathcal{R}_{sum}$ where $[\![\mu^*]\!], [\![t^*]\!]$ are the virtual oracles: // see Figure 9 for the definition of g
   $[\![t^*]\!] = g_{\chi,\boldsymbol{\xi}}\left([\![\tilde{t}_\beta]\!] - \tau, [\![\tilde{t}_\beta]\!] + \sigma \cdot [\![\mathsf{final_cts}]\!] - \tau, [\![f^{(1)}_{\mathsf{WS}_t}]\!], [\![f^{(1)}_{\mathsf{S}}]\!]\right)$
   $[\![\mu^*]\!] = g_{\chi,\boldsymbol{\rho}}\big([\![\tilde{\mu}]\!] + \sigma \cdot ([\![\mathsf{read_ts}]\!] + 1) - \tau, [\![\tilde{\mu}]\!] + \sigma \cdot [\![\mathsf{read_ts}]\!] - \tau, [\![f^{(1)}_{\mathsf{WS}_\mu}]\!], [\![f^{(1)}_{\mathsf{RS}}]\!]\big)$.

   $\mathcal{V}$ queries $[\![f^{(1)}_{\mathsf{WS}_\mu}]\!], [\![f^{(1)}_{\mathsf{RS}}]\!]$ on $(\mathbf{1}, 0) \in \mathbb{F}^{\ell^*}$, $[\![f^{(1)}_{\mathsf{WS}_t}]\!], [\![f^{(1)}_{\mathsf{S}}]\!]$ on $(\mathbf{1}, 0) \in \mathbb{F}^{b^*}$, and then checks if $f^{(1)}_{\mathsf{WS}_t}(\mathbf{1}, 0) \cdot f^{(1)}_{\mathsf{WS}_\mu}(\mathbf{1}, 0) = f^{(1)}_{\mathsf{RS}}(\mathbf{1}, 0) \cdot f^{(1)}_{\mathsf{S}}(\mathbf{1}, 0)$.

Figure 8: PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$, where $\boldsymbol{t}_\beta \in \mathbb{F}^{2^{b^*}}$, $\tilde{\mu} \in \mathbb{F}^{(\leq 1)}[X_1, \ldots, X_{\ell^*}]$.

Function $g_{\chi,\boldsymbol{\rho}}(f^{(0)}_1, f^{(0)}_2, f^{(1)}_1, f^{(1)}_2)$

for $k = 1, 2$, let $f_k(X_0, \boldsymbol{X}) \leftarrow (1 - X_0) \cdot f^{(0)}_k(\boldsymbol{X}) + X_0 \cdot f^{(1)}_k(\boldsymbol{X})$
Return $(f_1(1, \boldsymbol{X}) - f_1(\boldsymbol{X}, 0) \cdot f_1(\boldsymbol{X}, 1) + \chi \cdot (f_2(1, \boldsymbol{X}) - f_2(\boldsymbol{X}, 0) \cdot f_2(\boldsymbol{X}, 1))) \cdot \tilde{eq}(\boldsymbol{\rho}, \boldsymbol{X})$

Figure 9: Function g used by PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$

and the verifier checks that $\overline{\mathsf{WS}}_t \cdot \overline{\mathsf{WS}}_\mu = \overline{\mathsf{RS}} \cdot \overline{\mathsf{S}}$. The prover must finally convince the verifier that each of the four values is computed correctly; this is done using Quark's grand product.

Lemma 8 (Grand product Lemma [SL20]). *$P = \prod_{\boldsymbol{i} \in \{0,1\}^\ell} v(\boldsymbol{i})$ if and only if there exists a multilinear polynomial f in $\ell + 1$ variables such that $f(1, \ldots, 1, 0) = P$, and $\forall \boldsymbol{i} \in \{0,1\}^\ell$ it holds $f(0, \boldsymbol{i}) = v(\boldsymbol{i})$ and $f(1, \boldsymbol{i}) = f(\boldsymbol{i}, 0) \cdot f(\boldsymbol{i}, 1)$.*

In Fig. 8 we present the full specification of the PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$ for $\hat{\mathcal{R}}_{range}$.

Below, we state and prove the security of $\hat{\Pi}^{\mathbb{F}}_{range}$. Our proof is inspired to that of [CCC+25, Lemma 3.2]; here we specialize it to an instantiation over a finite field and prove round-by-round soundness rather than soundness.

Theorem 8. *Let p be a prime number and let us consider the finite field $\mathbb{F} = \mathbb{Z}_p$. Let ℓ^*, β be positive integers such that $\beta = 2^{b^*} < p$ and $2^{\ell^*} < p$. Then $\hat{\Pi}^{\mathbb{F}}_{range}$ is a PIOP for $\hat{\mathcal{R}}_{range}$, with table $\boldsymbol{t}_\beta = \{0, \ldots, \beta - 1\}$, that has perfect completeness and RBR-knowledge error $\delta'_{range} = \frac{2^{\ell^*} + 2^{b^*}}{p}$.*

Proof. We show that $\hat{\Pi}^{\mathbb{F}}_{range}$ is RBR-sound with error $\delta'_{range} = \frac{2^{\ell^*} + 2^{b^*}}{p}$. RBR-knowledge with the same error follows by applying Lemma 6, using the fact that $\hat{\mathcal{R}}_{range}$'s witness is only the polynomial $\tilde{\mu}$ that is provided as an oracle in the statement.

Notation. Given the polynomials $\tilde{\mu}, \tilde{t}_\beta, \mathsf{read_ts}, \mathsf{final_cts}$ available after the first round, let $\mathsf{WS}, \mathsf{RS}, \mathsf{S}$ be the multisets of pairs as in the statement of Lemma 7, and let $\mathsf{WS}_t, \mathsf{WS}_\mu$ be the following partition of WS:

$$\mathsf{WS}_t = \{(\tilde{t}_\beta(\boldsymbol{j}), 0) : \boldsymbol{j} \in \{0,1\}^{b^*}\} \qquad \mathsf{WS}_\mu = \{(\tilde{\mu}(\boldsymbol{i}), \mathsf{read_ts}(\boldsymbol{i}) + 1) : \boldsymbol{i} \in \{0,1\}^{\ell^*}\}$$

For $\alpha \in \{\mathsf{WS}_t, \mathsf{WS}_\mu, \mathsf{RS}, \mathsf{S}\}$ let use define the polynomials

$$P_\alpha(Z, Z') = \prod_{(a_0, a_1) \in \alpha} (a_0 + a_1 \cdot Z - Z') \in \mathbb{F}[Z, Z'],$$

and let

$$P(Z, Z') = P_{\mathsf{WS}_t}(Z, Z') \cdot P_{\mathsf{WS}_\mu}(Z, Z') - P_{\mathsf{RS}}(Z, Z') \cdot P_{\mathsf{S}}(Z, Z').$$

Note that $\deg P \leq 2^{\ell^*} + 2^{b^*}$ and that the coefficients of P are determined by $\tilde{t}_\beta, \tilde{\mu}, \mathsf{read_ts}, \mathsf{final_cts}$, hence they are fixed before $\mathcal{V}$ sends σ, τ.

For $\alpha \in \{\mathsf{WS}_t, \mathsf{WS}_\mu, \mathsf{RS}, \mathsf{S}\}$ let $f_\alpha(X_0, \boldsymbol{X}) = (1 - X_0) \cdot f^{(0)}_\alpha(\boldsymbol{X}) + X_0 \cdot f^{(1)}_\alpha(\boldsymbol{X})$, where $f^{(0)}_\alpha$ is as in Figure 8 and $f^{(1)}_\alpha$ is the polynomial sent by $\mathcal{P}$ as an oracle in round 2. Let $\bar{\alpha} = f^{(1)}_\alpha(\mathbf{1}, 0) = f_\alpha(1, \ldots, 1, 0)$ denote the values that $\mathcal{V}$ queries in step 6. We define two events, both determined by the oracles available after round 2:

- GP: for every α and every $\boldsymbol{i}$ in the corresponding hypercube ($\{0,1\}^{\ell^*}$ for $\alpha = \mathsf{WS}_\mu, \mathsf{RS}$ and $\{0,1\}^{b^*}$ for $\alpha = \mathsf{WS}_t, \mathsf{S}$) it holds $f_\alpha(1, \boldsymbol{i}) = f_\alpha(\boldsymbol{i}, 0) \cdot f_\alpha(\boldsymbol{i}, 1)$, i.e., the grand-product statement is satisfied;
- V_6: the check of step 6 passes, i.e., $\overline{\mathsf{WS}}_t \cdot \overline{\mathsf{WS}}_\mu = \overline{\mathsf{RS}} \cdot \overline{\mathsf{S}}$.

Finally, let state_{sum} be the state function defined by the RBR-soundness of Π_{sum}, and let us define state'_{sum} as the state function for the parallel execution of the two sumchecks Π_{sum} in step 3, which is defined to output 0 on a partial transcript of the parallel execution iff state_{sum} outputs 0 on at least one of its two instances.

Since $f_\alpha(\boldsymbol{X},0)\cdot f_\alpha(\boldsymbol{X},1)$ has degree 2 in each variable and $\tilde{\mathsf{eq}}(\boldsymbol{\rho},\boldsymbol{X})$ has degree 1, the virtual oracles μ^*, t^* have degree 3 in each variable. We use that Π_{sum}, run over $\mathbb{F}$ on a virtual oracle of degree d in each variable, is RBR-sound with per-round error $d/|\mathbb{F}|$; this is the round-by-round analogue of Theorem 5 and follows from the standard round-by-round analysis of the sum-check protocol [CCH+19]. Hence the per-round RBR-soundness error of the two instances of Π_{sum} run in step 3 is $3/p$. Let us write $\mathsf{r}_{sum} = \max(\ell^*, b^*)$ for the number of rounds of these Π_{sum} executions in step 3, so that $\mathsf{r}_{range} = \mathsf{r}_{sum} + 2$.

The state function. We define $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_i)$ on every (partial) transcript, from $i = 0$ to $i = 2 + \mathsf{r}_{sum}$, as follows:

- For $\mathsf{tr}_0 = \emptyset$: $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0$ iff $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]) \notin \mathcal{L}_{\hat{\mathcal{R}}_{range}}$.
- For $\mathsf{tr}_1 = [\![\mathsf{read_ts}]\!]|[\![\mathsf{final_cts}]\!]|(\sigma,\tau)$: $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0$ iff $P(\sigma,\tau) \neq 0$.
- For $\mathsf{tr}_2 = \mathsf{tr}_1|\left([\![f^{(1)}_{\mathsf{WS}_\mu}]\!], [\![f^{(1)}_{\mathsf{RS}}]\!], [\![f^{(1)}_{\mathsf{WS}_t}]\!], [\![f^{(1)}_{\mathsf{S}}]\!]\right)|(\chi, \boldsymbol{\rho}, \boldsymbol{\xi})$: $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 0$ iff $\neg \mathsf{V}_6$, or $((0, [\![\mu^*]\!]); \mu^*) \notin \mathcal{R}_{sum}$, or $((0, [\![t^*]\!]); t^*) \notin \mathcal{R}_{sum}$.
- For $\mathsf{tr}_{2+k} = \mathsf{tr}_2|\mathsf{tr}''_k$, where tr''_k is the partial transcript of Π_{sum} parallel execution at the k-th round, $1 \leq k \leq \mathsf{r}_{sum}$: $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k}) = 0$ iff $\neg \mathsf{V}_6$ or $\mathsf{state}'_{sum}(0, ([\![\mu^*]\!], [\![t^*]\!]), \mathsf{tr}''_k) = 0$.

We also let $\mathsf{tr}''_0 = \emptyset$. Note that V_6 is determined by tr_2, so it is fixed in all the rounds of step 5.

The first property of RBR-soundness holds by construction: if $(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!]) \notin \mathcal{L}_{\hat{\mathcal{R}}_{range}}$ then $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0$.

For the second property, let $\mathsf{tr} = \mathsf{tr}_{2+\mathsf{r}_{sum}}$ be a complete transcript with $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}) = 0$. Then either $\neg \mathsf{V}_6$, in which case the check of step 6 fails, or $\mathsf{state}_{sum}(\mathsf{tr}''_{\mathsf{r}_{sum}}) = 0$, in which case the verifier of Π_{sum} rejects by the second property of RBR-soundness applied to Π_{sum}. In both cases $\hat{\Pi}^{\mathbb{F}}_{range}.\mathcal{V}$ rejects.

For the third property, we show below that if state rejects on a partial transcript of round i, for any prover's message of round $i+1$, there is a negligible probability that state turns into accept on the next partial transcript, over the random choice of the round-$(i+1)$ verifier's challenges.

Claim. If $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0$, then for any prover's message $[\![\mathsf{read_ts}]\!], [\![\mathsf{final_cts}]\!]$,

$$\Pr_{\sigma,\tau \leftarrow \$\mathbb{F}}[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 1] \leq \frac{2^{\ell^*} + 2^{b^*}}{p}.$$

Proof. By definition $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_0) = 0$ there exists $\boldsymbol{i}^* \in \{0,1\}^{\ell^*}$ such that $\tilde{\mu}(\boldsymbol{i}^*) \notin \boldsymbol{t}_\beta$. Since $\beta = 2^{b^*} < p$, the elements of $\boldsymbol{t}_\beta = \{0, \ldots, \beta - 1\}$ are pairwise distinct in $\mathbb{F}$, and since $2^{\ell^*} < p$ we can apply Lemma 7 to $\tilde{\mu}$ and $\tilde{t}_\beta$. This gives us that $\mathsf{WS} \neq \mathsf{RS} \cup \mathsf{S}$ as multisets, *for every* choice of the polynomials $\mathsf{read_ts}, \mathsf{final_cts}$ sent by $\mathcal{P}$ in round 1. By [Set20, Lemmas 7.2–7.4], we have that the polynomial $P(Z, Z') \neq 0 \in \mathbb{F}[Z, Z']$.

Since $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 1$ iff $P(\sigma,\tau) = 0$, the claim follows by applying the Schwartz-Zippell lemma to P, which is of total degree at most $2^{\ell^*} + 2^{b^*}$. □

Next we analyze the transition from tr_1 to tr_2.

Claim. If $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0$, then for any prover's message $[\![f^{(1)}_{\mathsf{WS}_\mu}]\!], [\![f^{(1)}_{\mathsf{RS}}]\!], [\![f^{(1)}_{\mathsf{WS}_t}]\!], [\![f^{(1)}_{\mathsf{S}}]\!]$,

$$\Pr_{\chi,\boldsymbol{\rho},\boldsymbol{\xi}}[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1] \leq \frac{\max(\ell^*, b^*) + 1}{p}.$$

Proof. Recall that $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0$ means $P(\sigma, \tau) \neq 0$, and that whether GP occurs is determined only by the prover's message, *independently* of $\chi, \boldsymbol{\rho}, \boldsymbol{\xi}$. We distinguish the two cases.

If GP occurs, then for every α the polynomial f_α satisfies the assumption of Lemma 8 (with polynomial $v = f^{(0)}_\alpha$). Indeed, $f_\alpha(0, \boldsymbol{X}) = f^{(0)}_\alpha(\boldsymbol{X})$ by construction and therefore the value $\bar{\alpha} = f_\alpha(1, \ldots, 1, 0) = \prod_{\boldsymbol{i}} f^{(0)}_\alpha(\boldsymbol{i})$. By the definition of the polynomials $f^{(0)}_\alpha$ we have $\prod_{\boldsymbol{i}} f^{(0)}_\alpha(\boldsymbol{i}) = P_\alpha(\sigma, \tau)$. Hence $\bar{\alpha} = P_\alpha(\sigma, \tau)$ for all four α, and thus V_6 would give $P(\sigma, \tau) = 0$, contradicting $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_1) = 0$. Therefore $\neg\mathsf{V}_6$ and $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1$ with probability 0.

If GP does not occur, there are a multiset α^* and a point $\boldsymbol{i}^*$ of the corresponding hypercube for which the grand-product statement fails. Belowe we analyze the case $\alpha^* \in \{\mathsf{WS}_\mu, \mathsf{RS}\}$; the case $\alpha^* \in \{\mathsf{WS}_t, \mathsf{S}\}$ is symmetric, following by considering $\boldsymbol{\xi}$ and b^* in place of $\boldsymbol{\rho}$ and ℓ^* respectively.

For $\boldsymbol{i} \in \{0,1\}^{\ell^*}$ let

$$A(\boldsymbol{i}) = f_{\mathsf{WS}_\mu}(1, \boldsymbol{i}) - f_{\mathsf{WS}_\mu}(\boldsymbol{i}, 0) \cdot f_{\mathsf{WS}_\mu}(\boldsymbol{i}, 1), \qquad B(\boldsymbol{i}) = f_{\mathsf{RS}}(1, \boldsymbol{i}) - f_{\mathsf{RS}}(\boldsymbol{i}, 0) \cdot f_{\mathsf{RS}}(\boldsymbol{i}, 1),$$

and let $\tilde{G}$ be the multilinear polynomial such that $\tilde{G}(\boldsymbol{i}) = A(\boldsymbol{i}) + \chi \cdot B(\boldsymbol{i})$ for all $\boldsymbol{i}$.

By the definition of g in Figure 9 and since $\sum_{\boldsymbol{i} \in \{0,1\}^{\ell^*}} h(\boldsymbol{i}) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\rho}, \boldsymbol{i}) = \tilde{h}(\boldsymbol{\rho})$ for the MLE $\tilde{h}$ of any $h : \{0,1\}^{\ell^*} \to \mathbb{F}$, we have

$$\sum_{\boldsymbol{i} \in \{0,1\}^{\ell^*}} \mu^*(\boldsymbol{i}) = \widetilde{G}(\boldsymbol{\rho}).$$

By the assumption that GP does not occur, there is an $\boldsymbol{i}^*$ such that $A(\boldsymbol{i}^*) \neq 0$ or $B(\boldsymbol{i}^*) \neq 0$. Hence, $A(\boldsymbol{i}^*) + X \cdot B(\boldsymbol{i}^*)$ is a non-zero polynomial in X of degree at most 1, and thus $A(\boldsymbol{i}^*) + \chi \cdot B(\boldsymbol{i}^*) \neq 0$ except with probability $\leq 1/p$ over $\chi \leftarrow\!\!\$\ \mathbb{F}$. Therefore, $\widetilde{G}$ is a non-zero multilinear polynomial in ℓ^* variables, and by Lemma 1 $\widetilde{G}(\boldsymbol{\rho}) = 0$ with probability at most ℓ^*/p over $\boldsymbol{\rho} \leftarrow\!\!\$\ \mathbb{F}^{\ell^*}$. Since $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1$ implies in particular $((0, [\![\mu^*]\!]); \mu^*) \in \mathcal{R}_{sum}$, i.e., $\widetilde{G}(\boldsymbol{\rho}) = 0$, in this case of $\alpha^* \in \{\mathsf{WS}_\mu, \mathsf{RS}\}$ we can bound $\Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1] \leq (\ell^* + 1)/p$. In the symmetric case of $\alpha^* \in \{\mathsf{WS}_t, \mathsf{S}\}$ we instead can bound $\Pr[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_2) = 1] \leq (b^* + 1)/p$. This proves the bound of the claim. □

Finally, we analyze the transition from tr_2 to tr_{2+k}.

Claim. For every $k = 0, \ldots, \mathsf{r}_{sum} - 1$, if $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k}) = 0$, then for any prover's message π of the $(k+1)$-th round of step 5,

$$\Pr_{\rho}[\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k+1}) = 1] \leq \frac{3}{p},$$

where $\mathsf{tr}_{2+k+1} = \mathsf{tr}_2|\mathsf{tr}''_k|\pi|\rho$ and the probability is over the $(k+1)$-th verifier's challenge ρ of step 5.

Proof. Since $k+1 \geq 1$, by definition $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k+1}) = 1$ iff V_6 and $\mathsf{state}_{sum}(\mathsf{tr}''_k|\pi|\rho) = 1$. As observed above, V_6 is determined by tr_2, hence it is fixed and independent of ρ.

If V_6 does not occur, then $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k+1}) = 1$ never occurs and the probability of the claim is 0.

If V_6 occurs, then $\mathsf{state}(\boldsymbol{t}_\beta, [\![\tilde{\mu}]\!], \mathsf{tr}_{2+k}) = 0$ implies $\mathsf{state}'_{sum}(0, ([\![\mu^*]\!], [\![t^*]\!]), \mathsf{tr}''_k) = 0$. For $k \geq 1$ this is immediate from the definition of state on tr_{2+k}, whereas for $k = 0$ the definition of state on tr_2 gives that at least one of $((0, [\![\mu^*]\!]); \mu^*), ((0, [\![t^*]\!]); t^*)$ is not in $\mathcal{R}_{sum}$, which implies $\mathsf{state}'_{sum}(0, ([\![\mu^*]\!], [\![t^*]\!]), \mathsf{tr}''_0) = \mathsf{state}'_{sum}(0, ([\![\mu^*]\!], [\![t^*]\!]), \emptyset) = 0$ by the first property of RBR-soundness applied to the parallel execution of Π_{sum}. The claim then follows by the third property of RBR-soundness applied to the instance of Π_{sum} on which state_{sum} evaluates to 0, whose virtual oracle has degree 3 in each variable. □

The RBR-soundness error of $\hat{\Pi}^{\mathbb{F}}_{range}$ is the maximum of the three per-round bounds established by the claims above, namely

$$\max\left\{\frac{2^{\ell^*}+2^{b^*}}{p}; \frac{\max(\ell^*, b^*)+1}{p}; \frac{3}{p}\right\} = \frac{2^{\ell^*}+2^{b^*}}{p}.$$

□

D.5 Our Optimized PIOP $\hat{\Pi}^*_{range}$

In this section, we present a PIOP protocol, called $\hat{\Pi}^*_{range}$, for the relation $\mathcal{R}_{range}$ of equation (3), which is more optimized than the PIOP Π_{range} obtained by combining the PIOP Π_{range} of Section 4 (Figure 2) with the PIOPs $\hat{\Pi}_{range}$ and $\hat{\Pi}^{\mathbb{F}}_{range}$ described in the previous sections.

For this protocol, we reuse the same preliminaries and building blocks of Section D, except that we use the following bound instead of that of equation (24):

$$\left(\frac{\ell+\nu}{\sigma}\right)^{\kappa} \approx 2^{-\lambda} \tag{29}$$

We present the protocol in Figure 10.

The following theorem states the correctness and security of $\hat{\Pi}^*_{range}$.

Theorem 9. *Let $B = 2^b$, $N = 2^\nu$, $c = 2^\gamma$ be such that $c \mid b$ and $\beta = B^{1/c}$, and let T_B and $\boldsymbol{t}_\beta = \{0, \ldots, \beta-1\} \subset \mathbb{Z}$ be the tables of Lemma 4. Consider the parameters $q, p, \beta, \sigma, m, \kappa$ satisfying equations (23), (29) and (25) for $\ell^* = \ell + \nu + \gamma$. Let $\hat{\Pi}^{\mathbb{F}}_{range}$ be a PIOP for the relation $\hat{\mathcal{R}}_{range}$ over the finite field $\mathbb{F} = \mathbb{Z}_p$ that is perfectly complete and has RBR-knowledge error δ'_{range}. Then the $\hat{\Pi}^*_{range}$ protocol of Figure 10 is a PIOP for $\mathcal{R}_{range}$ with perfect completeness and RBR-knowledge error* $\max\left\{\left(\frac{\ell+\nu}{\sigma}\right)^{\kappa}; \frac{\ell+\nu+\gamma-m}{p}; \delta'_{range}\right\}$.

PIOP $\hat{\Pi}^*_{range}$ for $(T_B, [\![\tilde{v}]\!]; \tilde{v}) \in \mathcal{R}_{range}$

1. $\mathcal{P}$ computes $\tilde{u} = \mathsf{unpack}(\tilde{v}) \in R_q^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}]$ and $\tilde{\mu} \in \mathbb{Z}^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z}']$ as the decomposition of $\tilde{u}$ according to equation (4), computes $\tilde{\mu}' = \pi_p(\tilde{\mu}) \in \mathbb{Z}_p[\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z}']$ and sends $[\![\tilde{\mu}']\!]$.
   $\mathcal{V}$ samples $\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{\kappa-1} \leftarrow\!\!\$\ \Sigma^{\ell+\nu}$ and sends them to $\mathcal{P}$.
2. $\mathcal{P}$ sends $(\tilde{\psi}'_0, \ldots, \tilde{\psi}'_{\kappa-1})$, where, for $\eta \in [\kappa]$, $\boldsymbol{r}'_\eta = \pi_p(r_{\eta,1}, \ldots, r_{\eta,m})$ and:

$$\tilde{\psi}'_\eta(X_{m+1}, \ldots X_\ell, \boldsymbol{Z}, \boldsymbol{Z}') = \tilde{\mu}'(\boldsymbol{r}'_\eta, X_{m+1}, \ldots X_\ell, \boldsymbol{Z}, \boldsymbol{Z}')$$

   $\mathcal{V}$ samples $\boldsymbol{s}' \leftarrow\!\!\$\ \mathbb{F}_p^{\ell-m+\nu+\gamma}$.
   For all $\eta \in [\kappa]$, queries $[\![\tilde{\mu}']\!]$ on $(\boldsymbol{r}'_\eta, \boldsymbol{s}') \in \mathbb{F}_p^{\ell+\nu+\gamma}$ and the virtual oracle $[\![\tilde{u}]\!]$ on $\hat{\boldsymbol{r}}_\eta = \pi_q(\boldsymbol{r}_\eta) \in \mathbb{Z}_q^{\ell+\nu}$. Then $\mathcal{V}$ checks if:

$$\forall \eta \in [\kappa] : \tilde{\psi}'_\eta(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{s}') \in \mathbb{Z}_p$$

   and $\forall l \in [0, L], \eta \in [\kappa]$

$$\tilde{u}(\hat{\boldsymbol{r}}_\eta) \bmod p_l = \sum_{\substack{\boldsymbol{\delta} \in \{0,1\}^{\ell+\nu-m} \\ \boldsymbol{k} \in \{0,1\}^\gamma}} \iota(\tilde{\psi}'_\eta(\boldsymbol{\delta}, \boldsymbol{k})) \cdot \tilde{\mathsf{eq}}(\boldsymbol{\delta}; (r_{\eta,m+1}, \ldots, r_{\eta,\ell+\nu})) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{k})} \bmod p_l$$

3. $\mathcal{P}$ and $\mathcal{V}$ run a PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$ for $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]; \tilde{\mu}') \in \hat{\mathcal{R}}_{range}$, where $\boldsymbol{t}'_\beta = \pi_p(\boldsymbol{t}_\beta)$.

Figure 10: Construction of the PIOP $\hat{\Pi}^*_{range}$ from $\hat{\Pi}^{\mathbb{F}}_{range}$ over $\mathbb{Z}_p$

When instantiated with the PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$ from Section D.4, p should additionally be $> 2^{\ell+\nu+\gamma}$.

Throughout the two proofs below we write $\boldsymbol{r}_\eta = (\boldsymbol{r}_\eta^{\leq m}, \boldsymbol{r}_\eta^{>m})$, where $\boldsymbol{r}_\eta^{\leq m} = (r_{\eta,1}, \ldots, r_{\eta,m})$ and $\boldsymbol{r}_\eta^{>m} = (r_{\eta,m+1}, \ldots, r_{\eta,\ell+\nu})$, so that $\boldsymbol{r}'_\eta = \pi_p(\boldsymbol{r}_\eta^{\leq m})$ and $\hat{\boldsymbol{r}}_\eta = \pi_q(\boldsymbol{r}_\eta)$.

Correctness of $\hat{\Pi}^*_{range}$. Let $\tilde{v}$ be a polynomial such that $\forall \boldsymbol{b} \in \{0,1\}^\ell$ it holds $\tilde{v}(\boldsymbol{b}) \in T_B$. By Lemma 4, the polynomial $\tilde{\mu} \in \mathbb{Z}^{(\leq 1)}[\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z}']$ computed by $\mathcal{P}$ in step 1 satisfies equation (4) and $\tilde{\mu}(\boldsymbol{b}) \in \boldsymbol{t}_\beta$ for all $\boldsymbol{b} \in \{0,1\}^{\ell+\nu+\gamma}$. By construction, for $\tilde{\mu}' = \pi_p(\tilde{\mu})$ it holds $\tilde{\mu}'(\boldsymbol{b}) \in \boldsymbol{t}'_\beta$ $\forall \boldsymbol{b} \in \{0,1\}^{\ell+\nu+\gamma}$ since

$$\tilde{\mu}'(\boldsymbol{b}) = \pi_p(\tilde{\mu})(\boldsymbol{b}) = \pi_p(\tilde{\mu}(\boldsymbol{b})) \in \pi_p(\boldsymbol{t}_\beta) = \boldsymbol{t}'_\beta.$$

Therefore, by its correctness, the protocol $\hat{\Pi}^{\mathbb{F}}_{range}$ accepts.

Next, we have to show that the verifier's checks of step 2 are satisfied.

It is straightforward to see that the checks $\tilde{\psi}'_\eta(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{s}')$ are satisfied by construction.

Let us now focus on the last check. By construction of $\tilde{\psi}'_\eta$ we have $\tilde{\psi}'_\eta(\boldsymbol{\delta}, \boldsymbol{k}) = \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{\delta}, \boldsymbol{k})$ and, by applying Proposition 1 to $\tilde{\mu}$ on the point $(\boldsymbol{r}_\eta^{\leq m}, \boldsymbol{\delta}, \boldsymbol{k}) \in \mathbb{Z}^{\ell+\nu+\gamma}$ along with the bound on the value of $\tilde{\mu}(\boldsymbol{r}_\eta^{\leq m}, \boldsymbol{\delta}, \boldsymbol{k})$, which we claim later,

$$\iota(\tilde{\psi}'_\eta(\boldsymbol{\delta}, \boldsymbol{k})) = \iota\left(\pi_p(\tilde{\mu})\left(\pi_p(\boldsymbol{r}_\eta^{\leq m}, \boldsymbol{\delta}, \boldsymbol{k})\right)\right) = \tilde{\mu}(\boldsymbol{r}_\eta^{\leq m}, \boldsymbol{\delta}, \boldsymbol{k}).$$

Therefore, by the multilinearity of $\tilde{\mu}$, for the right-hand side of the last check we obtain the following equality over the integers:

$$\sum_{\substack{\boldsymbol{\delta}\in\{0,1\}^{\ell+\nu-m}\\ \boldsymbol{k}\in\{0,1\}^{\gamma}}} \iota(\tilde{\psi}'_{\eta}(\boldsymbol{\delta},\boldsymbol{k}))\cdot \tilde{\mathsf{eq}}(\boldsymbol{\delta};\boldsymbol{r}_{\eta}^{>m})\cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{k})}$$
$$= \sum_{\boldsymbol{k}\in\{0,1\}^{\gamma}} \beta^{\mathsf{to\text{-}int}(\boldsymbol{k})} \sum_{\boldsymbol{\delta}\in\{0,1\}^{\ell+\nu-m}} \tilde{\mu}(\boldsymbol{r}_{\eta}^{\leq m},\boldsymbol{\delta},\boldsymbol{k})\cdot \tilde{\mathsf{eq}}(\boldsymbol{\delta};\boldsymbol{r}_{\eta}^{>m}) = \sum_{\boldsymbol{k}\in\{0,1\}^{\gamma}} \tilde{\mu}(\boldsymbol{r}_{\eta},\boldsymbol{k})\cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{k})}.$$

Finally, both sides of equation (4) are multilinear in $(\boldsymbol{X},\boldsymbol{Z})$ and agree on $\{0,1\}^{\ell+\nu}$, so that equation, once reduced modulo q, holds as an identity of polynomials, i.e. $\pi_q\left(\sum_{\boldsymbol{k}\in\{0,1\}^{\gamma}} \tilde{\mu}(\boldsymbol{X},\boldsymbol{Z},\boldsymbol{k})\cdot\beta^{\mathsf{to\text{-}int}(\boldsymbol{k})}\right) = \tilde{u}(\boldsymbol{X},\boldsymbol{Z})$. Since π_q is a ring homomorphism, evaluating on $\boldsymbol{r}_{\eta}$ gives $\tilde{u}(\hat{\boldsymbol{r}}_{\eta}) = \pi_q\left(\sum_{\boldsymbol{k}\in\{0,1\}^{\gamma}} \tilde{\mu}(\boldsymbol{r}_{\eta},\boldsymbol{k})\cdot\beta^{\mathsf{to\text{-}int}(\boldsymbol{k})}\right)$, and thus the last check is satisfied for all $\eta\in[\kappa]$ and $l\in[0,L]$.

Claim. If $\tilde{\mu}(\boldsymbol{b})\in\boldsymbol{t}_{\beta}$ for all $\boldsymbol{b}\in\{0,1\}^{\ell+\nu+\gamma}$ then, for all $\eta\in[\kappa]$, $\boldsymbol{\delta}\in\{0,1\}^{\ell+\nu-m}$ and $\boldsymbol{k}\in\{0,1\}^{\gamma}$, $\left|\tilde{\mu}(\boldsymbol{r}_{\eta}^{\leq m},\boldsymbol{\delta},\boldsymbol{k})\right| < 2^m\beta\sigma^m < p/2$.

Proof. By the multi-linearity of the polynomial we have that

$$\tilde{\mu}(\boldsymbol{r}_{\eta}^{\leq m},\boldsymbol{\delta},\boldsymbol{k}) = \sum_{\boldsymbol{\phi}\in\{0,1\}^{m}} \tilde{\mu}(\boldsymbol{\phi},\boldsymbol{\delta},\boldsymbol{k})\,\tilde{\rho}_{\eta}(\boldsymbol{\phi})$$

where $\tilde{\rho}_{\eta}(\boldsymbol{\phi}) = \tilde{\mathsf{eq}}(\boldsymbol{\phi};\boldsymbol{r}_{\eta}^{\leq m}) = \prod_{i=1}^{m}(\phi_i r_{\eta,i} + (1-\phi_i)(1-r_{\eta,i}))$. Given that $0\leq r_{\eta,i}<\sigma$, we have that $-\sigma^m < \tilde{\rho}_{\eta}(\boldsymbol{\phi}) < \sigma^m$; thus by combining it with the assumption that $\tilde{\mu}(\boldsymbol{\phi},\boldsymbol{\delta},\boldsymbol{k})\in\boldsymbol{t}_{\beta}$, i.e. $0\leq\tilde{\mu}(\boldsymbol{\phi},\boldsymbol{\delta},\boldsymbol{k})\leq\beta-1$, we obtain $\left|\tilde{\mu}(\boldsymbol{r}_{\eta}^{\leq m},\boldsymbol{\delta},\boldsymbol{k})\right| < 2^m\beta\sigma^m$, which is smaller than $p/2$ by equation (23). □

This concludes the correctness proof.

Security of $\hat{\Pi}^{*}_{range}$. We prove the RBR-soundness, from which we get RBR-knowledge by then applying Lemma 6.

As in Section D.2, let $\tilde{\mu}''$ be the multilinear polynomial in $\mathbb{Z}^{(\leq 1)}[\boldsymbol{X},\boldsymbol{Z},\boldsymbol{Z}']$ such that $\tilde{\mu}''(\boldsymbol{b}) = \iota(\tilde{\mu}'(\boldsymbol{b}))$ for all $\boldsymbol{b}\in\{0,1\}^{\ell+\nu+\gamma}$; note that $\tilde{\mu}''=\tilde{\mu}$ when $\mathcal{P}$ is honest. We define the $\mathsf{state}(T_B,[\![\tilde{v}]\!],\mathsf{tr}_i)$ function on every (partial) transcript tr_i, from round $i=0$ to $i=2+\mathsf{r}_{range}$, where r_{range} denotes the number of rounds of the $\hat{\Pi}^{\mathbb{F}}_{range}$ protocol:

- For $\mathsf{tr}_0=\emptyset$, $\mathsf{state}(T_B,[\![\tilde{v}]\!],\mathsf{tr}_0)=0$ iff $(T_B,[\![\tilde{v}]\!])\notin\mathcal{L}_{\mathcal{R}_{range}}$.
- For $\mathsf{tr}_1=[\![\tilde{\mu}']\!](\boldsymbol{r}_0,\ldots,\boldsymbol{r}_{\kappa-1})$, $\mathsf{state}(T_B,[\![\tilde{v}]\!],\mathsf{tr}_1)=0$ iff $(\boldsymbol{t}'_{\beta},[\![\tilde{\mu}']\!])\notin\mathcal{L}_{\hat{\mathcal{R}}_{range}}$ or there exists $\eta\in[\kappa]$ such that $F(\hat{\boldsymbol{r}}_{\eta})\neq 0$, where $F(\boldsymbol{X},\boldsymbol{Z})\in\mathbb{Z}_q[\boldsymbol{X},\boldsymbol{Z}]$ is the multilinear polynomial

$$F(\boldsymbol{X},\boldsymbol{Z}) = \tilde{u}(\boldsymbol{X},\boldsymbol{Z}) - \pi_q\left(\sum_{\boldsymbol{k}\in\{0,1\}^{\gamma}} \tilde{\mu}''(\boldsymbol{X},\boldsymbol{Z},\boldsymbol{k})\cdot\beta^{\mathsf{to\text{-}int}(\boldsymbol{k})}\right).$$

- For $\mathsf{tr}_2 = \mathsf{tr}_1|(\tilde{\psi}'_0, \ldots, \tilde{\psi}'_{\kappa-1})|\boldsymbol{s}'$: $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 0$ iff
  - $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]) \notin \mathcal{L}_{\hat{\mathcal{R}}_{range}}$
  - or there exists $\eta \in [\kappa]$ such that $\neg \mathsf{V}_{1,\eta}$, or there exist $\eta \in [\kappa]$ and $l \in [0, L]$ such that $\neg \mathsf{V}_{2,\eta,l}$, where

$$\mathsf{V}_{1,\eta} := \left(\tilde{\psi}'_\eta(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_\eta, \boldsymbol{s}')\right)$$

$$\mathsf{V}_{2,\eta,l} := \left(\tilde{u}(\hat{\boldsymbol{r}}_\eta) \bmod p_l = \sum_{\substack{\boldsymbol{\delta} \in \{0,1\}^{\ell+\nu-m} \\ \boldsymbol{k} \in \{0,1\}^\gamma}} \iota(\tilde{\psi}'_\eta(\boldsymbol{\delta}, \boldsymbol{k})) \cdot \widetilde{\mathsf{eq}}(\boldsymbol{\delta}; \boldsymbol{r}_\eta^{>m}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{k})} \bmod p_l \right)$$

  are the two checks of step 2.
- For $\mathsf{tr}_{2+k} = \mathsf{tr}_2|\mathsf{tr}'_k$ where, for $1 \le k \le \mathsf{r}_{range}$, tr'_k is the partial transcript of $\hat{\Pi}^{\mathbb{F}}_{range}$ at the k-th round: $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_{2+k}) = 0$ iff $\mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_k) = 0$, or there exists $\eta \in [\kappa]$ such that $\neg \mathsf{V}_{1,\eta}$, or there exist $\eta \in [\kappa]$ and $l \in [0, L]$ such that $\neg \mathsf{V}_{2,\eta,l}$.

Let us now show that this state function satisfies the three properties of RBR-soundness.

First, it is easy to see that by construction, if $(T_B, [\![\tilde{v}]\!]) \notin \mathcal{L}_{\mathcal{R}_{range}}$ then $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0$.

Second, for any *complete* transcript $\mathsf{tr} = \mathsf{tr}_{2+\mathsf{r}_{range}} = \mathsf{tr}_2|\mathsf{tr}'_{\mathsf{r}_{range}}$, by construction we have that if $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}) = 0$, then either $\mathsf{state}'(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!], \mathsf{tr}'_{\mathsf{r}_{range}}) = 0$, which means that the verifier of $\hat{\Pi}^{\mathbb{F}}_{range}$ rejects by the second property of RBR-soundness (Definition 6) applied to $\hat{\Pi}^{\mathbb{F}}_{range}$, or at least one of the verifier's checks of step 2 fails. In any of these cases, $\hat{\Pi}^*_{range}.\mathcal{V}$ rejects.

Third, we show that if state rejects on a partial transcript of round i, for any prover's message of the round $i+1$, we have a concrete (negligible) bound on the probability that state accepts on the next partial transcript. We show this property round by round.

Claim. If $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0$, then for any prover's message $[\![\tilde{\mu}']\!]$ we have that

$$\Pr_{\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{\kappa-1} \leftarrow\$ \Sigma^{\ell+\nu}}[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1 = ([\![\tilde{\mu}']\!]|(\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{\kappa-1}))) = 1] \le \left(\frac{\ell+\nu}{\sigma}\right)^\kappa.$$

Proof. By definition, $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 1$ means that:
(i) $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]) \in \mathcal{L}_{\hat{\mathcal{R}}_{range}}$, i.e., $\tilde{\mu}'(\boldsymbol{b}) \in \boldsymbol{t}'_\beta$ for all $\boldsymbol{b} \in \{0,1\}^{\ell+\nu+\gamma}$;
(ii) for all $\eta \in [\kappa]$: $F(\hat{\boldsymbol{r}}_\eta) = 0$.

From (i) and the definition of $\tilde{\mu}''$ we have $\tilde{\mu}''(\boldsymbol{b}) = \iota(\tilde{\mu}'(\boldsymbol{b})) \in \iota(\boldsymbol{t}'_\beta)$; since $\boldsymbol{t}'_\beta = \pi_p(\boldsymbol{t}_\beta)$ and $\beta < p/2$ (from equation (23)), using ι's definition we have $\iota(\pi_p(\boldsymbol{t}_\beta)) = \boldsymbol{t}_\beta$, and thus $\tilde{\mu}''(\boldsymbol{b}) \in \boldsymbol{t}_\beta$ for all $\boldsymbol{b} \in \{0,1\}^{\ell+\nu+\gamma}$. In particular, the multilinear polynomial $\pi_q(\tilde{\mu}'') \in \mathbb{Z}_q^{(\le 1)}[\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z}']$ satisfies equation (5). On the other hand, $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0$ means $(T_B, [\![\tilde{v}]\!]) \notin \mathcal{L}_{\mathcal{R}_{range}}$, and therefore, by

Lemma 4, $\pi_q(\tilde{\mu}'')$ must violate equation (4), i.e., there exists $\boldsymbol{i}^* \in \{0,1\}^{\ell+\nu}$ such that $\tilde{u}(\boldsymbol{i}^*) \neq \pi_q\left(\sum_{\boldsymbol{k}\in\{0,1\}^\gamma} \tilde{\mu}''(\boldsymbol{i}^*, \boldsymbol{k}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{k})}\right)$, i.e. $F(\boldsymbol{i}^*) \neq 0$ and thus the polynomial $F(\boldsymbol{X}, \boldsymbol{Z})$ is nonzero.

Notice that F is determined by $\tilde{v}$ and by the oracle $[\![\tilde{\mu}']\!]$, hence it is fixed before $\mathcal{V}$ samples $\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{\kappa-1}$, and that these are sampled independently of each other. Therefore,

$$\begin{aligned}
&\Pr[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 1 \mid \mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_0) = 0] \\
&\leq \Pr\left[\bigwedge_{\eta\in[\kappa]} F(\pi_q(\boldsymbol{r}_\eta)) = 0 \;\middle|\; F(\boldsymbol{X}, \boldsymbol{Z}) \neq 0\right] \\
&= \prod_{\eta\in[\kappa]} \Pr_{\boldsymbol{r}_\eta \leftarrow\$ \Sigma^{\ell+\nu}} \left[F(\pi_q(\boldsymbol{r}_\eta)) = 0 \mid F(\boldsymbol{X}, \boldsymbol{Z}) \neq 0\right].
\end{aligned}$$

Since F is multilinear in $\ell + \nu$ variables, hence of total degree at most $\ell + \nu$, and since $\pi_q(\Sigma)$ is a subset of the exceptional set $\hat{S}_0$ of size σ, by generalized Schwartz-Zippel (Lemma 1) we get $\Pr_{\boldsymbol{r}_\eta \leftarrow\$ \Sigma^{\ell+\nu}}[F(\pi_q(\boldsymbol{r}_\eta)) = 0 \mid F(\boldsymbol{X}, \boldsymbol{Z}) \neq 0] \leq \frac{\ell+\nu}{\sigma}$, and the claimed bound follows by considering the probability that this occurs for κ independent samples; by equation (29) this bound is $\approx 2^{-\lambda}$. □

Claim. If $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 0$, then for any prover's message $(\tilde{\psi}'_0, \ldots, \tilde{\psi}'_{\kappa-1})$ we have that

$$\Pr_{\boldsymbol{s}'}[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2 = (\mathsf{tr}_1 | (\tilde{\psi}'_0, \ldots, \tilde{\psi}'_{\kappa-1}) | \boldsymbol{s}')) = 1] \leq \frac{\ell - m + \nu + \gamma}{p}.$$

Proof. Let $\mathcal{E}_2$ be the event "$\bigwedge_{\eta\in[\kappa]}\left(\tilde{\psi}'_\eta(X_{m+1}, \ldots, X_\ell, \boldsymbol{Z}, \boldsymbol{Z}') = \tilde{\mu}'(\boldsymbol{r}'_\eta, X_{m+1}, \ldots, X_\ell, \boldsymbol{Z}, \boldsymbol{Z}')\right)$", i.e. the event that the polynomials sent by $\mathcal{P}$ are the correct partial evaluations of $\tilde{\mu}'$. Notice that $\mathcal{E}_2$ is determined by the prover's message, hence it is fixed and independent of $\boldsymbol{s}'$; the same holds for the event $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]) \in \mathcal{L}_{\hat{\mathcal{R}}_{range}}$, which is determined by the oracle sent in step 1. We distinguish the two cases.

If $\mathcal{E}_2$ does not occur, let $\bar{\eta}$ be an index for which the two polynomials differ. Since $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 1$ requires in particular $\mathsf{V}_{1,\bar{\eta}}$, we have

$$\begin{aligned}
&\Pr[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 1 \mid \bar{\mathcal{E}}_2, \mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 0] \\
&\leq \Pr\left[\tilde{\psi}'_{\bar{\eta}}(\boldsymbol{s}') = \tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \boldsymbol{s}') \;\middle|\; \tilde{\psi}'_{\bar{\eta}} \neq \tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \cdot)\right] \;\leq\; \frac{\ell - m + \nu + \gamma}{p},
\end{aligned}$$

where the last inequality follows by the Schwartz-Zippel lemma over $\mathbb{Z}_p$, applied to the non-zero multilinear polynomial $\tilde{\psi}'_{\bar{\eta}} - \tilde{\mu}'(\boldsymbol{r}'_{\bar{\eta}}, \cdot)$ in $\ell - m + \nu + \gamma$ variables, of total degree at most $\ell - m + \nu + \gamma$, with the random $\boldsymbol{s}' \leftarrow\$ \mathbb{Z}_p^{\ell-m+\nu+\gamma}$.

If $\mathcal{E}_2$ occurs, we argue that $\Pr[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 1 \mid \mathcal{E}_2, \mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 0] = 0$. We may assume $(\boldsymbol{t}'_\beta, [\![\tilde{\mu}']\!]) \in \mathcal{L}_{\hat{\mathcal{R}}_{range}}$, as otherwise $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 0$ by definition. As in the proof of the previous claim, this gives $\tilde{\mu}''(\boldsymbol{b}) \in \boldsymbol{t}_\beta$ for

all $\boldsymbol{b} \in \{0,1\}^{\ell+\nu+\gamma}$, so that the claim of the correctness proof, applied to $\tilde{\mu}''$ in place of $\tilde{\mu}$, gives $\left|\tilde{\mu}''(\boldsymbol{r}_{\eta}^{\leq m}, \boldsymbol{\delta}, \boldsymbol{k})\right| < p/2$. Hence, by Proposition 1 and $\mathcal{E}_2$,

$$\iota(\tilde{\psi}'_{\eta}(\boldsymbol{\delta}, \boldsymbol{k})) = \iota\left(\tilde{\mu}'(\boldsymbol{r}'_{\eta}, \boldsymbol{\delta}, \boldsymbol{k})\right) = \tilde{\mu}''(\boldsymbol{r}_{\eta}^{\leq m}, \boldsymbol{\delta}, \boldsymbol{k}) \qquad \forall \eta \in [\kappa], \boldsymbol{\delta} \in \{0,1\}^{\ell+\nu-m}, \boldsymbol{k} \in \{0,1\}^{\gamma}.$$

By the same computation as in the correctness proof — which only uses the multilinearity of $\tilde{\mu}''$ — the right-hand side of $\mathsf{V}_{2,\eta,l}$ equals $\pi_q\left(\sum_{\boldsymbol{k}\in\{0,1\}^{\gamma}} \tilde{\mu}''(\boldsymbol{r}_{\eta}, \boldsymbol{k}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{k})}\right) \bmod p_l$, and therefore

$$\bigwedge_{l\in[0,L]} \mathsf{V}_{2,\eta,l} \iff \tilde{u}(\hat{\boldsymbol{r}}_{\eta}) = \pi_q\left(\sum_{\boldsymbol{k}\in\{0,1\}^{\gamma}} \tilde{\mu}''(\boldsymbol{r}_{\eta}, \boldsymbol{k}) \cdot \beta^{\mathsf{to\text{-}int}(\boldsymbol{k})}\right) \iff F(\hat{\boldsymbol{r}}_{\eta}) = 0,$$

where the first equivalence holds since $p_0 \cdots p_L = q$ and thus, by the CRT, two elements of $\mathbb{Z}_q$ are equal if and only if they are equal modulo p_l for all $l \in [0, L]$. Since $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_1) = 0$ and we assumed $(\boldsymbol{t}'_{\beta}, [\![\tilde{\mu}']\!]) \in \mathcal{L}_{\hat{\mathcal{R}}_{range}}$, there exists $\bar{\eta} \in [\kappa]$ with $F(\hat{\boldsymbol{r}}_{\bar{\eta}}) \neq 0$, hence $\neg\mathsf{V}_{2,\bar{\eta},\bar{l}}$ for some $\bar{l} \in [0, L]$ and $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_2) = 0$. □

To prove the last case, recall that, once tr_2 is fixed, so is the event

$$E := \exists\, \eta \in [\kappa],\, l \in [0, L] : \neg\mathsf{V}_{1,\eta} \vee \neg\mathsf{V}_{2,\eta,l},$$

that is, the event that at least one of the verifier's checks of step 2 fails. We also let $\mathsf{tr}'_0 = \emptyset$.

Claim. For every $k = 0, \ldots, \mathsf{r}_{range} - 1$, if $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_{2+k}) = 0$, then for any prover's message π of the $(k+1)$-th round of $\hat{\Pi}^{\mathbb{F}}_{range}$ we have that

$$\Pr_{\rho}\left[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_{2+k+1}) = 1\right] \leq \delta'_{range}, \quad \text{where } \mathsf{tr}_{2+k+1} = \mathsf{tr}_2|\mathsf{tr}'_k|\pi|\rho,$$

and where the probability is over the $(k+1)$-th verifier's challenge ρ of $\hat{\Pi}^{\mathbb{F}}_{range}$.

Proof. Since $k + 1 \geq 1$, the definition of state on tr_{2+k+1} gives

$$\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_{2+k+1}) = 1 \iff \mathsf{state}'(\boldsymbol{t}'_{\beta}, [\![\tilde{\mu}']\!], \mathsf{tr}'_k|\pi|\rho) = 1 \wedge \bar{E}.$$

Moreover, E is determined by tr_2, hence it is fixed and independent of ρ. We distinguish two cases.

If E occurs, then $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_{2+k+1}) = 1$ never occurs and the probability of the claim is 0.

If E does not occur, then the event $\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_{2+k}) = 0$ of the precondition implies $\mathsf{state}'(\boldsymbol{t}'_{\beta}, [\![\tilde{\mu}']\!], \mathsf{tr}'_k) = 0$: for $k \geq 1$ this is immediate from the definition of state on tr_{2+k}, whereas for $k = 0$ the definition of state on tr_2 gives $(\boldsymbol{t}'_{\beta}, [\![\tilde{\mu}']\!]) \notin \mathcal{L}_{\hat{\mathcal{R}}_{range}}$, which in turn implies $\mathsf{state}'(\boldsymbol{t}'_{\beta}, [\![\tilde{\mu}']\!], \mathsf{tr}'_0) = 0$ by the first

property of RBR-soundness (Definition 6) applied to $\hat{\Pi}^{\mathbb{F}}_{range}$. Therefore, for all $0 \leq k < \mathsf{r}_{range}$,

$$\begin{aligned}
&\Pr_{\rho}[\mathsf{state}(T_B, [\![\tilde{v}]\!], \mathsf{tr}_{2+k+1}) = 1] \\
&= \Pr_{\rho}[\mathsf{state}'(\boldsymbol{t}'_{\beta}, [\![\tilde{\mu}']\!], \mathsf{tr}'_k|\pi|\rho) = 1 \wedge \bar{E}] \\
&\leq \Pr_{\rho}[\mathsf{state}'(\boldsymbol{t}'_{\beta}, [\![\tilde{\mu}']\!], \mathsf{tr}'_k|\pi|\rho) = 1] \ \leq \ \delta'_{range},
\end{aligned}$$

where the last inequality follows by the third property of RBR-soundness applied to $\hat{\Pi}^{\mathbb{F}}_{range}$ and to the partial transcript tr'_k, on which state' evaluates to 0. □

The final RBR-soundness error is the max of the three per-round bounds, namely $\max\left\{\left(\frac{\ell+\nu}{\sigma}\right)^{\kappa}; \frac{\ell-m+\nu+\gamma}{p}; \delta'_{range}\right\}$, where the first term is $\approx 2^{-\lambda}$ by equation (29).

Efficiency of $\hat{\Pi}^{*}_{range}$ In this section, we detail the efficiency of the PIOP $\hat{\Pi}^{*}_{range}$ when instantiated with the PIOP $\hat{\Pi}^{\mathbb{F}}_{range}$ (instantiated with $\ell^* = \ell+\nu+\gamma$) described in the previous section.

The PIOP has the following efficiency profile:

- the number of rounds is $\mathsf{r} = \max(\ell^*, b/c) + 4$, as $\hat{\Pi}^{\mathbb{F}}_{range}$ involves two rounds plus the parallel execution of two sum-checks of size 2^{ℓ^*} and $2^{b/c}$, and then there are two rounds in $\hat{\Pi}^{*}_{range}$ itself;
- the prover sends $\mathsf{o} = 7$ oracles that are multilinear polynomials over $\mathbb{Z}_p$: 3 of size $2^{b/c}$ and 4 of size 2^{ℓ^*}; in addition, the prover sends the polynomials $(\tilde{\psi}'_0, \ldots, \tilde{\psi}'_{\kappa-1})$ that are $\kappa \cdot 2^{\ell^*-m}$ elements of $\mathbb{Z}_p$;
- the proving time is $\mathsf{T}_{\mathcal{P}} = O(\kappa \cdot 2^{\ell^*} + 2^{b/c})$ since this is the time needed to build the 7 oracle polynomials and since the two sum-checks can be run in linear time as they involve a product of multilinear polynomials, plus considering the cost of computing the κ partial evaluations of $\tilde{\mu}'$;
- the verifier time is $\mathsf{T}_{\mathcal{V}} = O(\ell^* + b/c + \kappa \cdot 2^{\ell^*-m})$ since $O(\ell^* + b/c)$ is the verification time of the two sum-check executions and since $O(\kappa \cdot 2^{\ell^*-m})$ is the time to perform the last verification checks in the round 2 of $\hat{\Pi}^{*}_{range}$; we recall that this term is $\mathsf{poly}(\ell + \nu + \gamma)$ by equation (25). Additionally, as we show below $\mathcal{V}$ has to compute the evaluation of $\tilde{t}'_{\beta}$ on $(\boldsymbol{r}^*_t, 0), (\boldsymbol{r}^*_t, 1)$, which can be done in time $O(b/c)$ since $\tilde{t}'_{\beta}(X_1, \ldots, X_{b/c}) = \sum_{k=1}^{b/c} X_k \cdot 2^{k-1}$.
- the number of verifier's queries is $\mathsf{q} = 15 + 2\kappa$, as justified below.

Assume that in $\hat{\Pi}^{\mathbb{F}}_{range}$ the sum-check on μ^* (resp. t^*) ends with a query of $\mathcal{V}$ of the virtual oracle on $\boldsymbol{r}_{\mu}$ (resp. $\boldsymbol{r}_t$). Below, for a vector $\boldsymbol{x}$ we denote $\boldsymbol{x}[1, \ldots, m]$ its prefix of length m, and for $\boldsymbol{x} = (x_1, \ldots, x_m)$, we denote by $\boldsymbol{x}^*$ the vector $(x_2, \ldots, x_m)$.

The protocol requires the following queries to the 7 oracles over $\mathbb{Z}_p$:

- $[\![\tilde{\mu}']\!]$ on $(\boldsymbol{r}'_0, \boldsymbol{s}'), \ldots, (\boldsymbol{r}'_{\kappa-1}, \boldsymbol{s}')$, $(\boldsymbol{r}^*_{\mu}, 0), (\boldsymbol{r}^*_{\mu}, 1)$;

- $[\![\mathsf{read_ts}]\!]$ on $(\boldsymbol{r}^*_\mu, 0), (\boldsymbol{r}^*_\mu, 1)$;
- $[\![\mathsf{final_cts}]\!]$ on $(\boldsymbol{r}^*_t, 0), (\boldsymbol{r}^*_t, 1)$;
- $[\![f^{(1)}_{\mathsf{WS}_\mu}]\!]$ on $\boldsymbol{r}_\mu, (\boldsymbol{r}^*_\mu, 0), (\boldsymbol{r}^*_\mu, 1), (\mathbf{1}, 0)$
- $[\![f^{(1)}_{\mathsf{RS}}]\!]$ on $\boldsymbol{r}_\mu, (\boldsymbol{r}^*_\mu, 0), (\boldsymbol{r}^*_\mu, 1), (\mathbf{1}, 0)$
- $[\![f^{(1)}_{\mathsf{WS}_t}]\!]$ on $\boldsymbol{r}_t, (\boldsymbol{r}^*_t, 0), (\boldsymbol{r}^*_t, 1), (\mathbf{1}, 0)$
- $[\![f^{(1)}_{\mathsf{S}}]\!]$ on $\boldsymbol{r}_t, (\boldsymbol{r}^*_t, 0), (\boldsymbol{r}^*_t, 1), (\mathbf{1}, 0)$

In addition, in the protocol $\hat{\Pi}^*_{range}$, the verifier queries $[\![\mathsf{unpack}(\tilde{v})]\!]$ on $\hat{\boldsymbol{r}}_1, \ldots, \hat{\boldsymbol{r}}_\kappa$, which turn into querying the packed polynomial $[\![\tilde{v}]\!]$ on $\hat{\boldsymbol{r}}_1[1, \ldots, \ell], \ldots, \hat{\boldsymbol{r}}_\kappa[1, \ldots, \ell]$.

To minimize the number of evaluation proofs, it is possible to apply a simple batching optimization to all those evaluations of the form $(\boldsymbol{r}^*, 0), (\boldsymbol{r}^*, 1)$. In this case, the prover first sends the results y and y' of the evaluations in $(\boldsymbol{r}^*, 0)$ and $(\boldsymbol{r}^*, 1)$ respectively, the verifier chooses a random point θ, and queries the polynomial on $(\boldsymbol{r}^*, \theta)$. Then, assuming the result of this query is y'', the verifier checks that $y'' = (1-\theta)y + \theta y'$.

By using this optimization, we obtain in total $\mathsf{q} = 15 + 2\kappa$ queries, as mentioned earlier.

Instantiating the concrete parameters. By equations (23) and (29), the parameters must satisfy

$$\log p > 1 + \log \beta + m \cdot (1 + \log \sigma) \qquad \text{and} \qquad \kappa \geq \frac{\lambda}{\log \sigma - \log(\ell + \nu)},$$

while the dominant term of the proof size is $\kappa \cdot 2^{\ell+\nu+\gamma-m}$ elements of $\mathbb{Z}_p$. Differently from $\hat{\Pi}_{range}$, where the challenges $\boldsymbol{r}_\eta$ are used to fix $m \leq \ell^*$ variables of $\tilde{\mu}$, here they are sampled in $\Sigma^{\ell+\nu}$ and thus m can be taken as large as $\ell + \nu$. This is what makes the proof size independent of N: for $m \leq \ell$ we would have $2^{\ell+\nu+\gamma-m} \geq 2^{\nu+\gamma} = c \cdot N$, i.e., a proof size linear in N, whereas taking $m > \ell$ removes this factor.

Let us consider the parameters used in the range proofs for the largest circuit in our experiments, that is $\ell = 9$, $\beta = 2^{11}$, $\gamma = 2$ and $N = 2^{13}$ (hence $\nu = 13$ and $\ell + \nu = 22$), and let us target $\lambda = 128$ and a prime p of 208 bits. Then we can take for example $m = 15$, $\sigma = 2^{11}$, $\kappa = 20$, which gives

$$2\beta \cdot (2\sigma)^m = 2^{192} < p, \qquad \left(\frac{\ell+\nu}{\sigma}\right)^\kappa = \left(\frac{22}{2^{11}}\right)^{20} \leq 2^{-130},$$

and a proof size of $\kappa \cdot 2^{\ell+\nu+\gamma-m} = 20 \cdot 2^9 = 10240$ elements of $\mathbb{Z}_p$, i.e., about 260 KB. This is the choice we use in our experiments (Table 3).

Alternatively, by lowering σ we can push m up to its maximum $\ell + \nu$ at the price of more repetitions. Taking $m = \ell + \nu = 22$, $\sigma = 2^7$, $\kappa = 51$, we get

$$2\beta \cdot (2\sigma)^m = 2^{188} < p, \qquad \left(\frac{\ell+\nu}{\sigma}\right)^\kappa = \left(\frac{22}{2^7}\right)^{51} \leq 2^{-129},$$

and a proof size of $51 \cdot 2^2 = 204$ elements of $\mathbb{Z}_p$, i.e., about 5 KB. The price of this second choice is a larger κ, which affects the prover's running time, that grows as $O(\kappa \cdot 2^{\ell+\nu+\gamma})$, and the batch size of the two evaluation proofs.

E Additional material for Section 5

E.1 The copycon wiring predicate in Equation (7)

Throughout this subsection $\mathbf{z}$ denotes the $\#(copy)$-bit index of the copy gate, i.e. the block $\mathtt{Z}_{[\#(copy)]}$ that Eq. (7) feeds to the predicate. Let $s := \max(\mu, \#(in))$, so that $m' = s + 1$ and $\mathbf{y} = (\mathbf{y}_{[s]}, \mathtt{Y}_s)$, where the most significant variable $\mathtt{Y}_s = \mathtt{Y}_{m'-1}$ is the one that g uses to choose between $\tilde{V}_{in}$ and $\tilde{V}_{3,quot}$ (see Section 5.1). $\mathtt{copycon}(\mathbf{t}, \mathbf{z}, \mathbf{x}, \mathbf{y})$ always evaluates to zero, except in the two following cases when it evaluates to 1:

$$\begin{cases} \Big((\mathtt{Y}_{m'-1} = 0) \wedge (\mathbf{x}_{[d,m)} = \mathbf{1}) \wedge (\mathbf{y}_{[\#(in),s)} = \mathbf{1}) \wedge (\mathbf{t} = (0,1)) \\ \wedge (\tilde{V}_{3,ptxt}(\mathbf{x}_{[d]}) \text{ multiplies } \tilde{V}_{in}(\mathbf{y}_{[\#(in)]}) \text{ in the } (\mathbf{z})\text{-th check}) \Big) \\ \text{or} \\ \Big((\mathtt{Y}_{m'-1} = 1) \wedge (\mathbf{x}_{[d,m)} = \mathbf{1}) \wedge (\mathbf{y}_{[\mu,s)} = \mathbf{1}) \wedge (\mathbf{t} = (0,1)) \\ \wedge (\tilde{V}_{3,ptxt}(\mathbf{x}_{[d]}) \text{ multiplies } \tilde{V}_{3,quot}(\mathbf{y}_{[\mu]}) \text{ in the } (\mathbf{z})\text{-th check}) \Big) \end{cases} \tag{30}$$

$\widetilde{\mathtt{copycon}}(\mathbf{t}, \mathbf{z}, \mathbf{x}, \mathbf{y})$ can be described as:

$\widetilde{\mathtt{copycon}}(\mathtt{T}, \mathtt{Z}, \mathtt{X}, \mathtt{Y}) = \widetilde{\mathtt{eq}}(\mathtt{T}, (0,1)) \cdot \widetilde{\mathtt{copycon}}'(\mathtt{Z}, \mathtt{X}, \mathtt{Y})$,

where $\widetilde{\mathtt{copycon}}'(\mathtt{Z}, \mathtt{X}, \mathtt{Y})$ can be described exactly as in Eq. (30) by removing the condition $\mathbf{t} = (0,1)$. Whenever a party needs to evaluate $\widetilde{\mathtt{copycon}}(\mathtt{T}, \mathtt{Z}, \mathtt{X}, \mathtt{Y})$ at a specific point, they can do it from the evaluation of $\widetilde{\mathtt{copycon}}'(\mathtt{Z}, \mathtt{X}, \mathtt{Y})$. In particular, the committed polynomial is $\widetilde{\mathtt{copycon}}'$, which has $\#(copy) + m + m'$ variables.

E.2 Wiring predicates from [CCC+25]

Let us provide the description of the wiring predicates introduced in [CCC+25].

$$\mathtt{rescon}_1(\mathbf{z}, \mathbf{x}) = \begin{cases} 1, & \text{if wire } \mathbf{x} \text{ holds the } \mathbf{z}\text{-th } c' \text{ value.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathtt{rescon}_3(\mathbf{z}, \mathbf{x}) = \begin{cases} -1, & \text{if wire } \mathbf{x} \text{ holds the } \mathbf{z}\text{-th remainder.} \\ -\mathsf{p}_l, & \text{if } \mathbf{x} \text{ belongs to the } l\text{-th HE layer and holds the } \mathbf{z}\text{-th quotient.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathtt{bdcon}_2(\mathbf{z}, \mathbf{x}) = \begin{cases} 1, & \text{if wire } \mathbf{x} \text{ holds the } \mathbf{z}\text{-th } d_2 \text{ value.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathtt{bdcon}_3(\mathbf{z}, \mathbf{x}) = \begin{cases} -\mathsf{PW}_{\omega_l}(1)[i], & \text{if } \mathbf{x} \text{ belongs to the } l\text{-th HE layer and holds} \\ & \text{the } i\text{-th decomposition of the } \mathbf{z}\text{-th } d_2 \text{ value.} \\ 0, & \text{otherwise.} \end{cases}$$

$$\texttt{evk}(\mathbf{z},\mathbf{x}) = \begin{cases} \mathsf{evk}_{l,b}[i], & \text{if wire } \mathbf{x} \text{ belongs to the } l\text{-th HE layer and} \\ & \text{holds the } i\text{-th decomposition of the } d_2 \text{ value} \\ & \text{corresponding to the } \mathbf{z}\text{-th } c'[b], \text{ where } b \in \{0,1\}. \\ 0, & \text{otherwise.} \end{cases}$$

The following predicate was used in [CCC+25] but originates from [LXZ21]

$$\texttt{Xmult}(\mathbf{z},\mathbf{x},\mathbf{y}) = \begin{cases} 1, & \text{if } \ V(\mathbf{x}) \cdot V(\mathbf{y}) \text{ is added to } V_2(\mathbf{z}). \\ 0, & \text{otherwise.} \end{cases} \tag{31}$$

In our work, we have introduced several changes so as to improve efficiency. First of all, we have added the metadata variables `T` that distinguish among the different kinds of gate. Second, we have drastically reduced the amount of `X` variables by converting them into `Z` variables. Notice how, in our case, $\texttt{rescon}_1(\mathbf{z},\mathbf{x})$ and $\texttt{bdcon}_2(\mathbf{z},\mathbf{x})$ have simplified practically to the point of disappearance. We elaborate on this next, in Appendix E.3. This conversion is not applied everywhere. The predicates $\texttt{rescon}_3, \texttt{bdcon}_3$ and `evk` above do not merely test $\mathbf{z}$ against $\mathbf{x}$: they read the HE layer l of the wire $\mathbf{x}$ in order to apply the level-dependent constants p_l, $\mathsf{PW}_{\omega_l}(1)[i]$ and $\mathsf{evk}_{l,b}[i]$. The δ variables encoding that layer must therefore stay in `X`. What our version changes is only that the layer is matched against $\mathtt{Z}_{[\delta]}$ and used to select the constant by a single fused factor $\widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{Z}_{[\delta]})\widetilde{\mathsf{eq}}(\mathsf{bin}(l), \mathtt{X}_{[\delta]})$, keeping the MLE multilinear in both blocks.

The evolution of the predicates in [CCC+25] to the ones we propose is implicit by the definitions above, Appendix E.3 and the detailed description of our MLEs in Section 5.1.

E.3 Too many gates in Equation (7)?

Notice how $\widetilde{\texttt{rescon}}_1(\mathtt{T}) = \widetilde{\mathsf{eq}}(\mathtt{T}, (0,0))$ does not include any `Z` variables and hence $\forall \mathbf{z} \in \{0,1\}^{\max(\mu, \#(copy))}$, $\widetilde{\texttt{rescon}}_1(0,0) \cdot \tilde{V}_1(\mathbf{z}_{[\mu]}) = \tilde{V}_1(\mathbf{z}_{[\mu]})$. This would seem to have the effect of having "too many $\widetilde{\texttt{rescon}}$ gates". There are two different cases to consider here, on which we elaborate next:

- First we have the excess gates that result from the difference between the actual number of $\widetilde{\texttt{rescon}}$ gates and the next power of two following that number, i.e. 2^μ (all possible values for $\mathbf{z}_{[\mu]}$). Let us label those exceeding gates as $\hat{E} \subseteq \{0,1\}^\mu$. Whereas an honest prover would set $\tilde{V}_1(\hat{\mathbf{z}}) = 0 \ \forall \hat{\mathbf{z}} \in \hat{E}$, a malicious prover could choose to put some non-zero values in there. The same discussion applies verbatim to $\widetilde{\texttt{copycon}}_{3,copy}(\mathtt{T}) \cdot \tilde{V}_{3,copy}(\mathtt{Z}_{[\#(copy)]})$ and to the exceeding copy gates in $\{0,1\}^{\#(copy)}$, since $\widetilde{\texttt{copycon}}_{3,copy}$ carries no `Z` either. Such malicious prover would then have two options: Either these values respect the constraints given in Equations (7), (11) and their dependencies, or they do not. In the latter case, the soundness of sumcheck will detect this and abort. In the former case, this would ultimately do no harm, since the $\widetilde{\texttt{Xmult}}$ predicates from Equation (12) are such that $\widetilde{\texttt{Xmult}}(\hat{\mathbf{z}},\mathbf{x},\mathbf{y}) = 0 \ \forall (\hat{\mathbf{z}},\mathbf{x},\mathbf{y}) \in (\hat{E} \times \{0,1\}^{s'} \times \{0,1\}^{s'})$. This is by the very definition of $\hat{E}$.

- Second, there are the potentially exceeding $\max(\mu, \#(copy)) - \mu$ variables in $\mathtt{Z}$. These extra values mean repeating the same check that many times, which is harmless both in terms of soundness and efficiency. More concretely, $\forall \mathbf{z}'_{extra} \in \{0,1\}^{\max(\mu,\#(copy))-\mu}$ the "$\widetilde{\mathtt{rescon}}$ part" (i.e. $\mathtt{T} = (0,0)$) of Equations (7) simplifies to

$$\begin{aligned}\tilde{V}_0(0,0,\mathbf{z}_{[\mu]},\mathbf{z}'_{extra}) &= \tilde{V}_1(\mathbf{z}_{[\mu]}) \\ &\quad + \sum_{\mathbf{x}\in\{0,1\}^m} \widetilde{\mathtt{rescon}}_3(0,0,\mathbf{z}_{[\delta]},\mathbf{x}) \cdot \tilde{V}_{3,rc_3}(\mathbf{x}_{[\delta]},\mathbf{z}_{[\delta,\mu)},x_\delta) \\ &= \tilde{V}_1(\mathbf{z}_{[\mu]}) + \sum_{b=0}^{1} \widetilde{\mathtt{rescon}}_3(0,0,\mathbf{z}_{[\delta]},\mathbf{x}^{(b)}) \cdot \tilde{V}_{3,rc_3}(\mathbf{z}_{[\mu]},b),\end{aligned}$$

where $\mathbf{x}^{(b)} := (\mathbf{z}_{[\delta]}, b, \mathbf{1})$. The second equality uses that, for boolean $\mathbf{z}$, the fused factor $\widetilde{\mathsf{eq}}(\mathsf{bin}(l),\mathbf{z}_{[\delta]})\widetilde{\mathsf{eq}}(\mathsf{bin}(l),\mathbf{x}_{[\delta]})$ inside $\widetilde{\mathtt{rescon}}_3$ leaves a single non-zero value of $\mathbf{x}_{[\delta]}$, namely $\mathbf{x}_{[\delta]} = \mathbf{z}_{[\delta]}$. Notice how the right hand side is independent of $\mathbf{z}'_{extra} \in \{0,1\}^{\max(\mu,\#(copy))-\mu}$.

E.4 Additional material for Section 5.2

Recombining into Equation (12) Here we continue with the explanation of how to regroup values coming from Equation (7) and Equation (11) so as to execute sumcheck on Equation (12). This essentially follows [CCC+25].

For a set S among $\{d_{01}, d_2\}$, define predicates $\mathtt{C}_{2,S}(\mathtt{Y},\mathtt{Z})$ as follows:

$$\mathtt{C}_{2,S}(\mathbf{y},\mathbf{z}) = \begin{cases} 1, & \text{if wire } \mathbf{y} \text{ in } V_{2,S} \text{ is the } \mathbf{z}\text{-th wire in } V_2. \\ 0, & \text{otherwise.}\end{cases}$$

$\mathcal{V}$ samples $s_{01}, s_2 \leftarrow\!\!\$\ \hat{S}_N$ so as to randomly combine $\mathcal{P}$'s claims. Now, by running the sum-check protocol on Equation (32), $\mathcal{P}$ ends by claiming an evaluation $\tilde{V}_2(\mathbf{r}_2)$ (for a random sum-check challenge $\mathbf{r}_2 \leftarrow\!\!\$\ \hat{S}_N^{\mu+1}$, since $\#(V_2) = \mu + 1$) which can be substituted into Equation (12) so as to advance towards the input layer.

$$\begin{aligned}&s_{01}\tilde{V}_{2,d_{01}}(\mathbf{r}_{d_{01}}) + s_2\tilde{V}_{2,d_2}(\mathbf{r}_2) = \\ &s_{01}\Big(\sum_{\mathbf{z}\in\{0,1\}^{\mu+1}} \widetilde{\mathtt{C}}_{2,d_{01}}(\mathbf{r}_{d_{01}},\mathbf{z})\tilde{V}_2(\mathbf{z})\Big) + s_2\Big(\sum_{\mathbf{z}\in\{0,1\}^{\mu+1}} \widetilde{\mathtt{C}}_{2,d_2}(\mathbf{r}_2,\mathbf{z})\tilde{V}_2(\mathbf{z})\Big) \\ &= \sum_{\mathbf{z}\in\{0,1\}^{\mu+1}} \tilde{V}_2(\mathbf{z})\Big(s_{01}\widetilde{\mathtt{C}}_{2,d_{01}}(\mathbf{r}_{d_{01}},\mathbf{z}) + s_2\widetilde{\mathtt{C}}_{2,d_2}(\mathbf{r}_2,\mathbf{z})\Big). \qquad (32)\end{aligned}$$

For simplicity, the most significant variable of $\tilde{V}_2$ chooses between $\tilde{V}_{2,d_{01}}$ and $\tilde{V}_{2,d_2}$, i.e.[11] $\tilde{V}_2(\mathbf{z}_{[\mu]},0) = \tilde{V}_{2,d_{01}}(\mathbf{z}_{[\mu]})$ and $\tilde{V}_2(\mathbf{z}_{[\mu-1]},1,1) = \tilde{V}_{2,d_2}(\mathbf{z}_{[\mu-1]})$. Let us

[11] It is arbitrary whether to have $\tilde{V}_2(\mathbf{z}_{[\mu-1]},1,1) = \tilde{V}_{2,d_2}(\mathbf{z}_{[\mu-1]})$ and $\tilde{V}_2(\mathbf{z}_{[\mu-1]},0,1) = 0$ or viceversa.

define the corresponding predicates. Notice that $\mathtt{X}$ is of length μ and $\mathtt{Y}$ is of length $\mu - 1$.

$$\widetilde{\mathtt{C}}_{2,d_{01}}(\mathtt{X}, \mathtt{Z}) = (1 - \mathtt{Z}_\mu) \cdot \widetilde{\mathtt{eq}}(\mathtt{X}, \mathtt{Z}_{[0]})$$
$$\widetilde{\mathtt{C}}_{2,d_2}(\mathtt{Y}, \mathtt{Z}) = \mathtt{Z}_\mu \cdot \mathtt{Z}_{\mu-1} \cdot \widetilde{\mathtt{eq}}(\mathtt{Y}, \mathtt{Z}_{[1]})$$

While we could have defined $\tilde{V}_2$ directly as we just have described and use it to replace calls to $\tilde{V}_{2,d_{01}}$ and $\tilde{V}_{2,d_2}$ in Equations (7) and (11), that would have made our protocol more expensive. Had we done so, we would have required a standard two-to-one reduction in Equation (12), which requires twice the amount of queries to the $\widetilde{\mathtt{Xmult}}$ predicates in there. Since those encode the "actual HE circuit" and we implement them using polynomial commitments, we would rather do the simpler reassignment from Equation (32).

E.5 Prover algorithms

Here we list the necessary algorithms for a linear time prover when executing sumcheck on algorithms, adapted from Libra [XZZ+19]. We take verbatim from there the following algorithms and their dependencies: `Sumcheck` (Algorithm 2), `SumcheckProduct` (Algorithm 3) and `Precompute` (given as a procedure within Algorithm 4). Notice that these algorithms represent polynomials as their evaluation at the set $\{0, 1, 2\}$, which has to be an exceptional set in R_q in order to establish an isomorphism with their coefficient form. This is always the case for our choice of parameters, but otherwise $\{0, 1, 2\}$ can be replaced with any other exceptional set of the same size.

In all our algorithms, we start challenging from the least to the most significant variables. This has positive efficiency impacts due to the sparsity provided by prefixes in the MLEs of the wiring predicates (which are placed at their most significant position). Some of our algorithms exploit that kind of sparsity, as well as the possibility of multiplying vectors of different length.

Algorithm 5 is one of the central pieces of our efficient proofs for $\mathcal{R}_{AC}$. It requires several subroutines, given in Algorithms 1 to 4. Throughout, we abbreviate the following partially evaluated factors:

$$\begin{aligned}
\widehat{V}_{pt}(\mathbf{x}) &:= \tilde{V}_{3,ptxt}(\mathbf{x}_{[d]}) \cdot g(\boldsymbol{r}_{\boldsymbol{Y}}), \\
\widehat{V}_{BD}(\mathbf{x}) &:= \tilde{V}_{3,BD}(\mathbf{x}_{[\delta]}, \boldsymbol{r}_{\boldsymbol{Z}_{[\delta,\mu-1)}}, \mathbf{x}_{[\delta,2\delta)}), \\
\widehat{V}_{rc}(\mathbf{x}) &:= \tilde{V}_{3,rc_3}(\mathbf{x}_{[\delta]}, \boldsymbol{r}_{\boldsymbol{Z}_{[\delta,\mu)}}, x_\delta),
\end{aligned} \tag{33}$$

whose look-up tables we write $\mathbf{T}^{\mathtt{X}}_{\widehat{V}_{pt}} := g(\boldsymbol{r}_{\boldsymbol{Y}}) \cdot \mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,ptxt}}$, $\mathbf{T}^{\mathtt{X}}_{\widehat{V}_{BD}} := \mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,BD}}$ and $\mathbf{T}^{\mathtt{X}}_{\widehat{V}_{rc}} := \mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,rc_3}}$. Moreover, inside Algorithms 4 and 5, once $\mathtt{T}, \mathtt{Z}$ and $\mathtt{Y}$ are all fixed to their challenges, we write $\widehat{\mathtt{copycon}}(\mathbf{x}), \widehat{\mathtt{bdcon}}_3(\mathbf{x})$ and $\widehat{\mathtt{rescon}}_3(\mathbf{x})$ for the resulting functions of $\mathbf{x}$ alone (i.e. $\widehat{\mathtt{bdcon}}_3(\mathbf{x}) := \widetilde{\mathtt{bdcon}}_3(\boldsymbol{r}_{\boldsymbol{T}}, \boldsymbol{r}_{\boldsymbol{Z}_{[\delta]}}, \mathbf{x})$, $\widehat{\mathtt{rescon}}_3(\mathbf{x}) := \widetilde{\mathtt{rescon}}_3(\boldsymbol{r}_{\boldsymbol{T}}, \boldsymbol{r}_{\boldsymbol{Z}_{[\delta]}}, \mathbf{x})$, $\widehat{\mathtt{copycon}}(\mathbf{x}) := \widetilde{\mathtt{copycon}}(\boldsymbol{r}_{\boldsymbol{T}}, \boldsymbol{r}_{\boldsymbol{Z}_{[\#(\boldsymbol{copy})]}}, \mathbf{x}, \boldsymbol{r}_{\boldsymbol{Y}})$). The rounds that challenge the $\mathtt{X}$ variables are collected in Algorithm 4, which

Algorithm 5 calls once its look-up tables are in place. To simplify the writing, Algorithm 5's description we use the `SumcheckProduct` algorithm in lines 5 and 6. In practice and in our implementation, those are replaced with a more efficient `SparseSumcheckProduct` version which appears in Algorithm 9.

Algorithm 1: Algorithm $\texttt{Init_copy_Y}(\widetilde{\texttt{copycon}}, \mathbf{T}^{\mathtt{X}}_{V_{3,ptxt}}, \boldsymbol{r_T}, \boldsymbol{r_Z})$

Input : Sparse multi-linear $\widetilde{\texttt{copycon}}(\mathtt{T},\mathtt{Z},\mathtt{X},\mathtt{Y})$. Multi-linear $V_{3,ptxt}(\mathtt{X}_{[d]})$ and its initial look-up table. Random challenges $\boldsymbol{r_T} \in \hat{S}_N^2, \boldsymbol{r_Z} \in \hat{S}_N^{\#(copy)}$.

Output: Look-up table $\mathbf{T}^{\mathtt{Y}}_{f_1}$.

1 $\mathbf{T}_{\boldsymbol{r_T}} \leftarrow \texttt{Precompute}(\boldsymbol{r_T})$.

2 $\mathbf{T}_{\boldsymbol{r_Z}} \leftarrow \texttt{Precompute}(\boldsymbol{r_Z})$.

3 $\forall \mathbf{y} \in \{0,1\}^{m'}$, set $\mathbf{T}^{\mathtt{Y}}_{f_1}[\mathbf{y}] = 0$.

4 For every $(\mathbf{t},\mathbf{z},\mathbf{x},\mathbf{y}) \in \{0,1\}^{2+\#(copy)+m+m'}$ such that $\widetilde{\texttt{copycon}}(\mathbf{t},\mathbf{z},\mathbf{x},\mathbf{y}) \neq 0$, do:

5 $\mathbf{T}^{\mathtt{Y}}_{f_1}[\mathbf{y}] \leftarrow \mathbf{T}^{\mathtt{Y}}_{f_1}[\mathbf{y}] + \mathbf{T}_{\boldsymbol{r_T}}[\mathbf{t}] \cdot \mathbf{T}_{\boldsymbol{r_Z}}[\mathbf{z}] \cdot \widetilde{\texttt{copycon}}(\mathbf{t},\mathbf{z},\mathbf{x},\mathbf{y}) \cdot \mathbf{T}^{\mathtt{X}}_{V_{3,ptxt}}[\mathbf{x}_{[d]}]$.

6 **return** $\mathbf{T}^{\mathtt{Y}}_{f_1}$

Algorithm 2: Algorithm $\texttt{Init_copy_X}(\widetilde{\texttt{copycon}}, \boldsymbol{r_T}, \boldsymbol{r_Z}, \boldsymbol{r_Y})$

Input : Sparse multi-linear $\widetilde{\texttt{copycon}}(\mathtt{T},\mathtt{Z},\mathtt{X},\mathtt{Y})$. Random challenges $\boldsymbol{r_T} \in \hat{S}_N^2, \boldsymbol{r_Z} \in \hat{S}_N^{\#(copy)}, \boldsymbol{r_Y} \in \hat{S}_N^{m'}$.

Output: Look-up table $\mathbf{T}^{\mathtt{X}}_{f_3}$.

1 $\mathbf{T}_{\boldsymbol{r_T}}[\mathbf{t}] \leftarrow \texttt{Precompute}(\boldsymbol{r_T})$.

2 $\mathbf{T}_{\boldsymbol{r_Z}}[\mathbf{z}] \leftarrow \texttt{Precompute}(\boldsymbol{r_Z})$.

3 $\mathbf{T}_{\boldsymbol{r_Y}}[\mathbf{y}] \leftarrow \texttt{Precompute}(\boldsymbol{r_Y})$.

4 $\forall \mathbf{x} \in \{0,1\}^{m}$, set $\mathbf{T}^{\mathtt{X}}_{f_3}[\mathbf{x}] = 0$.

5 For every $(\mathbf{t},\mathbf{z},\mathbf{x},\mathbf{y}) \in \{0,1\}^{2+\#(copy)+m+m'}$ such that $\widetilde{\texttt{copycon}}(\mathbf{t},\mathbf{z},\mathbf{x},\mathbf{y}) \neq 0$, do:

6 $\mathbf{T}^{\mathtt{X}}_{f_3}[\mathbf{x}] \leftarrow \mathbf{T}^{\mathtt{X}}_{f_3}[\mathbf{x}] + \mathbf{T}_{\boldsymbol{r_T}}[\mathbf{t}] \cdot \mathbf{T}_{\boldsymbol{r_Z}}[\mathbf{z}] \cdot \mathbf{T}_{\boldsymbol{r_Y}}[\mathbf{y}] \cdot \widetilde{\texttt{copycon}}(\mathbf{t},\mathbf{z},\mathbf{x},\mathbf{y})$.

7 **return** $\mathbf{T}^{\mathtt{X}}_{f_3}$

We invoke `Init_X` also on the intermediate-value MLEs $\tilde{V}_{3,BD}$ and $\tilde{V}_{3,rc_3}$, which carry no `T` variables and for which only *part* of their arguments comes from `Z`, their HE-level block being read from $\mathtt{X}_{[\delta]}$. In every call we therefore pass exactly the block of challenges that the argument depends on: $\texttt{Init_X}(\tilde{V}_{3,BD}, \boldsymbol{r_{Z_{[\delta,\mu-1)}}})$ returns a table indexed by the 2δ variables $\mathtt{X}_{[2\delta]}$ (level and p_i), and $\texttt{Init_X}(\tilde{V}_{3,rc_3}, \boldsymbol{r_{Z_{[\delta,\mu)}}})$ one indexed by the $\delta+1$ variables $\mathtt{X}_{[\delta+1]}$ (level and the rmd-vs-quot bit). Likewise $\widetilde{\texttt{bdcon}}_3, \widetilde{\texttt{rescon}}_3$ only depend on $\mathtt{Z}_{[\delta]}$ and $\widetilde{\texttt{evk}}$ on $\mathtt{Z}_{[\delta+1]}$.

The tables $\mathbf{T}^{\mathtt{X}}_{V_{3,ptxt}}$, $\mathbf{T}^{\mathtt{Y}}_{g}$ and $\mathbf{T}^{\mathtt{Z}}_{\tilde{V}_2}$ are not produced by `Init_X`: the polynomials they tabulate depend neither on `T` nor on any block of `Z` that is fixed to a

Algorithm 3: Algorithm $\texttt{Init_X}(f, \boldsymbol{r_T}, \boldsymbol{r_Z})$

Input : Multi-linear $f(\mathtt{T}, \mathtt{Z}, \mathtt{X})$. Random challenges $\boldsymbol{r_T} \in \hat{S}_N^2$ and $\boldsymbol{r_Z}$, the block of $\mathtt{Z}$-challenges that f depends on.

Output: Look-up table $\mathbf{T}_f^{\mathtt{X}}$.

$\mathbf{T}_{\boldsymbol{r_T}}[\mathbf{t}] \leftarrow \texttt{Precompute}(\boldsymbol{r_T})$.

$\mathbf{T}_{\boldsymbol{r_Z}}[\mathbf{z}] \leftarrow \texttt{Precompute}(\boldsymbol{r_Z})$.

$\forall \mathbf{x} \in \{0,1\}^m$, set $\mathbf{T}_f^{\mathtt{X}}[\mathbf{x}] = 0$.

For every $(\mathbf{t}, \mathbf{z}, \mathbf{x})$ in the domain of f such that $f(\mathbf{t}, \mathbf{z}, \mathbf{x}) \neq 0$, do:

$\mathbf{T}_f^{\mathtt{X}}[\mathbf{x}] \leftarrow \mathbf{T}_f^{\mathtt{X}}[\mathbf{x}] + \mathbf{T}_{\boldsymbol{r_T}}[\mathbf{t}] \cdot \mathbf{T}_{\boldsymbol{r_Z}}[\mathbf{z}] \cdot f(\mathbf{t}, \mathbf{z}, \mathbf{x})$.

return $\mathbf{T}_f^{\mathtt{X}}$

challenge, so they are simply the initial bookkeeping tables of $\tilde{V}_{3,ptxt}$, g and $\tilde{V}_2$, which the prover assembles from values it already holds.

Algorithm 4: Algorithm $\texttt{Sumcheck_X}(\widehat{\texttt{copycon}}, \mathbf{T}_{f_3}^{\mathtt{X}}, \widehat{V}_{pt}, \mathbf{T}_{\tilde{V}_{3,ptxt}}^{\mathtt{X}} \cdot g(\boldsymbol{r_Y}), \widehat{\texttt{bdcon}}_3, \mathbf{T}_{\widetilde{\texttt{bdcon}}_3}^{\mathtt{X}}, \widehat{V}_{BD}, \mathbf{T}_{\tilde{V}_{3,BD}}^{\mathtt{X}}, \widehat{\texttt{rescon}}_3, \mathbf{T}_{\widetilde{\texttt{rescon}}_3}^{\mathtt{X}}, \widehat{V}_{rc}, \mathbf{T}_{\tilde{V}_{3,rc_3}}^{\mathtt{X}})$

Input : The look-up tables $\mathbf{T}_{f_3}^{\mathtt{X}}, \mathbf{T}_{\widetilde{\texttt{bdcon}}_3}^{\mathtt{X}}, \mathbf{T}_{\widetilde{\texttt{rescon}}_3}^{\mathtt{X}}$ of the three wiring predicates and $\mathbf{T}_{\tilde{V}_{3,ptxt}}^{\mathtt{X}} \cdot g(\boldsymbol{r_Y}), \mathbf{T}_{\tilde{V}_{3,BD}}^{\mathtt{X}}, \mathbf{T}_{\tilde{V}_{3,rc_3}}^{\mathtt{X}}$ of the three factors they are paired with, as built in Algorithm 5.

Output: $\mathbf{a}_{[m',m+m']}$, which are the prover sumcheck messages for the $\mathtt{X}$ variables in Equation (7)

Comment: The three products are added up inside a single *sumcheck, so their rounds must be interleaved rather than run one after the other: the message of round i is the sum of the three round-i contributions, and only once it has been sent is $r_{\mathtt{X}_i}$ available and the six look-up tables advanced by one variable. We write* $\texttt{SumcheckProduct}^{(i)}$ *for the i-th round of* $\texttt{SumcheckProduct}$*, that is, for the three elements $(a_{i,0}, a_{i,1}, a_{i,2})$ it outputs in that round. Interleaving at the level of rounds is also what lets each product keep its own number of variables for the second term, namely d for $\tilde{V}_{3,ptxt}$, 2δ for $\tilde{V}_{3,BD}$ and $\delta + 1$ for $\tilde{V}_{3,rc_3}$.*

for $i \leftarrow 0$ **to** $m - 1$ **do**

  The next three steps are executed in parallel:

  $\mathbf{a}'_{m'+i} \leftarrow \texttt{SumcheckProduct}^{(i)}(\widehat{\texttt{copycon}}(\mathbf{x}), \mathbf{T}_{f_3}^{\mathtt{X}}, \widehat{V}_{pt}(\mathbf{x}), \mathbf{T}_{\tilde{V}_{3,ptxt}}^{\mathtt{X}} \cdot g(\boldsymbol{r_Y}))$

  $\mathbf{a}''_{m'+i} \leftarrow \texttt{SumcheckProduct}^{(i)}(\widehat{\texttt{bdcon}}_3(\mathbf{x}), \mathbf{T}_{\widetilde{\texttt{bdcon}}_3}^{\mathtt{X}}, \widehat{V}_{BD}(\mathbf{x}), \mathbf{T}_{\tilde{V}_{3,BD}}^{\mathtt{X}})$

  $\mathbf{a}'''_{m'+i} \leftarrow \texttt{SumcheckProduct}^{(i)}(\widehat{\texttt{rescon}}_3(\mathbf{x}), \mathbf{T}_{\widetilde{\texttt{rescon}}_3}^{\mathtt{X}}, \widehat{V}_{rc}(\mathbf{x}), \mathbf{T}_{\tilde{V}_{3,rc_3}}^{\mathtt{X}})$

  $\mathbf{a}_{m'+i} \leftarrow \mathbf{a}'_{m'+i} + \mathbf{a}''_{m'+i} + \mathbf{a}'''_{m'+i}$

  Send $\mathbf{a}_{m'+i}$, receive $r_{\mathtt{X}_i} \in \hat{S}_N$ and advance the six look-up tables with it, as $\texttt{FunctionEvaluations}$ and $\texttt{SparseFuncEval}$ do.

end

return $\mathbf{a}_{[m',m+m']}$

Algorithm 5: `Sumcheck` Equation (7)

Output: $\mathbf{a}_{[m+m']}$, which are the prover sumcheck messages for Equation (7)

1 *Comment: We challenge first on the* $\mathtt{Y}$ *variables, then, in Algorithm 4, on the* $\mathtt{X}$ *ones. Notice the abuse of notation in lines 4 and 5, since* $\tilde{V}_{3,BD}$ *and* $\tilde{V}_{3,rc_3}$ *have no* $\mathtt{T}$ *variables and only their wire-indexing block comes from* $\mathtt{Z}$*. The abbreviations* $\widehat{V}_{BD}(\mathbf{x})$ *and* $\widehat{V}_{rc}(\mathbf{x})$ *are the ones introduced in Equation (33), and we drop the fixed arguments* $\boldsymbol{r_T}, \boldsymbol{r_{Z_{[\delta]}}}$ *from* $\widetilde{\mathtt{bdcon}}_3$ *and* $\widetilde{\mathtt{rescon}}_3$*.*

2 $\mathbf{T}^{\mathtt{X}}_{\widetilde{\mathtt{bdcon}}_3} \leftarrow \mathtt{Init_X}(\widetilde{\mathtt{bdcon}}_3, \boldsymbol{r_T}, \boldsymbol{r_{Z_{[\delta]}}})$

3 $\mathbf{T}^{\mathtt{X}}_{\widetilde{\mathtt{rescon}}_3} \leftarrow \mathtt{Init_X}(\widetilde{\mathtt{rescon}}_3, \boldsymbol{r_T}, \boldsymbol{r_{Z_{[\delta]}}})$

4 $\mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,BD}} \leftarrow \mathtt{Init_X}(\tilde{V}_{3,BD}, \boldsymbol{r_{Z_{[\delta,\mu-1)}}})$

5 $\mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,rc_3}} \leftarrow \mathtt{Init_X}(\tilde{V}_{3,rc_3}, \boldsymbol{r_{Z_{[\delta,\mu)}}})$

6 $\mathbf{T}^{\mathtt{Y}}_{f_1} \leftarrow \mathtt{Init_copy_Y}(\widetilde{\mathtt{copycon}}, \mathbf{T}^{\mathtt{X}}_{V_{3,ptxt}}, \boldsymbol{r_T}, \boldsymbol{r_{Z_{[\#(copy)]}}})$

7 The next two steps are executed in parallel:

8 $\mathbf{a}'_{[m']} \leftarrow \mathtt{SumcheckProduct}(\sum_{\mathbf{x}\in\{0,1\}^m} \widetilde{\mathtt{copycon}}(\boldsymbol{r_T}, \boldsymbol{r_{Z_{[\#(copy)]}}}, \mathbf{x}, \mathbf{y}) \cdot \tilde{V}_{3,ptxt}(\mathbf{x}_{[d]}), \mathbf{T}^{\mathtt{Y}}_{f_1}, g(\mathbf{y}), \mathbf{T}^{\mathtt{Y}}_{g}, \boldsymbol{r_Y})$

9 $\mathbf{a}_{[m']} \leftarrow \mathbf{a}'_{[m']} + \mathbf{1} \cdot \Big(\sum_{\mathbf{x}\in\{0,1\}^m} \big(\widehat{\mathtt{bdcon}}_3(\mathbf{x}) \cdot \widehat{V}_{BD}(\mathbf{x}) + \widehat{\mathtt{rescon}}_3(\mathbf{x}) \cdot \widehat{V}_{rc}(\mathbf{x}) \big) \Big)$

10 $\mathbf{T}^{\mathtt{X}}_{f_3} \leftarrow \mathtt{Init_copy_X}(\widetilde{\mathtt{copycon}}, \boldsymbol{r_T}, \boldsymbol{r_{Z_{[\#(copy)]}}}, \boldsymbol{r_Y})$

11 $\mathbf{a}_{[m',m+m']} \leftarrow \mathtt{Sumcheck_X}(\widehat{\mathtt{copycon}}, \mathbf{T}^{\mathtt{X}}_{f_3}, \widehat{V}_{pt}, \mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,ptxt}} \cdot g(\boldsymbol{r_Y}), \widehat{\mathtt{bdcon}}_3, \mathbf{T}^{\mathtt{X}}_{\widetilde{\mathtt{bdcon}}_3}, \widehat{V}_{BD}, \mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,BD}}, \widehat{\mathtt{rescon}}_3, \mathbf{T}^{\mathtt{X}}_{\widetilde{\mathtt{rescon}}_3}, \widehat{V}_{rc}, \mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,rc_3}})$

12 $\mathbf{a}_{[m+m']} \leftarrow \mathbf{a}_{[m+m']} + \widetilde{\mathtt{bdcon}}_2(\boldsymbol{r_T}) \cdot \tilde{V}_{2,d_2}(\boldsymbol{r_{Z_{[\mu-1]}}}) + \widetilde{\mathtt{rescon}}_1(\boldsymbol{r_T}) \cdot \tilde{V}_1(\boldsymbol{r_{Z_{[\mu]}}}) - \widetilde{\mathtt{copycon}}_{3,copy}(\boldsymbol{r_T}) \cdot \tilde{V}_{3,copy}(\boldsymbol{r_{Z_{[\#(copy)]}}})$

13 **return** $\mathbf{a}_{[m+m']}$

Algorithm 6: `Sumcheck` Equation (11)

Output: $\mathbf{a}_{[2\delta]}$, which are the prover sumcheck messages for Equation (11).

1 *Comment: The challenges* $\boldsymbol{r_Z}$ *of this equation are those output by the sumcheck of Eq. (7) for* $\tilde{V}_1$*. Still, the table of* $\tilde{V}_{3,BD}$ *must be rebuilt from scratch, since the one from line 4 of Algorithm 5 is conditioned on a different block of* $\mathtt{Z}$*.*

2 $\mathbf{T}^{\mathtt{X}}_{\widetilde{\mathtt{evk}}} \leftarrow \mathtt{Init_X}(\widetilde{\mathtt{evk}}, \boldsymbol{r_{Z_{[\delta+1]}}})$

3 $\mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,BD}} \leftarrow \mathtt{Init_X}(\tilde{V}_{3,BD}, \boldsymbol{r_{Z_{(\delta,\mu)}}})$

4 $\mathbf{a}'_{[2\delta]} \leftarrow \mathtt{SumcheckProduct}(\widetilde{\mathtt{evk}}(\boldsymbol{r_{Z_{[\delta+1]}}}, \mathbf{x}), \mathbf{T}^{\mathtt{X}}_{\widetilde{\mathtt{evk}}}, \widehat{V}_{BD}(\mathbf{x}), \mathbf{T}^{\mathtt{X}}_{\tilde{V}_{3,BD}}, \boldsymbol{r_X})$

5 $\mathbf{a}_{[2\delta]} \leftarrow \mathbf{a}'_{[2\delta]} + \tilde{V}_{2,d_{01}}(\boldsymbol{r_Z})$

6 **return** $\mathbf{a}_{[2\delta]}$

Algorithm 7: `Sumcheck` Equation (32)

Output: $\mathbf{a}_{[\mu+1]}$, which are the prover sumcheck messages for Equation (32).

1 $\mathbf{T}^{\mathtt{Z}}_{\widetilde{\mathtt{C}}_{2,d_{01}}} \leftarrow \mathtt{Init_X}(\widetilde{\mathtt{C}}_{2,d_{01}}, \mathbf{r}_{d_{01}})$

2 $\mathbf{T}^{\mathtt{Z}}_{\widetilde{\mathtt{C}}_{2,d_2}} \leftarrow \mathtt{Init_X}(\widetilde{\mathtt{C}}_{2,d_2}, \mathbf{r}_{d_2})$

3 $\mathbf{a}_{[\mu+1]} \leftarrow \mathtt{SumcheckProduct}(s_{01}\widetilde{\mathtt{C}}_{2,d_{01}}(\mathbf{r}_{d_{01}}, \mathbf{z}) + s_2\widetilde{\mathtt{C}}_{2,d_2}(\mathbf{r}_{d_2}, \mathbf{z}), s_{01}\mathbf{T}^{\mathtt{Z}}_{\widetilde{\mathtt{C}}_{2,d_{01}}} + s_2\mathbf{T}^{\mathtt{Z}}_{\widetilde{\mathtt{C}}_{2,d_2}}, \tilde{V}_2(\mathbf{z}), \mathbf{T}^{\mathtt{Z}}_{\tilde{V}_2}, \boldsymbol{r}_{\boldsymbol{Z}})$

4 **return** $\mathbf{a}_{[\mu+1]}$

For Equation (12), follow essentially `SumCheckGKR` from Libra [XZZ+19].

Exploiting sparsity *Function Evaluations for sparse polynomials* There are different MLEs that we use in the algorithms above which are very sparse. In particular, this is the case for $\widetilde{\mathtt{bdcon}}_3$ and $\widetilde{\mathtt{rescon}}_3$. The former only has $D(D+1)/2$ non-zero evaluations in $\mathtt{X}$, despite that vector being much longer and the latter, only $2D$. Moreover, these predicates are sparse in a very structured way, where all non-zero entries share the all-one vector as a common prefix: the padding factors $\widetilde{\mathtt{eq}}(\mathtt{X}_{[2\delta,m)}, \mathbf{1})$ and $\widetilde{\mathtt{eq}}(\mathtt{X}_{[\delta+1,m)}, \mathbf{1})$ pin the $L = m - 2\delta$, resp. $L = m - \delta - 1$, most significant positions. Algorithm 8 shows how to compute the sumcheck messages for this functions more efficiently. Whenever `FunctionEvaluations` would be called upon these predicates, replace it with `SparseFuncEval`. Notice that whenever Algorithm 8 is called upon $f \in \{\widetilde{\mathtt{bdcon}}_3, \widetilde{\mathtt{rescon}}_3\}$, they have already become a function only in $\mathtt{X}$ variables, due to the previous execution of $\mathtt{Init_X}(f, \boldsymbol{r}_{\boldsymbol{T}}, \boldsymbol{r}_{\boldsymbol{Z}_{[\delta]}})$.

SumcheckProduct with different length and sparsity Using Algorithm 8, we construct a more efficient, sparse version of `SumcheckProduct` which we denote `SparseSumcheckProduct` and provide in Algorithm 9. This is a replacement for lines 5 and 6 in Algorithm 4.

Algorithm 8: Algorithm `SparseFuncEval`$(f(\mathtt{X}_0, \ldots, \mathtt{X}_{m-1}), \mathbf{T}_f, L, \boldsymbol{r}_{\mathtt{X}})$

Input : Multi-linear f and its initial bookkeeping table $\mathbf{T}_f$. Prefix length L, i.e. length of the all-one vector $\mathbf{1}$. Random challenges $\mathbf{r} \in \hat{S}_N^?$.

Output: All function evaluations for sumcheck.

for $i \leftarrow 0$ **to** $m-1$ **do**
  if $i < m - L$ **then**
    foreach $\boldsymbol{b} \in \{0,1\}^{m-i-1}$ **do**
      for $t \leftarrow 0$ **to** 2 **do**
        $f(r_0, \ldots, r_{i-1}, t, \mathbf{b}_{[m-i-1]}) \leftarrow$
        $\mathbf{T}_f[\mathbf{0}_{[i]}, 0, \mathbf{b}_{[m-i-1]}] \cdot (1-t) + \mathbf{T}_f[\mathbf{0}_{[i]}, 1, \mathbf{b}_{[m-i-1]}] \cdot t$;
      end
      $\mathbf{T}_f[\mathbf{0}_{[i+1]}, \mathbf{b}_{[m-i-1]}] \leftarrow$
      $\mathbf{T}_f[\mathbf{0}_{[i]}, 0, \mathbf{b}_{[m-i-1]}] \cdot (1-r_i) + \mathbf{T}_f[\mathbf{0}_{[i]}, 1, \mathbf{b}_{[m-i-1]}] \cdot r_i$;
    end
  end
  else if $i \geq m - L$ **then**
    for $t \leftarrow 0$ **to** 2 **do**
      $f(\boldsymbol{r}_{[i]}, t, \mathbf{1}_{[m-i-1]}) \leftarrow \mathbf{T}_f[\mathbf{0}_{[i]}, \mathbf{1}_{[m-i]}] \cdot t$
    end
    $\mathbf{T}_f[\mathbf{0}_{[i+1]}, \mathbf{1}_{[m-i-1]}] \leftarrow \mathbf{T}_f[\mathbf{0}_{[i]}, \mathbf{1}_{[m-i]}] \cdot r_i$;
  end
end
Let $\mathcal{F}$ contain all function evaluations $f(\cdot)$ computed above. **return** $\mathcal{F}$

Algorithm 9: Algorithm `SparseSumcheckProduct`$(f, \mathbf{T}_f^{\mathtt{X}}, g, \mathbf{T}_g^{\mathtt{X}}, \mathbf{r})$

Input : Multi-linear $f(\mathtt{X}_0, \ldots, \mathtt{X}_{L-1})$, $g(\mathtt{X}_{[L']})$ where $L' < L$. f is a sparse wiring predicate where non-zero values share the prefix in $\mathtt{X}$. Random challenges $\mathbf{r} \in \hat{S}_N^L$.

Output: sumcheck messages for $\sum_{\mathbf{x}} f(\mathbf{x}) \cdot g(\mathbf{x}_{[L']})$ Each message $a_{i,t}$ consists of 3 elements.

$\mathcal{F} \leftarrow$ `SparseFuncEval`$(f, \mathbf{T}_f^{\mathtt{X}}, \boldsymbol{r}_{\boldsymbol{X}})$.
$\mathcal{G} \leftarrow$ `FunctionEvaluations`$(g, \mathbf{T}_g^{\mathtt{X}}, \boldsymbol{r}_{\boldsymbol{X}_{[\boldsymbol{L'}]}})$.
for $i \leftarrow 0$ **to** $L-1$ **do**
  for $t = 0, 1, 2$ **do**
    if $i < L'$ **then**
      $a_{i,t} =$
      $\sum_{\mathbf{b}_{[i+1,L)} \in \{0,1\}^{L-i-1}} f(r_0, \ldots, r_{i-1}, t, \mathbf{b}) \cdot g(r_0, \ldots, r_{i-1}, t, \mathbf{b}_{[i+1,L')})$
    end
    else if $L' \leq i < L$ **then**
      $a_{i,t} = \sum_{\mathbf{b}_{[i+1,L)} \in \{0,1\}^{L-i-1}} f(r_0, \ldots, r_{i-1}, t, \mathbf{b}) \cdot g(r_0, \ldots, r_{L'-1})$
    end
  end
end
return $\mathbf{a} = (a_0, \ldots, a_{L-1})$

F Putting everything together: The Π_{CKKS} PIOP

We are finally in a position to define our global PIOP Π_{CKKS}. In [CCC+25], the final claimed values of Π_{AC} were reduced to claims about $D+1$ oracles: $[\![\tilde{V}_{out}]\!]$ for the claimed ciphertext outputs, $[\![\tilde{V}_{aux,2^u}]\!]$ for all non-deterministic inputs from the prover that need to be proven smaller than 2^u and finally, for $j = 1, \ldots, D-1$, $[\![\tilde{V}_{aux,j}]\!]$ for non-deterministic inputs from the prover that need to be proven smaller than q/q_{L-j-1}. To those oracles, our richer arithmetization has added $[\![\tilde{V}_{3,ptxt}]\!]$ and $[\![\tilde{V}_{3,copy}]\!]$.

In the next section, we explain how the different oracles from our Π_{AC} in Section 5.2 can be recombined so that we optimize our use of range proofs from Section 4 and polynomial commitments. In summary, $[\![\tilde{V}_{aux,2^u}]\!]$ can be implemented from $[\![\tilde{V}_{3,rc_3}]\!]$, $[\![\tilde{V}_{aux,j}]\!]$ from $[\![\tilde{V}_{3,BD}]\!]$, $[\![\tilde{V}_{3,rc_3}]\!]$ and $[\![\tilde{V}_{out}]\!]$ from $[\![\tilde{V}_{3,rc_3}]\!]$, $[\![\tilde{V}_{3,copy}]\!]$.

F.1 Rearranging into the final oracles and final queries

After executing the sumcheck protocol on Equation (12), we still need to match the syntax of Π_{AC} given in Π_{CKKS}. In [CCC+25], claimed values were reduced to claims about $D+1$ oracles: $[\![\tilde{V}_{out}]\!]$ for the claimed ciphertext outputs, $[\![\tilde{V}_{aux,2^u}]\!]$ for all non-deterministic inputs from the prover that need to be proven smaller than 2^u and finally, for $j = 1, \ldots, D-1$, $[\![\tilde{V}_{aux,j}]\!]$ for non-deterministic inputs from the prover that need to be proven smaller than q/q_{L-j-1}

Rather than following the strategy from [CCC+25], in our work we exploit existing relations between the oracles that were already employed during Π_{AC} to produce the ones we need. This avoids the cost of one sumcheck to do recombinations and reassignments [CCC+25, Eq (23)], together with the computation of the relevant reassigning predicates by both the prover and the verifier. Moreover, it also requires less queries.

1. Given the oracle $[\![\tilde{V}_{3,rc_3}]\!]$, we have implicit access to $[\![\tilde{V}_{3,quot}]\!]$, so we don't need to split it. Given a query $\tilde{V}_{3,quot}(\mathbf{x}_{[quot]})$, translate it to $\tilde{V}_{3,rc_3}(\mathbf{x}_{[quot]}, 1)$. Following the same idea, we can also produce oracles $[\![\tilde{V}_{aux,j}]\!]_{j=1}^{D-1}$: Given a query $\tilde{V}_{aux,j}(\mathbf{x}_{[\#(aux,j)]})$, it can be answered with $\tilde{V}_{3,rc_3}(\mathsf{bin}(j), \mathbf{x}_{[\#(aux,j)]}, 1)$.
2. Going in the opposite direction, we can also avoid using sumcheck to recombine $[\![\tilde{V}_{3,BD}]\!]$, $[\![\tilde{V}_{3,rc_3}]\!]$ into a single oracle $[\![\tilde{V}_{aux,2^u}]\!]$. Instead, we create a virtual oracle that includes the two smaller ones. All values in $[\![\tilde{V}_{3,BD}]\!]$ would belong to $[\![\tilde{V}_{aux,2^u}]\!]$, whereas for $[\![\tilde{V}_{3,rc_3}]\!]$ the legal queries are those of the form $\tilde{V}_{3,rc_3}(0, \mathbf{x}_{[\mu]})$. Let $\mathsf{Z} = (\mathsf{Z}_0, \ldots, \mathsf{Z}_{\mu+\delta-1})$. The following MLE shows us how to query the virtual oracle (we are assuming $\delta \geq 1$):

$$\tilde{V}_{aux,2^u}(\mathsf{Z}) = (1 - \mathsf{Z}_{\mu+\delta-1}) \cdot \tilde{V}_{3,BD}(\mathsf{Z}_{[\mu+\delta-1]}) + \mathsf{Z}_{\mu+\delta-1} \cdot \tilde{V}_{3,rc_3}(\mathsf{Z}_{[\mu]}, 0)$$

3. Finally, the $\tilde{V}_{out}$ oracle can also be easily implemented by querying directly $[\![\tilde{V}_{3,rc_3}]\!]$ or $[\![\tilde{V}_{3,copy}]\!]$ at the relevant wire that has to be revealed.

4. Hence, merely by implementing $[\![\tilde{V}_{3,copy}]\!], [\![\tilde{V}_{3,rc_3}]\!]$ and $[\![\tilde{V}_{3,BD}]\!]$ through a polynomial commitment, we can verify all claimed values $\tilde{V}_{3,quot}(\mathbf{r}_{quot})$, $\tilde{V}_{3,quot}(\mathbf{r}'_{quot}), \tilde{V}_{3,BD}(\mathbf{r}_{BD})$, $\tilde{V}_{3,rc_3}(\mathbf{r}_{rc_3})$, $\tilde{V}_{3,copy}(\mathbf{r}_{copy})$ and $\tilde{V}_{3,copy}(\mathbf{r}'_{copy})$ as well as we can perform the necessary range checks. Moreover, we apply a final optimization: we implement the three oracles as a *single* PC with $2+\max(\#(copy), \mu+1, \mu+\delta-1)$ variables. This can be done using the same idea as when we defined $[\![\tilde{V}_{aux,2^u}(\mathsf{Z})]\!]$

G Auxiliary Material on SparsePack

G.1 Our SparsePack PIOP for Evaluating Sparse Polynomials

In this section, we provide a full description of SparsePack. Precisely, we denote by SparsePack-Π_{iptree} its version that internally relies on a PIOP Π_{iptree}.

SparsePack Indexer. On input the relation's index $(p_0, \ldots, p_{a-1})$, define the polynomial $\mathsf{pol} \in \mathbb{Z}_q^{(\leq 1)}[X_1, \ldots, X_\ell, T_1, T_2, Z_1, \ldots, Z_{\alpha+\mu}]$ such that, for all $\boldsymbol{b} \in \{0,1\}^\ell$, $\boldsymbol{i} \in \{0,1\}^\alpha$, $\boldsymbol{j} \in \{0,1\}^\mu$ and $\boldsymbol{j}' \in \{0,1\}^{\mu-1}$:

$$\begin{aligned}
\mathsf{pol}(\boldsymbol{b}, 0, 0, \boldsymbol{i}, \mathbf{0}) &= \mathsf{val}(\boldsymbol{b}, \boldsymbol{i}) \\
\mathsf{pol}(\boldsymbol{b}, 1, 0, \boldsymbol{i}, \boldsymbol{j}) &= \mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}) \\
\mathsf{pol}(\boldsymbol{b}, 1, 1, \boldsymbol{i}, \boldsymbol{j}', 0) &= \mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}', 0) \cdot \mathsf{ind}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}', 1), \\
\mathsf{pol}(\boldsymbol{b}, 1, 1, \boldsymbol{i}, \boldsymbol{j}', 1) &= 0
\end{aligned}$$

and pol is 0 on all the remaining points of $\{0,1\}^{\ell+2+\alpha+\mu}$. Note that, by multilinearity, the following identities hold as polynomials, where ind^* is defined as in Section 6:

$$\mathsf{pol}(\boldsymbol{X}, 1, 0, \boldsymbol{Z}) = \mathsf{ind}(\boldsymbol{X}, \boldsymbol{Z}), \quad \mathsf{pol}(\boldsymbol{X}, 1, 1, \boldsymbol{Z}_{[1,\alpha+\mu-1]}, 0) = \mathsf{ind}^*(\boldsymbol{X}, \boldsymbol{Z}_{[1,\alpha+\mu-1]}).$$

The indexer returns $\mathtt{i} = [\![\overline{\mathsf{pol}}]\!]$, where $\overline{\mathsf{pol}} = \mathsf{pack}(\mathsf{pol}) \in R_q^{(\leq 1)}[X_1, \ldots, X_n]$ and $n = \ell + 2 + \alpha + \mu - \nu$.

SparsePack-Π_{iptree} online phase.

1. $\mathcal{V}$ sends $\boldsymbol{t} \leftarrow\!\!\$\ \hat{S}_N^\alpha$.
2. $\mathcal{P}$ and $\mathcal{V}$ run Π_{iptree} for $((\tilde{y}(\boldsymbol{t}), [\![\mathsf{val}]\!], [\![f_2]\!], [\![E^*_{\mu-1}]\!]); \mathsf{val}, f_2, E^*_{\mu-1}) \in \mathcal{R}_{iptree}$, where $\mathsf{pol} = \mathsf{unpack}(\overline{\mathsf{pol}})$, $\mathsf{val} = \mathsf{pol}(\boldsymbol{X}, 0, 0, \boldsymbol{Z}_{[1,\alpha]}, \mathbf{0})$, $f_2 = \tilde{\mathsf{eq}}(\boldsymbol{X}, \boldsymbol{t})$, and $E^*_{\mu-1}$ is as in equation (15).
3. Let $\Pi_{iptree}.\mathcal{V}$'s queries be: $(\boldsymbol{r}_1, \boldsymbol{r}'_1)$ to $[\![\mathsf{val}]\!]$, $\boldsymbol{r}_2$ to $[\![f_2]\!]$, and $(\boldsymbol{r}, \boldsymbol{s})$ to $[\![E^*_{\mu-1}]\!]$. $\mathcal{P}$ sends $\psi_{\mathsf{val}} = \mathsf{val}(\boldsymbol{r}_1, \boldsymbol{r}'_1)$, $\psi_0 = \mathsf{ind}(\boldsymbol{r}, \boldsymbol{s}, 0)$, $\psi_1 = \mathsf{ind}(\boldsymbol{r}, \boldsymbol{s}, 1)$, and $\psi^* = \mathsf{ind}^*(\boldsymbol{r}, \boldsymbol{s})$.
   $\mathcal{V}$ answers the query to $[\![f_2]\!]$ by computing $\tilde{\mathsf{eq}}(\boldsymbol{r}_2, \boldsymbol{t})$ on its own, and the query to $[\![E^*_{\mu-1}]\!]$ by computing, according to equation (15),

$$\psi = \psi_0 \cdot \tilde{x}^*_0(\boldsymbol{s}) + \psi_1 \cdot \tilde{x}^*_1(\boldsymbol{s}) + \psi^* \cdot \tilde{x}^*_2(\boldsymbol{s}) + \tilde{x}^*_3(\boldsymbol{s}).$$

4. $\mathcal{V}$ sends random $\tau, \zeta \leftarrow\!\!\$\ \hat{S}_N$.
5. $\mathcal{V}$ computes $\psi_{\mathsf{pol}} = (\psi_0 \cdot (1-\zeta) + \psi_1 \cdot \zeta) \cdot (1-\tau) + \psi^* \cdot (1-\zeta) \cdot \tau$. Note that if ψ_0, ψ_1, ψ^* are correct then $\psi_{\mathsf{pol}} = \mathsf{pol}(\boldsymbol{r}, 1, \tau, \boldsymbol{s}, \zeta)$, by the layout of pol fixed by the indexer and the multilinearity of pol in T_2 and $Z_{\alpha+\mu}$.
   $\mathcal{P}$ and $\mathcal{V}$ run the PIOP $\Pi_{ueval}^{(\kappa)}$ on $((\psi_{\mathsf{val}}, \boldsymbol{x}_{\mathsf{val}}, \psi_{\mathsf{pol}}, \boldsymbol{x}_{\mathsf{pol}}, [\![\overline{\mathsf{pol}}]\!]); \overline{\mathsf{pol}}) \in \mathcal{R}_{ueval}$ where $\boldsymbol{x}_{\mathsf{val}} = (\boldsymbol{r}_1, 0, 0, \boldsymbol{r}'_1, \boldsymbol{0})$ and $\boldsymbol{x}_{\mathsf{pol}} = (\boldsymbol{r}, 1, \tau, \boldsymbol{s}, \zeta)$.
   $\Pi_{ueval}^{(\kappa)}$ ends with the verifier querying $[\![\overline{\mathsf{pol}}]\!]$ on $\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_\kappa \in \mathbb{Z}_q^n$.

We refer to Section 6 for an intuitive explanation of the protocol. Here we provide an additional intuition on the last part of the protocol that deals with evaluating the unpacked polynomial pol. Our solution requires querying the unpacked $[\![\mathsf{pol}]\!]$ on points in R_q, meaning that we cannot use the result of Lemma 3 to compute them from queries to $[\![\overline{\mathsf{pol}}]\!]$. The last piece of the protocol, which solves this issue, is the use of our PIOP Π_{ueval} of Section 3.2 for proving evaluations of unpacked polynomials on R_q points.

An alternative idea to avoid the use of Π_{ueval} and instead use Lemma 3 might have been to instantiate Π_{iptree} so as to use random challenges in the exceptional set $\hat{S}_0 \subset \mathbb{Z}_q$. This change would lead to querying pol on points of $\mathbb{Z}_q$. However, it would yield no gain in communication since the sumcheck of equation (14) involves $\tilde{x}$, which encodes the evaluation inputs that are in R_q. Thus using challenges in $\hat{S}_0$ would still require the prover to send full R_q elements in every round of the sumcheck, in addition to the fact that such a sumcheck, with R_q-dependent communication size, would need to be repeated several times to achieve a desired level of RBR-soundness.

Next, we state and prove the security of $\mathsf{SparsePack}$.

Theorem 10. *Let Π_{ueval} be the PIOP for $\mathcal{R}_{ueval}$ of Section 3.2, and let Π_{iptree} be a PIOP for $\mathcal{R}_{iptree}$ (with $\beta = \mu - 1$) with δ-knowledge-soundness. Then $\mathsf{SparsePack}$-Π_{iptree} is a PIOP for $\mathcal{R}_{speval}$ with δ_{speval}-knowledge soundness where $\delta_{speval} = \delta + (\alpha+2)/|\hat{S}_N| + ((2n+5\nu+1)/|\hat{S}_0|)^\kappa$, where $n = \ell + \alpha + \mu + 2 - \nu$ is the number of variables of $\overline{\mathsf{pol}}$.*

Proof. Assume we start with a false statement $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R}_{speval})$. Then the polynomial identity of equation (13) does not hold and, since both its sides are multilinear in $\boldsymbol{Z}$, the probability over the choice of $\boldsymbol{t}$ that the equality of equation (14) holds is $\leq \alpha/|\hat{S}_N|$.

Assume that equation (14) does not hold, let $\psi_{\mathsf{val}}, \psi_0, \psi_1, \psi^*$ be the values sent by the prover in step 3, and let ψ be the value computed by the verifier from them. If $\psi_0 = \mathsf{ind}(\boldsymbol{r}, \boldsymbol{s}, 0)$, $\psi_1 = \mathsf{ind}(\boldsymbol{r}, \boldsymbol{s}, 1)$, and $\psi^* = \mathsf{ind}^*(\boldsymbol{r}, \boldsymbol{s})$, then $\psi = E^*_{\mu-1}(\boldsymbol{r}, \boldsymbol{s})$ by equation (15). Hence the probability that the verifier accepts in Π_{iptree}, conditioned on ψ_0, ψ_1, ψ^* being correct and $\psi_{\mathsf{val}} = \mathsf{val}(\boldsymbol{r}_1, \boldsymbol{r}'_1)$, is $\leq \delta$ by the soundness of Π_{iptree}.

If any of ψ_0, ψ_1, ψ^* is wrong, then $\psi_{\mathsf{pol}} - \mathsf{pol}(\boldsymbol{r}, 1, \tau, \boldsymbol{s}, \zeta)$ is a nonzero polynomial of total degree 2 in (τ, ζ), hence $\psi_{\mathsf{pol}} = \mathsf{pol}(\boldsymbol{r}, 1, \tau, \boldsymbol{s}, \zeta)$ holds with probability at most $2/|\hat{S}_N|$ over the choice of τ, ζ.

Finally, if $\psi_{\mathsf{pol}} \neq \mathsf{pol}(\boldsymbol{r}, 1, \tau, \boldsymbol{s}, \zeta)$ or $\psi_{\mathsf{val}} \neq \mathsf{pol}(\boldsymbol{r}_1, 0, 0, \boldsymbol{r}'_1, \boldsymbol{0}) = \mathsf{val}(\boldsymbol{r}_1, \boldsymbol{r}'_1)$ then the verifier accepts in $\Pi^{(\kappa)}_{ueval}$ with probability at most $((2(\ell+\alpha+\mu+2)+3\nu+1)/|\hat{S}_0|)^\kappa = ((2n+5\nu+1)/|\hat{S}_0|)^\kappa$, by Theorem 1 (applied to $a=2$ evaluation points of a packed polynomial with n variables) and Theorem 4.

Furthermore, using a similar proof, if Π_{iptree} has RBR-soundness δ', then we can argue the RBR-soundness of SparsePack-Π_{iptree} to be the maximum of

$$\delta'_1 = \frac{\alpha}{|\hat{S}_N|},\ \delta',\ \delta'_4 = \frac{2}{|\hat{S}_N|},\ \left(\frac{2n+5\nu+1}{|\hat{S}_0|}\right)^{\frac{\kappa}{n+\nu}}.$$

G.2 Instantiations of SparsePack and Efficiency

To obtain SparsePack we instantiate Π_{iptree} with a PIOP that uses Thaler's approach for proving a tree of multiplications [Tha13], adapted to work for $2^{\ell+\alpha}$ trees of multiplications of depth $\mu-1$, done in parallel, followed by an inner product. We refer to the next section for more details on this protocol. Here we recall that it can be proven by performing $\mu-1$ sequential sumchecks of increasing width $2^{\ell+\alpha+\eta}$, for $\eta=0$ to $\mu-2$, which, as required by Π_{iptree}, end up with the verifier querying the polynomials $\mathsf{val}(\boldsymbol{r}_1, \boldsymbol{r}'_1)$, $E^*_{\mu-1}(\boldsymbol{r}, \boldsymbol{s})$, and computing $\tilde{\mathsf{eq}}(\boldsymbol{r}'_1, \boldsymbol{t})$. The query to $E^*_{\mu-1}(\boldsymbol{r}, \boldsymbol{s})$ is answered by the verifier itself from the values $\psi_0 = \mathsf{ind}(\boldsymbol{r}, \boldsymbol{s}, 0)$, $\psi_1 = \mathsf{ind}(\boldsymbol{r}, \boldsymbol{s}, 1)$, and $\psi^* = \mathsf{ind}^*(\boldsymbol{r}, \boldsymbol{s})$ sent by the prover, so no additional sumcheck is needed. Finally, all these queries are proven with an execution of $\Pi^{(\kappa)}_{ueval}$.

In total, SparsePack's communication consists of $4\ell(\mu-1)+(4\mu-1)\alpha+2\mu^2-2\mu+2$ elements of H_q (see the paragraph on diagonal challenges below), $\kappa(4(\ell+\alpha+\mu)+1)$ elements of $\mathbb{Z}_q$, and one PC evaluation proof for a polynomial of $\ell+\alpha+\mu+2-\nu$ variables.

We compare SparsePack with Spark, as specified in Spartan [Set20], with the counter polynomials produced and committed in preprocessing, and with c dimensions as in [STW24]. We compile it with the PC transform for R_q of [CCC+25] over Basefold, i.e., one BaseFold PC on each CRT component of R_q, without batching across them. As in Spartan, the grand products of all the multiset checks are proven as a single data-parallel layered circuit [Tha13], with one sumcheck per layer. For $c = \lceil m/\ell \rceil$, this solution would require: $3 \cdot a \cdot c$ committed polynomials of size 2^ℓ in the preprocessing, $a \cdot c$ committed polynomials of size 2^ℓ generated at proving time, an overall communication of $(c+2)\ell + 2(\lceil \log c \rceil + \alpha + 1)\ell(\ell-1) + 2\ell$ elements of R_q, and the PC evaluation proofs for $4ac$ multilinear polynomials of ℓ variables, queried at two points, which we batch into two Basefold evaluation proofs [ZCF24].

To obtain a concrete comparison, we consider the following parameters, $N = 2^{13}, \log q \approx 135, \ell = 15, m = 34, a = 2$, which correspond to the largest circuit of one of our neural networks applications (see Section 8).

Even ignoring the size of PC evaluation proofs, SparsePack's proof is 25.4 KiB, against 235.3 MiB for Spark. More interestingly, SparsePack avoids altogether running the PC on large polynomials: it does not generate any commitment

during the proof, and it computes a *single* evaluation proof for the preprocessed (packed) polynomial $\overline{\mathsf{pol}}$ that, thanks to packing, has only 11 variables; using our new BatchFold scheme this proof is 163 MiB and, together with Π_{ueval}, takes about two minutes to compute. In contrast, Spark prover has to commit to 6 polynomials of size 2^{15} (which takes about 10 minutes) and prove evaluations of $4ac = 24$ polynomials of size 2^{15}. Using two batched Basefold proofs over R_q as in [CCC+25], these evaluations would take a total of about 7 minutes, yielding a proof of 6.2 GiB.

Diagonal challenges. The figures above and Table 2 take the verifier's challenges in $\hat{S}_N$ from its diagonal copy of H_q, i.e., one element of $\mathbb{F}_{p_i^d}$ repeated in the N/d slots of each p_i. This was already the case in [CCC+26b], but the protocol there did not exploit it. Since all the points at which the sparse polynomials are evaluated are then diagonal, every table of Π_{iptree} lies in the diagonal subring of R_q, which is isomorphic to H_q: its sumchecks run over H_q and send elements of H_q, N/d times smaller than those of R_q, and the evaluation proof of $\overline{\mathsf{pol}}$ needs no common-point sumcheck. Without this, the sumchecks of SparsePack would take 28m13s, 3m28s and 1m44s for our three circuits (instead of about one second) and send 50.8, 35.5 and 33.4 MiB; SparsePack would then be 0.7×, 1.4× and 1.3× as fast as Spark (instead of 10.6×, 23.5× and 11.2×), and still 28–31× shorter. The same choice gives the prover of Π_{ueval} the time $\mathcal{O}(a2^{\ell+\nu})$ of Theorem 1: at points that are not diagonal, building its table E directly could take N times more operations in $\mathbb{Z}_q$.

G.3 Details on our instantiation of Π_{iptree}

In this section we give more details on how to instantiate our application of the Π_{iptree} PIOP using Thaler's approach for trees of multiplications.

Consider the polynomial $E^*_{\mu-1}(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Z}')$ that is of degree 1 in $\boldsymbol{X}$ and degree 2 in $(\boldsymbol{Z}, \boldsymbol{Z}')$. Let $\tilde{E}_{\mu-2}$ be the multilinear polynomial such that

$$\forall (\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}) \in \{0,1\}^{\ell+\alpha+\mu-2} : \tilde{E}_{\mu-2}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}) = E^*_{\mu-1}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 0) \cdot E^*_{\mu-1}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 1)$$

Next, for $\eta = \mu - 3$ down to 1, let $\tilde{E}_\eta$ be the multilinear polynomial such that

$$\forall (\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}) \in \{0,1\}^{\ell+\alpha+\eta} : \tilde{E}_\eta(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}) = \tilde{E}_{\eta+1}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 0) \cdot \tilde{E}_{\eta+1}(\boldsymbol{b}, \boldsymbol{i}, \boldsymbol{j}, 1)$$

Then Π_{iptree} on $((\tilde{y}(\boldsymbol{t}), [\![\mathsf{val}]\!], [\![f_2]\!], [\![E^*_{\mu-1}]\!]); \mathsf{val}, f_2, E^*_{\mu-1}) \in \mathcal{R}_{iptree}$ runs $\mu - 1$ sumchecks, sequentially, of increasing width $2^{\ell+\alpha+\eta}$, for $\eta = 0$ to $\mu - 2$. This proceeds as follows.

For $\eta = 0$, it starts with a sumcheck of width $2^{\ell+\alpha}$ and degree 3 in ℓ variables and 4 in α variables, for the claim

$$\tilde{y}(\boldsymbol{t}) = \sum_{(\boldsymbol{b},\boldsymbol{i}) \in \{0,1\}^{\ell+\alpha}} \mathsf{val}(\boldsymbol{b}, \boldsymbol{i}) f_2(\boldsymbol{i}) \cdot \tilde{E}_1(\boldsymbol{b}, \boldsymbol{i}, 0) \cdot \tilde{E}_1(\boldsymbol{b}, \boldsymbol{i}, 1)$$

Its communication is $4\ell + 5\alpha + 2$ elements of R_q, and it ends with a "promise" claim $v_1 = \tilde{E}_1(\boldsymbol{r}_1)$.

For $\eta = 1$ to $\mu - 3$, it runs a sumcheck of width $2^{\ell+\alpha+\eta}$ and degree 3 in all $\ell + \alpha + \eta$ variables, for the claim

$$v_\eta = \sum_{(\boldsymbol{b},\boldsymbol{i},\boldsymbol{j})\in\{0,1\}^{\ell+\alpha+\eta}} \tilde{E}_{\eta+1}(\boldsymbol{b},\boldsymbol{i},\boldsymbol{j},0)\cdot\tilde{E}_{\eta+1}(\boldsymbol{b},\boldsymbol{i},\boldsymbol{j},1)\cdot\tilde{\mathsf{eq}}((\boldsymbol{b},\boldsymbol{i},\boldsymbol{j}),\boldsymbol{r}_\eta)$$

Its communication is $4(\ell+\alpha+\eta)+2$ elements of R_q, and it ends with a "promise" claim $v_{\eta+1} = \tilde{E}_{\eta+1}(\boldsymbol{r}_{\eta+1})$.

For $\eta = \mu-2$, it runs a sumcheck of width $2^{\ell+\alpha+\mu-2}$ of degree 3 in ℓ variables and 5 in $\alpha + \eta$ variables, for the claim

$$v_{\mu-2} = \sum_{(\boldsymbol{b},\boldsymbol{i},\boldsymbol{j})\in\{0,1\}^{\ell+\alpha+\mu-2}} E^*_{\mu-1}(\boldsymbol{b},\boldsymbol{i},\boldsymbol{j},0)\cdot E^*_{\mu-1}(\boldsymbol{b},\boldsymbol{i},\boldsymbol{j},1)\cdot\tilde{\mathsf{eq}}((\boldsymbol{b},\boldsymbol{i},\boldsymbol{j}),\boldsymbol{r}_\eta)$$

Its communication is $4\ell + 6(\alpha+\mu-2)+2$ elements of R_q, and it ends with the claim $\psi = E^*_{\mu-1}(\boldsymbol{r},\boldsymbol{s})$.

Therefore the total communication of this protocol is $4\ell(\mu-1)+(4\mu-1)\alpha+2\mu^2-2\mu-2$ elements of R_q.

H Details about BaseFold

In this section we describe the BaseFold polynomial commitment, and our protocol $\mathsf{Eval}^{\mathsf{com}}_M$ for batched openings at a common point built on top on it, in detail. Moreover, we show a slight improvement on the lower bounds for the minimum distance of the codes involved, that translates into a better soundness error.

H.1 The BaseFold Error Correcting Codes

The central component of the BaseFold polynomial commitment is a probabilistic construction of a family of linear codes. Starting from a fixed code C_0, the procedure constructs $\{C_i\}_{i\in\mathbb{N}}$ such that the length and dimension of C_i are double the ones from C_{i-1}. Thus, if k_i and n_i respectively denote the dimension and length of C_i, we have that $k_i = 2^i k_0$ and $n_i = 2^i n_0$. For each i, a vector $\boldsymbol{t}_{i-1}$ is chosen uniformly at random in $(\mathbb{F}^*)^{n_{i-1}}$(once and for all). The encoding function Enc_i with code C_i is then defined from Enc_{i-1} and $\boldsymbol{t}_{i-1}$ as described in Figure 11.

H.2 The BaseFold Commitment and Evaluation

In order to commit to a multilinear polynomial f in $\mathbb{F}[X_1,\ldots,X_\ell]$, we set up a family of codes $\{C_i\}_{i=0}^d$ as above such that $k_0 = 2^{\ell_0}$ and $d = \ell - \ell_0$. Note that $k_d = 2^\ell$, which is the number of coefficients of f. We identify $f = \sum_{\boldsymbol{j}\in\{0,1\}^\ell} f_{\boldsymbol{j}}\mathbf{X}^{\boldsymbol{j}}$ with the vector $\boldsymbol{f}\in\mathbb{F}^{k_d}$ given by $\boldsymbol{f}[\mathsf{to\text{-}int}(\boldsymbol{j})] = f_{\boldsymbol{j}}$.

The commitment to f is then π_d, an oracle to $\mathsf{Enc}_d(\boldsymbol{f})$, which in practice will be instantiated with a vector commitment such as a Merkle-tree based commitment.

Encoding algorithm for C_i in Basefold

Recursive encoding function Enc_i for messages in $\mathbb{F}^{k_i}$, with $k_i = 2^i k_0$. It requires to have fixed a $[n_0, k_0]$-linear code C_0 with generator matrix $G_0 \in \mathbb{F}^{k_0 \times n_0}$, and for each $j < i$, a vector $\boldsymbol{t}_j \in (\mathbb{F}^*)^{n_j}$ (where $n_j = 2^j n_0$).

- If $i = 0$, then $\mathsf{Enc}_0(\mathbf{m}) = \mathbf{m}G_0 \in \mathbb{F}^{n_0}$
- Else if $i > 0$ the encoding of $\mathbf{m} = (\mathbf{m}_L, \mathbf{m}_R) \in \mathbb{F}^{k_i}$ where $\mathbf{m}_L, \mathbf{m}_R \in \mathbb{F}^{k_{i-1}}$ is given by

$$\begin{aligned}\mathsf{Enc}_i(\mathbf{m}) = &\\ &(\mathsf{Enc}_{i-1}(\mathbf{m}_L), \mathsf{Enc}_{i-1}(\mathbf{m}_L))\\ &+ (\mathsf{Enc}_{i-1}(\mathbf{m}_R), \mathsf{Enc}_{i-1}(\mathbf{m}_R)) \ \circ \ (\mathbf{t}_{i-1}, -\mathbf{t}_{i-1})\end{aligned}$$

where $\circ$ denotes the componentwise product.
In other words $\mathsf{Enc}_i(\mathbf{m}) = \mathbf{m}G_i$, where the generator matrix G_i of C_i is

$$G_i = \begin{pmatrix} G_{i-1} & G_{i-1} \\ G_{i-1} \cdot T_{i-1} & G_{i-1} \cdot -T_{i-1} \end{pmatrix}$$

where $T_{i-1} \in \mathbb{F}^{n_{i-1} \times n_{i-1}}$ is the diagonal matrix with diagonal $\mathbf{t}_{i-1}$.

Figure 11: Encoding algorithm for C_i in Basefold

Rather than describing the protocol Eval from [ZCF24] for a single opening of a committed polynomial, we already incorporate our additional steps to obtain $\mathsf{Eval}_M^{\mathsf{com}}$, the protocol that simultaneously proves the correct evaluation of M committed polynomials at the same point; the original Basefold evaluation protocol can be retrieved as the case $M = 1$.

Concretely the protocol $\mathsf{PC.Eval}_M^{\mathsf{com}}$ is a proof for the following relation:

$$\begin{aligned}\mathcal{R}_{\mathsf{Eval}_M^{\mathsf{com}}} = \big\{ &((\boldsymbol{z}, (y_m)_{m=0}^{M-1}, (\pi_{d,f_m})_{m=0}^{M-1}); (f_m)_{m=0}^{M-1}) :\\ &\pi_{d,f_m} \text{ oracle to } \mathsf{Enc}_d(\boldsymbol{f}_m) \wedge f_m(\boldsymbol{z}) = y_m, \forall m \in [M]\}\end{aligned}$$

The protocol $\mathsf{PC.Eval}_M^{\mathsf{com}}$ can be found in Figure 12.

We now prove that the soundness of $\mathsf{PC.Eval}_M^{\mathsf{com}}$.

Lemma 9 (Lemma 5 restated). *Fix a finite field $\mathbb{F}$ an d let $\{C_i\}_{i=0}^d$ linear codes over $\mathbb{F}$ be defined as above. If $\mathsf{error}_{\mathsf{PC.Eval}_M^{\mathsf{com}}}$ and $\mathsf{error}_{\mathsf{PC.Eval}}$ denote respectively the soundness error of $\mathsf{PC.Eval}_M^{\mathsf{com}}$ and $\mathsf{PC.Eval}$ then $\mathsf{error}_{\mathsf{PC.Eval}_M^{\mathsf{com}}} \leq \frac{M-1}{|\mathbb{F}|} + \mathsf{error}_{\mathsf{PC.Eval}}$.*

Proof. We note that steps 2-9 are a proof $\mathsf{PC.Eval} = \mathsf{PC.Eval}_1^{\mathsf{com}}$ for the claim that $(\boldsymbol{z}, y', \pi_{d,u}) \in \mathcal{L}(\mathcal{R}_{\mathsf{Eval}_1^{\mathsf{com}}})$ where $y' = \sum_{m=0}^{M-1} s^m y_m$ and $\pi_{d,u} = \sum_{m=0}^{M-1} s^m \pi_{d,m}$ where $\pi_{d,m}$ are the oracles in the initial statement.

Hence it is enough to show that if the received starting claim is false, i.e. $(\boldsymbol{z}, (y_m)_{m=0}^{M-1}, (\pi_{d,m})_{m=0}^{M-1}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{Eval}_M^{\mathsf{com}}})$ then $(\boldsymbol{z}, y', \pi_{d,u}) \in \mathcal{L}(\mathcal{R}_{\mathsf{Eval}_1^{\mathsf{com}}})$ can happen with probability at most $(M-1)/|\mathbb{F}|$ over the choice of s.

$\mathsf{PC.Eval}_M^{\mathsf{com}}$: Batched opening proof at common point $\boldsymbol{z}$

Inputs: oracles π_{d,f_m}, evaluations y_m, $m \in [M]$, and evaluation point $\boldsymbol{z} \in \mathbb{F}^\ell$. Proof that $f_m(\boldsymbol{z}) = y_m$, $m \in [M]$.

1. If $M > 1$, the verifier selects $s \in \mathbb{F}$, sends it to the prover. The prover sets $u(\boldsymbol{X}) := \sum_{m=0}^{M-1} s^m f_m(\boldsymbol{X})$. This implicitly defines $\pi_d := \pi_{d,u} = \sum_{m=0}^{M-1} s^m \pi_{d,f_m}$. Else, if $M = 1$, define $\pi_d := \pi_{d,f_0}$, $u := f_0$
2. For $i = d-1, \ldots, 1$,
   (a) The verifier selects $\alpha_i \in \mathbb{F}$ uniformly at random and sends it to the prover.
   (b) The prover constructs $\pi_i = (g_1(\alpha_i), \ldots, g_{n_i}(\alpha_i))$ where $g_j(X)$ is the unique polynomial of degree 1 with $g_j(\boldsymbol{t}_i[j]) = \pi_{i+1}[j]$ and $g_j(-\boldsymbol{t}_i[j]) = \pi_{i+1}[j+n_i]$.
   (c) If $i > 0$, the prover sends oracle π_i to the verifier.
   (d) In addition, the prover constructs the univariate polynomial
   $$h_i(X) = \sum_{\mathbf{b} \in \{0,1\}^{\ell_0+i-1}} u(\mathbf{b}, X, \alpha_i, \ldots, \alpha_{d-1}) \cdot \widetilde{\mathsf{eq}}((\mathbf{b}, X, \alpha_i, \ldots, \alpha_{d-1}), \mathbf{z})$$
   and sends $h_i(X)$ to the verifier.
3. The verifier selects $\alpha_0 \in \mathbb{F}$ uniformly at random and sends it to the prover.
4. The prover sends $\boldsymbol{m}$, the vector of coefficients of the ℓ_0-variate polynomial $u(X_1, \ldots, X_{\ell_0}, \alpha_0, \ldots, \alpha_{d-1})$.
5. The verifier computes $\pi_0 = \mathsf{Enc}_0(\boldsymbol{m})$.
6. The verifier now verifies the validity of the commitment by executing rp times the following check. For ι in $1, \ldots, \mathsf{rp}$:
   (a) Samples $\mu := \mu_\iota \in [1, \ldots, n_{d-1}]$
   (b) For each $i = d-1, \ldots, 0$:
      i. Queries $\pi_{i+1}[\mu]$ and $\pi_{i+1}[\mu + n_i]$ if they have not been queried before. If $M > 1$, the queries with $i = d-1$ are carried out by querying all $(\pi_{d,f_m}[\mu], \pi_{d,f_m}[\mu + n_i])_{m=0}^{M-1}$ and defining $\pi_d[\mu] = \sum_{m=0}^{M-1} s^m \pi_{d,f_m}[\mu]$ and analogously with $\pi_d[\mu + n_i]$.
      ii. Constructs $g_\mu(X)$ from $\pi_{i+1}[\mu]$ and $\pi_{i+1}[\mu + n_{i-1}]$ as above.
      iii. Queries $\pi_i[\mu]$ and checks $g_\mu(\alpha_i) \stackrel{?}{=} \pi_i[\mu]$.
      iv. Updates μ to $(\mu \bmod n_{i-1})$.
7. The verifier checks
   $$h_1(\alpha_0) \stackrel{?}{=} \sum_{\mathbf{b} \in \{0,1\}^{\ell_0}} u(\mathbf{b}, \alpha_0, \ldots, \alpha_{d-1}) \cdot \widetilde{\mathsf{eq}}((\mathbf{b}, \alpha_0, \ldots, \alpha_{d-1}), \mathbf{z}),$$
   using the received $\boldsymbol{m}$ to compute the evaluations of f on the right hand side.
8. The verifier also checks
   $$h_d(0) + h_d(1) \stackrel{?}{=} \sum_{m=0}^{M-1} s^m y_m$$
   and
   $$h_i(0) + h_i(1) \stackrel{?}{=} h_{i+1}(\alpha_i)$$
   for $i \in [1, \ldots, d-1]$
9. The proof is accepted if all checks pass.

Figure 12: $\mathsf{PC.Eval}_M^{\mathsf{com}}$: Batched opening proof of M evaluations at common point for the Basefold Polynomial Commitment.

Assume the claim is false, then there is some $m \in [M]$ such that either $\pi_{d,m}$ is not a valid oracle to $\mathsf{Enc}_d(\boldsymbol{f}_m)$ for some $\boldsymbol{f}_m \in \mathbb{F}^{k_d}$ or $f_m(\boldsymbol{z}) \neq y_m$ for the associated f_m.

First suppose that for some m, π_{d,f_m} is a oracle to a vector $\boldsymbol{v}_m \notin C_d$. Consider a parity check matrix $H \in \mathbb{F}^{(n_d-k_d)\times n_d}$ i.e. a matrix H such that for every $\boldsymbol{c} \in \mathbb{F}^{n_d}$, $\boldsymbol{c} \in C_d$ iff $H\boldsymbol{c} = 0$. Consequently since $\boldsymbol{v}_m \notin C_d$ there is a row H_i of H with $H_i\boldsymbol{v}_m \neq 0$. Then the polynomial $h(T) = \sum_{m=0}^{M-1}(H_i\boldsymbol{v}_m)\cdot T^m$ is nonzero, so it has at most $M-1$ roots and hence $Pr[h(s)=0] \leq (M-1)/|\mathbb{F}|$ when s is sampled uniformly at random in $\mathbb{F}$. But if $h(s) \neq 0$, then by linearity $H_i(\sum_{m=0}^{M-1} s^m\boldsymbol{v_m}) \neq 0$ so $\sum_{m=0}^{M-1} s^m\boldsymbol{v_m} \notin C_d$ and therefore $\pi_{d,u}$ is not an oracle to a codeword in C_d, so $(\boldsymbol{z}, y', \pi_{d,u}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{Eval}_1^{\mathsf{com}}})$.

Otherwise for all m, $\pi_{d,m}$ is a oracle to some $\mathsf{Enc}_d(\boldsymbol{f}_m)$, and clearly $\pi_{d,u}$ is an oracle to $\mathsf{Enc}_d(\boldsymbol{u})$ with $u = \sum_{m=0}^{M-1} s^m f_m$. Since the claim is false, $f_m(\boldsymbol{z}) \neq y_m$ for some m. Then in an analogous way as before, the polynomial $h'(T) := \sum_{m=0}^{M-1}(f_m(\boldsymbol{z}) - y_m)T^m$ is nonzero so $\Pr[h'(s)=0] \leq (M-1)/|\mathbb{F}|$ when s is sampled uniformly at random in $\mathbb{F}$, while if $h'(s) \neq 0$ we have $u(\boldsymbol{z}) = \sum_{m=0}^{M-1} s^m f_m(\boldsymbol{z}) \neq y'$, so again $(\boldsymbol{z}, y', \pi_{d,u}) \notin \mathcal{L}(\mathcal{R}_{\mathsf{Eval}_1^{\mathsf{com}}})$.

Now we retrieve the soundness result obtained in BaseFold. The soundness error depends on the relative minimum distance Δ_{C_i} of the codes C_i defined as $\Delta(C_i) = d_{\mathsf{min}}(C_i)/n_i$

Theorem 11 (Soundness of Basefold (case $M = 1$), from [ZCF24]). *Let $\{C_i\}_{i=0}^d$, the linear codes constructed above, and $\gamma, \delta \in (0,1)$ satisfy the conditions*[12] *$\delta < J_\gamma(J_\gamma(\Delta_{C_d}))$ and $3\delta - d\gamma < \Delta_{C_d}$, where $J_\gamma(x) := 1 - \sqrt{1-x(1-\gamma)}$. Then the soundness error of $\mathsf{PC.Eval}$ satisfies*

$$\mathsf{error}_{\mathsf{PC.Eval}} \leq \frac{2d}{\gamma^3|\mathbb{F}|} + (1-\delta+\gamma d)^{\mathsf{rp}}$$

Corollary 1. *In the same conditions as above, the soundness error of $\mathsf{error}_{\mathsf{PC.Eval}_M^{\mathsf{com}}}$ satisfies*

$$\mathsf{error}_{\mathsf{PC.Eval}_M^{\mathsf{com}}} \leq \frac{M-1}{|\mathbb{F}|} + \frac{2d}{\gamma^3|\mathbb{F}|} + (1-\delta+\gamma d)^{\mathsf{rp}}$$

The size of the proof, assuming we instantiate the oracles π_i with Merkle commitments to vectors of length $n_i = 2^i \cdot n_0$ with hash functions of output size κ bits, is:

$$(2\mathsf{rp}\log n_0(M+1) + 2\mathsf{rp}Md + \mathsf{rp}(d-1)d)\,\kappa + [3(d-1) + 2\mathsf{rp}(M+d-1) + 2^{\ell_0}]\log|\mathbb{F}|$$

bits. If we consider ℓ_0 and n_0 small, so $d \sim \ell$, and considering that for security parameter λ we need $\log|\mathbb{F}|, \kappa, \mathsf{rp} = \mathcal{O}(\lambda)$, this is

$$\mathcal{O}\big((M\ell + \ell^2)\lambda^2\big)$$

[12] Note that [ZCF24] states $\Delta_{C_{i+1}} \leq \Delta_{C_i}$ for all $i \in [0, d-1]$ as an additional condition, but this is always satisfied in the stated code construction

communication. Compare this with naively computing M individual proofs for Basefold. The communication complexity in that case would be

$$\mathcal{O}\big(M\ell^2\lambda^2\big).$$

H.3 Improvements on the minimum distance estimates of the BaseFold codes

The security of the Basefold commitment depends on the relative minimum distance $\Delta_{C_i} = d_{\min}(C_i)/n_i$ of the codes C_i. We state next a lower bound on the minimum distance of these codes which is a slight improvement with respect to the one stated in [ZCF24].

Theorem 12. *Let $\lambda \in \mathbb{N}$. Define $\{C_i\}_{i\in\mathbb{N}}$ as above, with the additional assumption that C_0 is a MDS code (i.e. $d_{\min}(C_0) = n_0 - k_0 + 1$). For every $d \in \mathbb{N}$,*

$$\Pr[d_{\min}(C_d) \leq n_d - t_d] \leq d \cdot 2^{-\lambda}$$

where $t_0 = k_0$, and for every $i > 0$, $t_i = 2t_{i-1} + \ell_i$ with

$$\ell_i = \frac{\lambda + 2t_{i-1}(1 + \log(\frac{|\mathbb{F}|}{|\mathbb{F}|-1})) + 0.585n_i}{\log(|\mathbb{F}| - 1) - 1}.$$

The probability above is over the uniform choices of $\boldsymbol{t}_i \in (\mathbb{F}^*)^{n_i}$, $i = 0, \ldots, d-1$, in the definition of the codes.

For $|\mathbb{F}| \geq 2^{11}$, we can set

$$\ell_i = \frac{\lambda + 2.002t_{i-1} + 0.585n_i}{\log|\mathbb{F}| - 1.001}.$$

For comparison, in [ZCF24, Theorem 2], the corresponding ℓ_i (for a sufficiently large field) is given as

$$\ell_i = \frac{2(i-1)\log n_0 + \lambda + 2.002t_{i-1} + 0.6n_i}{\log|\mathbb{F}| - 1.001}$$

which is less tight.

In the remaining of this section, we prove this result. For our proof we will restate the results of [ZCF24] using a more code-theoretic notation, that uses the notion of shortened code. This also simplifies somewhat the proofs.

For $n > 0$ and $S \subseteq [n]$ we denote $\pi_S : \mathbb{F}^n \to \mathbb{F}^{|S|}$ the projection to the coordinates in S. Given $S \in [n]$ we write $\bar{S} := [n] \setminus S$ for its complementary. We denote the Hamming weight of vector $\mathbf{v}$ as $w_{\mathsf{Ham}}(\mathbf{v})$. Recall that a $[n, k, d]_q$-linear code C is a linear code C over $\mathbb{F}_q$ of length n, dimension k (i.e. cardinality q^k) and minimum distance $d_{\mathsf{min}}(C) = d$.

Definition 8 (C shortened at S). *Given a code $C \subseteq \mathbb{F}^n$, the shortening of C at S (or C shortened at S) is the code $C^S = \{\pi_{\bar{S}}(\boldsymbol{c}) : c \in C, \pi_S(\boldsymbol{c}) = 0\}$. I.e. the code that we obtain by restricting to the codewords in C that are 0 in all the coordinates in S and then removing these 0-coordinates.*

Lemma 10. *If C is a $[n,k,d]_q$-linear code, then if C^S is a $[n-|S|, k_S, d_S]_q$-linear code such that $k_S \leq k$ and, if $C^S \neq \{0\}$, $d_S \geq d$.*

Proof. It is obvious that C^S is of length $n-|S|$, and that $k_S \leq k$ because the amount of codewords can only decrease. If $C^S \neq \{0\}$, there is some nonzero codeword in $\boldsymbol{w} \in C^S$ which means there is a nonzero codeword $\boldsymbol{c}$ in C with $\pi_S(\boldsymbol{c}) = 0$, $\pi_{\bar{S}}(\boldsymbol{c}) = \boldsymbol{w} \neq \boldsymbol{0}$. Then clearly $d \leq w_H(\boldsymbol{c}) = w_H(\pi_{\bar{S}}(\boldsymbol{c})) = w_H(\boldsymbol{w})$ so indeed $d_S \geq d$.

Lemma 11 (Singleton bound). *For any $[n,k,d]_q$-linear code C, we have $k+d \leq n+1$*

Proof. Well known fact in code theory.

Definition 9 (MDS code). *A MDS code is a $[n,k,d]_q$-linear code C with $k+d = n+1$.*

Definition 10. *Let $S \subseteq [n_{d-1}]$. We define $m_d(S)$ as the set of pair of codewords in C_{d-1} such that they have common zeros exactly in the coordinates of S, i.e.*

$$m_d(S) := \{(c,c') \in (C_{d-1})^2 : \forall i \in S,\ c[i] = c'[i] = 0 \\ \forall i \notin S,\ c[i] \neq 0 \vee c'[i] \neq 0\}$$

Note $(C_{d-1})^2$ is the disjoint union of $m_d(S)$ for $S \subseteq [n_{d-1}]$, as every pair of codewords belongs to exactly one $m_d(S)$.

We now offer a proof of Lemma 2 in [ZCF24] that we consider is slightly simpler, using the coding-theory definition and facts above.

Lemma 12. *[Lemma 2 in [ZCF24]]. Suppose $d_{min}(C_{d-1}) > n_{d-1} - t$ for some t. Let $S \subseteq [n_{d-1}]$. Then:*

- *If $t \leq |S| < n_{d-1}$, then $|m_d(S)| = 0$.*
- *If $|S| < t$, then $|m_d(S)| \leq |\mathbb{F}|^{2t-2|S|}$.*

Proof. First, note $|m_d(S)| \leq (|(C_{d-1})^S|)^2$, where $(C_{d-1})^S$ is C_{d-1} shortened at S. It is not necessarily an equality, because a codeword in C_{d-1}^S may be 0 in some coordinates of $\bar{S}$.

Then:

- If $|S| \geq t$, $m_d(S) = 0$, unless $S = [n_{d-1}]$ for which $m_d(S) = 1$
  Indeed, because $d_{min}(C_{d-1}) > n_{d-1} - t$ there are no nonzero codewords in C_{d-1} with t zeros, and $(\boldsymbol{0}, \boldsymbol{0}) \in m_d([n_{d-1}])$.
- If $|S| < t$, $m_d(S) \leq |\mathbb{F}|^{2t-2|S|}$.
  Indeed in this case, either $(C_{d-1})^S = \{0\}$ and then $m_d(S) = 1$ or $(C_{d-1})^S \neq \{\boldsymbol{0}\}$. Then $(C_{d-1})^S$ is a code of length $n_{d-1} - |S|$ and distance $d_{min}((C_{d-1})^S) \geq d_{min}(C_{d-1}) \geq n_{d-1} - t + 1$.
  Then, by the Singleton bound,

$$\dim(C_{d-1})^S \leq (n_{d-1} - |S|) - d_{min}((C_{d-1})^S) + 1 \leq t - |S|.$$

  Consequently

$$m_d(S) \leq |(C_{d-1})^S|^2 \leq (|\mathbb{F}|^{t-|S|})^2.$$

Now we work towards the improvement on the lower bounds. The following lemma shaves off a factor n_{d-1} with respect to Lemma 3 in [ZCF24]. The improvement stems from a better bound on the expression (35). Up to that point the proof is as in [ZCF24].

Lemma 13. *[Improvement on Lemma 3 in [ZCF24]] Let $S \subseteq [n_{d-1}]$, $(\boldsymbol{c}, \boldsymbol{c'}) \in m_d(S)$. Let $\hat{\boldsymbol{c}} = (\boldsymbol{c} + \boldsymbol{c'} \circ \boldsymbol{t}_{d-1}, \boldsymbol{c} - \boldsymbol{c'} \circ \boldsymbol{t}_{d-1})$ where $\boldsymbol{t}_{d-1}$ is chosen uniformly at random in $(\mathbb{F} \setminus \{0\})^{n_{d-1}}$. Then for $0 < t \leq n_d$,*

$$\Pr[w_H(\hat{\boldsymbol{c}}) \leq n_d - t] < 2^{n_{d-1}-|S|} \cdot \left(\frac{2}{|\mathbb{F}|-1}\right)^{t-2|S|}$$

where the probability is over the choice of $\boldsymbol{t}_{d-1}$

Proof. For all $i \in [n_{d-1} \setminus S]$, either $\boldsymbol{c}[i] \neq 0$ or $\boldsymbol{c'}[i] \neq 0$. For the sake of notation let $a_i := \boldsymbol{c}[i]$, $b_i = \boldsymbol{c'}[i]$, $u_i = \boldsymbol{t}_{d-1}[i]$

Then the Hamming weight of the pair of coordinates $(\hat{\boldsymbol{c}}[i], \hat{\boldsymbol{c}}[n_{d-1} + i])$ is

$$w_{\mathsf{Ham}}(a_i + b_i \cdot u_i, a_i - b_i \cdot u_i) = \begin{cases} 1 & \text{if } a_i \neq 0 \text{ and } u_i = \pm b_i/a_i \\ 2 & \text{otherwise} \end{cases}$$

Note that the weight cannot be 0 as this would imply $u_i = 0$, but $\boldsymbol{t}_{d-1}$ is sampled from $(\mathbb{F} \setminus \{0\})^{n_{d-1}}$.

The case 1 above happens with probability at most $\frac{2}{|\mathbb{F}|-1}$ over the random choice of u_i, and then case 2 happens with probability at least $1 - \frac{2}{|\mathbb{F}|-1}$. Since all u_i for $i \notin S$ are chosen independently, we have that the Hamming weight $w_{\mathsf{Ham}}(\hat{\boldsymbol{c}}) = \sum_{i \notin S} w_{\mathsf{Ham}}(\hat{\boldsymbol{c}}[i], \hat{\boldsymbol{c}}[n_{d-1} + i])$ is distributed according to a sum of independently distributed Bernouilli variables X_i, with $\Pr[X_i = 1] \leq \frac{2}{|\mathbb{F}|-1}, \Pr[X_i = 2] = 1 - \Pr[X_i = 1] \geq \frac{2}{|\mathbb{F}|-1}$. To simplify, and since we are upper bounding the probability of $w_{\mathsf{Ham}}(\hat{\boldsymbol{c}})$ being smaller than some quantity, we define X'_i the Bernouilli variable with exact probabilities $\Pr[X'_i = 1] = \frac{2}{|\mathbb{F}|-1}$, $\Pr[X'_i = 2] = 1 - \frac{2}{|\mathbb{F}|-1}$ and we have

$$\Pr[w_H(\hat{\boldsymbol{c}}) \leq n_d - t] \leq \Pr[\sum_{i \notin S} X'_i \leq n_d - t.]$$

Moreover, we can write $X''_i = 2 - X'_i$ (hence $\Pr[X''_i = 1] = \frac{2}{|\mathbb{F}|-1}$, $\Pr[X''_i = 0] = 1 - \frac{2}{|\mathbb{F}|-1}$) and we have $\sum_{i \notin S} X'_i = 2(n_{d-1} - |S|) - \sum_{i \notin S} X''_i$, so $\sum_{i \notin S} X'_i \leq n_d - t$ if and only if $\sum_{i \notin S} X''_i \geq 2(n_{d-1} - |S|) - (n_d - t) = t - 2|S|$. Hence we want an upper bound for $\Pr[\sum_{i \notin S} X''_i \geq t - 2|S|]$. Now

$$\Pr[\sum_{i \notin S} X''_i \geq t - 2|S|] = \sum_{j=t-2|S|}^{n_{d-1}-|S|} \Pr[\sum_{i \notin S} X''_i = j] \tag{34}$$

$$= \sum_{j=t-2|S|}^{n_{d-1}-|S|} \binom{n_{d-1} - |S|}{j} \left(\frac{2}{|\mathbb{F}|-1}\right)^j \left(1 - \frac{2}{|\mathbb{F}|-1}\right)^{n_{d-1}-|S|-j} \tag{35}$$

We now note that for any positive integers $0 < a < b$, any real $0 < p < 1$,

$$\sum_{j=a}^{b} \binom{b}{j} p^j (1-p)^{b-j} \leq \sum_{j=a}^{b} 2^b p^j (1-p)^{b-j} = 2^b p^a \sum_{k=0}^{b-a} p^k (1-p)^{b-a-k} \leq 2^b p^a$$

since

$$\sum_{k=0}^{b-a} p^k (1-p)^{b-a-k} \leq \sum_{k=0}^{b-a} \binom{b-a}{k} p^k (1-p)^{b-a-k},$$

which is the binomial expansion of $(p + (1-p))^{b-a} = 1$

Therefore

$$\Pr[\sum_{i \notin S} X_i'' \geq t - 2|S|] \leq 2^{n_{d-1}-|S|} \left(\frac{2}{|\mathbb{F}|-1}\right)^{t-2|S|}$$

which gives the result.

In the following lemma again we can obtain further improvements by using the binomial theorem to bound some of the expressions of the proof (e.g. (41) and below)

Lemma 14. *[Improvement on Lemma 4 in [ZCF24]] Suppose* $d_{\mathsf{min}}(C_{d-1}) \leq n_{d-1} - t_{d-1}$. *Let* $t_d = 2t_{d-1} + \ell_d$ *where*

$$\ell_d = \frac{\lambda + 2t_{d-1}(1 + \log(\frac{|\mathbb{F}|}{|\mathbb{F}|-1})) + 0.585 n_d}{\log(|\mathbb{F}|-1) - 1}$$

then

$$\Pr[d_{\mathsf{min}}(C_d) \leq n_d - t_d] \leq 2^{-\lambda}$$

Proof. Note that every $\hat{\boldsymbol{c}} \in C_d$ is of the form $\hat{\boldsymbol{c}} = (\boldsymbol{c} + \boldsymbol{c'} \circ \boldsymbol{t}_{d-1}, \boldsymbol{c} - \boldsymbol{c'} \circ \boldsymbol{t}_{d-1})$ for a unique $(\boldsymbol{c}, \boldsymbol{c'}) \in C^{d-1}$. We use the shorthand $(\boldsymbol{c}, \boldsymbol{c'}) = F(\hat{\boldsymbol{c}})$ Moreover, $F(\hat{\boldsymbol{c}}) \in m_d(S)$ for a unique S.

Then

$$\Pr[d_{\mathsf{min}}(C_d) \leq n_d - t_d] \leq \sum_{\hat{\mathbf{c}} \in C_d \setminus \{0\}} \Pr[w_H(\hat{\mathbf{c}}) \leq n_d - t_d] \tag{36}$$

$$= \sum_{S \subsetneq [n_{d-1}]} \sum_{\substack{\hat{\mathbf{c}} \in C_d: \\ f(\hat{\mathbf{c}}) \in m_d(S)}} \Pr[w_H(\hat{\mathbf{c}}) \leq n_d - t_d] \tag{37}$$

$$\leq \sum_{S \subsetneq [n_{d-1}]} |m_d(S)| \cdot 2^{n_{d-1}-|S|} \left(\frac{2}{|\mathbb{F}|-1}\right)^{t_d - 2|S|} \tag{38}$$

$$\leq \sum_{j=0}^{t_{d-1}-1} |\mathbb{F}|^{2t_{d-1}-2j} \binom{n_{d-1}}{j} 2^{n_{d-1}-j} \left(\frac{2}{|\mathbb{F}|-1}\right)^{t_d-2j} \tag{39}$$

$$= \sum_{j=0}^{t_{d-1}-1} \left(\frac{|\mathbb{F}|}{|\mathbb{F}|-1}\right)^{2t_{d-1}-2j} \binom{n_{d-1}}{j} 2^{n_{d-1}+2t_{d-1}-3j} \left(\frac{2}{|\mathbb{F}|-1}\right)^{\ell_d}. \tag{40}$$

where we have used Lemma 13 for the inequality (37)≤(38), Lemma 12 for the inequality (38)≤(39) and $t_d = 2t_{d-1} + \ell_d$ and rearranging terms for the last equality

Now (40) can be rearranged as

$$2^{n_{d-1}+2t_{d-1}} \cdot \left(\frac{|\mathbb{F}|}{|\mathbb{F}|-1}\right)^{2t_{d-1}} \cdot \left(\frac{2}{|\mathbb{F}|-1}\right)^{\ell_d} \cdot \sum_{j=0}^{t_{d-1}-1} \binom{n_{d-1}}{j} \left(1 - \frac{1}{|\mathbb{F}|}\right)^{2j} 2^{-3j} \tag{41}$$

We focus on the sum, upper bounding $(1 - \frac{1}{|\mathbb{F}|})$ by 1, which we can write as $1^{n_{d-1}-j}$ by reasons that will become clear next. We get

$$\sum_{j=0}^{t_{d-1}-1} \binom{n_{d-1}}{j} \left(1 - \frac{1}{|\mathbb{F}|}\right)^{2j} 2^{-3j} \leq \sum_{j=0}^{n_{d-1}} \binom{n_{d-1}}{j} \cdot \left(\frac{1}{8}\right)^{j} 1^{n_{d-1}-j}$$

and we get a binomial sum in the right hand side which equals

$$\left(\frac{1}{8} + 1\right)^{n_{d-1}} = \left(\frac{9}{8}\right)^{n_{d-1}}$$

Coming back to the full expression in (41) (and merging the terms $2^{n_{d-1}}$ and the term $\left(\frac{9}{8}\right)^{n_{d-1}}$ we have obtained as bound for the sum), we get

$$\Pr[d_{\mathsf{min}}(C_d) \leq n_d - t_d] \leq 2^{2t_{d-1}} \cdot \left(\frac{|\mathbb{F}|}{|\mathbb{F}|-1}\right)^{2t_{d-1}} \cdot \left(\frac{2}{|\mathbb{F}|-1}\right)^{\ell_d} \cdot \left(\frac{9}{4}\right)^{n_{d-1}}$$

For this to be smaller than $2^{-\lambda}$ we need to set ℓ_d as

$$\ell_d \geq \frac{\lambda + 2t_{d-1}(1 + \log(\frac{|\mathbb{F}|}{|\mathbb{F}|-1}) + n_{d-1}\log(9/4)}{\log(\frac{|\mathbb{F}|-1}{2})}$$

Using that $\log(9/4) = 1.1699... \leq 1.17$ and that $n_{d-1} = n_d/2$, we have that it is enough to set

$$\ell_d = \frac{\lambda + 2t_{d-1}(1 + \log(\frac{|\mathbb{F}|}{|\mathbb{F}|-1})) + 0.585n_d}{\log(|\mathbb{F}| - 1) - 1}$$

The final bound can be now be obtained as in [ZCF24]

Proof (Proof of Theorem 12).

Induction. For $d = 0$ it comes from the assumption that C_0 is an MDS code. In this case $d_{min}(C_0) = n_0 - k_0 + 1 > n_0 - t_0$, so clearly the stated probability is 0.

If $d > 0$, we assume the property is true for every $i \leq d-1$, i.e. $\Pr[d_{min}(C_i) \leq n_i - t_i] \leq i \cdot 2^{-\lambda}$.

Let E_i be the event "$d_{min}(C_i) \leq n_i - t_i$". From the above we are assuming $Pr[E_i] \leq i \cdot 2^{-\lambda}$ for $i \leq d - 1$.

Now note

$$\begin{aligned}\Pr[E_d] &= \Pr[E_d|E_{d-1}] \cdot \Pr[E_{d-1}] + \Pr[E_d|\neg E_{d-1}] \cdot \Pr[\neg E_{d-1}] \\ &\leq \Pr[E_{d-1}] + \Pr[E_d|\neg E_{d-1}] \\ &\leq (d-1) \cdot 2^{-\lambda} + \Pr[E_d|\neg E_{d-1}]\end{aligned}$$

But $\Pr[E_d|\neg E_{d-1}]$ is precisely the probability analyzed in Lemma 14 and it is at most $2^{-\lambda}$. Hence $\Pr[E_d] \leq d \cdot 2^{-\lambda}$ and we are done.

Finally for $|\mathbb{F}| \geq 2^{11}$, $\log(|\mathbb{F}|) - \log(|\mathbb{F}| - 1) \leq 0.001$, which implies both an upper bound $\log(\frac{|\mathbb{F}|}{|\mathbb{F}|-1}) \leq 0.001$ for the expression in the enumerator and a lower bound $\log(|\mathbb{F}| - 1) - 1 \geq \log|\mathbb{F}| - 1.001$ for the one in the denominator.

I Implementation and Evaluation additional details

I.1 Neural Networks

We evaluate the efficiency of our method on a generic neural networks architecture. The aim of our implementation is to demonstrate the feasability of our approach, and not to optimise the machine learning aspect of our solution. For this reason, we do not focus on a particular dataset. We note however, that the neural network architecture that we use could be used, for example, for the income prediction dataset [K$^+$96] or the Wisconsin breast cancer [WMSS93] one.

We focus on a small neural network for a binary classification task, and make use of the following architecture.

Fully connected layer: this layer connects the input nodes to the output nodes. Here, we have 30 inputs for n_0 outputs;

Square activation: this is an activation layer; the function $f(x) = x^2$ is applied to all n_0 input nodes;

Fully connected layer: another fully connected layer, connecting n_0 inputs to n_1 outputs;

Square activation: this activation layer applies $f(x) = x^2$ to all n_1 inputs;

Fully connected layer: the final fully connected layer, taking n_1 inputs into 2 outputs, corresponding to the two classes.

CKKS constraints We use the following CKKS parameters for all our experiments. We let the ring degree $N = 2^{13}$, $L = 2$ and $\log_2(p_i) = 45$, for all the p_i in the modulus chain, so that our top modulus is the product of three primes $q = q_0 = p_0 p_1 p_2$. Given these parameters, we choose the splitting degree $d = 4$ to achieve soundness. Both the secret key and the error distributions are Gaussians of width $\sigma = 3.2$. The parameters achieve at least 128 bits of security, as estimated by the lattice estimator [APS15].

I.2 Additional experimental results

Table 3 shows the complete measurements for our range proofs. Column "Other" gathers the rest of the prover: the digit decomposition, the partial evaluations $\tilde{\psi}'_\eta$, the counters and the fingerprints of the lookup. The first-layer quotients, which are split into $c = 8$ chunks, are timed with the measurement for $c = 4$ at the same ℓ^*, i.e., the same number of chunks and the same table.

Table 3: Timing performance of our range proofs: the optimized variant $\hat{\Pi}^*_{range}$ over $\mathbb{F}_p$ (208-bit p, $(\sigma, m, \kappa) = (2^{11}, 15, 20)$), measured.

Model (n_0, n_1)	`SumCheck`	`Com`	`Eval`	Other	Total
$(128, 64)$	13s	3m47s	1m30s	24s	5m54s
$(64, 16)$	6s	1m47s	43s	11s	2m47s
$(32, 16)$	3s	52s	22s	6s	1m24s

(n_0, n_1)	`Xmult`	$\widetilde{\texttt{copycon}}$	$\tilde{V}_{3,det}$	$\tilde{V}_{3,copy}$	$\lceil \log(D) \rceil$	$\tilde{V}_{3,BD}$	$\tilde{V}_{3,quot}$	Non-zero		$\#\widetilde{\texttt{copycon}}'$ variables
								`Xmult`	$\widetilde{\texttt{copycon}}$	
$(128, 64)$	768	24320	12160	388	2	11	10	10	15	34
$(64, 16)$	320	5952	2976	164	2	10	9	9	13	30
$(32, 16)$	192	3008	1504	100	2	9	8	8	12	27

Table 4: Circuit sizes for the neural network architectures considered. The entries in the columns (n_0, n_1) represent the sizes of the hidden layers. The entries in the columns `Xmult` and $\widetilde{\texttt{copycon}}$ are reported in concrete values. The columns $\tilde{V}_{3,BD}$ and $\tilde{V}_{3,quot}$ report the number of PC variables needed to represent the wires as multilinear polynomials. Entries in the columns "Non-zero" are reported in $\log_2$ values.