Sailor: Sumcheck and GKR over $\mathbb{Z}_{p^k}$ via Galois Rings

Dani Vilardell* Yupeng Zhang†

Abstract

*The field of interactive proofs has seen significant developments in recent years, expanding the frontiers of secure computation and zero knowledge proofs. Sailor (**S**umcheck **A**nd GKR over $\mathbb{Z}_{p^k}$ v**I**a ga**LO**is **R**ings) introduces innovative adaptations of Sumcheck and GKR protocols over the ring structure $\mathbb{Z}/p^k\mathbb{Z}$. These protocols leverage Galois Rings and Reverse Multiplication Friendly Embeddings, providing efficient alternatives for real-world computations over $\mathbb{Z}_{2^k}$.*

Our work establishes a theoretical foundation for these protocols, even considering the edge case with p being 2. We offer a basic toolkit for ring-based cryptographic constructions, a new Sumcheck protocol over $\mathbb{Z}_{p^k}$ with minimal overhead and two versions of the GKR protocol over $\mathbb{Z}_{p^k}$. These advancements facilitate secure computations over rings.

Despite the exponential growth of zero knowledge proof protocols for secure computation, our paper addresses the gap in adapting these protocols to ring structures, presenting one of the first constructions in this domain.

1 Introduction

An interactive proof is a type of protocol that allows two parties, a prover and a verifier, to interact to establish the truth or correctness of a statement or a mathematical claim.

In recent years, the field of interactive proofs has witnessed remarkable advancements that have expanded the horizons of secure computation and zero knowledge proofs. This paper builds on these advancements by introducing adaptations of Sumcheck and GKR (Goldwasser-Kalai-Rothblum) protocols tailored to the ring structure $\mathbb{Z}/p^k\mathbb{Z}$. These adaptations leverage sophisticated mathematical constructs such as Galois Rings and Reverse Multiplication Friendly Embeddings (RMFE) to push the boundaries of what was previously attainable in cryptographic primitives.

As the models of computation in real-life programing and the computer architectures (such as CPU words) are formulated as operations over the ring $\mathbb{Z}_{2^{32}}$ or $\mathbb{Z}_{2^{64}}$, protocols over $\mathbb{Z}_{2^k}$ are more efficient when implemented. However the fact that half of the ring $\mathbb{Z}_{2^k}$ are zero divisors presents non-trivial technical difficulties. Although $\mathbb{Z}_{2^k}$-arithmetic can be emulated using prime moduli, this comes with an unavoidable overhead.

Our study provides a rigorous theoretical foundation for constructing these protocols, ensuring their security and correctness within the $\mathbb{Z}/p^k\mathbb{Z}$ ring structure. It's noteworthy that our approach accommodates the case when p equals 2, catering to the demand for hardware-friendly constructions, a crucial consideration in modern computing.

While the paper focuses on theoretical advancements and mathematical insights, it lays the groundwork for future practical implementations. Our work contributes to the broader landscape of secure computation and data protection by enabling the application of Sumcheck and GKR protocols in ring-based settings.

By combining mathematical rigor with practical relevance, this research underscores the importance of Galois Rings and Reverse Multiplication Friendly Embeddings (RMFE) in expanding the cryptographic toolkit, offering opportunities for secure computations and privacy-preserving applications in various domains.

1.1 Our contributions

In this paper we propose the first constructions over $\mathbb{Z}_{p^k}$ for several cryptographic primitives greatly used in the world of secure computation and Zero Knowledge Proofs. Both the Sumcheck and GKR protocol constructions have a small overhead factors in comparison to the classic protocols, and in the GKR case, maintain the succinctness of proof size and verification time in the size of the arithmetic circuit representing the statement C. Our work is based on the classic constructions presented by Lund et al. in [1] for Sumcheck and Chiesa et al. version of the GKR from [2], a protocol originally introduced by Goldwasser, Kalai and Rothblum in [3].

Basic toolkit for ring based cryptographic constructions: We propose and bring the the ring domain multiple tools widely used in the world of cryptography and ZKP such as the Schwartz Zipple

*UC Berkeley
†University or Illinois Urbana-Campaign

lemma or the idea of batching multiple claims by using a linear combinations. We bring more powerful tools, namely, the reverse multiplication friendly embedding (RMFE) techniques from the MPC literature [4, 5, 6, 7, 8] into the literature of secure computation and interactive proofs.

Sumcheck over $\mathbb{Z}_{p^k}$: This paper features a new version of the Sumcheck over any ring of the form $\mathbb{Z}_{p^k}$ with just a $O(\log n)$ overhead factor respect to the classic one, n being the security parameter of the protocol. It uses a new version of the Schwartz Zipple lemma over Galois Rings and, thanks to the fact that only simple tools are needed, it's easy to implement. The protocol can also be optimized the same way the classic is by using the ideas from the Libra paper [9].

Degree D Sumcheck over $\mathbb{Z}_{p^k}$: The fourth section proposes a different Sumcheck construction for the case when the summation is done with the product of D functions instead of just having one. It saves the prover and verifier from computing the product and works with each evaluation separately. Moreover, it also only has an overhead factor of $O(D \log d)$ and the ideas presented in the Libra paper [9] also work in this case. It's construction makes use of advanced mathematical tools such as RMFE making it harder to implement than the previous protocol.

GKR over $\mathbb{Z}_{p^k}$: Finally in sections 5 and 6 two protocols are presented as constructions for the GKR protocol over $\mathbb{Z}_{p^k}$. The first approach is the trivial approach and like the first Sumcheck, it's very easy to implement as it doesn't make use of the RMFE tool. It has an overhead factor in comparison to the classic GKR of $O(n \log n)$. By making use of RMFE and a more complex protocol, we can bring this overhead complexity to $O(n)$, and this is what is don in section 6. Once again, the Libra construction can also be adapted in our case and helps reduce greatly the overall protocol complexity.

1.2 Related Work

It's worth noting that while the field of ZKP has exponentially grown these past few years, very few research in bringing these new protocols over more general structures such as rings has been made. To our knowledge, these are all existing works that construct ZK protocols over rings.

Shuo Chen et al. in [10] extend the GKR and Sumcheck construction to finite commutative rings as long as the points used to define the polynomial extension and the random challenges from the verifier belong to a sampling set as defined in [11]. The security parameter ends up being inversely proportional to this set's cardinality, which in the case of working over the ring $\mathbb{Z}_{p^k}$ ends up being $p-1$ making this protocol unusable with small prime values. Nonetheless, their proposed approach does not have any overhead in comparison to the classic one, as they prove that the trivial lifting to rings works fine as long as p is large enough. We solve this issue making our protocol security not depend on the ring being used as long as the ring is of the form $\mathbb{Z}_{p^k}$ by relying in the Schwartz Zipple lemma over Galois Rings instead of the one in $\mathbb{Z}_{p^k}$ and making it possible to work over $\mathbb{Z}_{2^k}$.

A more recent construction following the previous paper presented by Eduardo Soria-Vazquez in [12] generalizes the construction over infinite and non-commutativer rings. Even though this construction expands the structures where GKR and Sumcheck are complete, it still does not provide soundness over rings of small characteristic, in particular $\mathbb{Z}_{2^k}$. It is worth mentioning that, the same way it is done in 6, both papers achieve a quasi linear or linear prover time by using the tools provided in Libra [9].

The same Sumcheck over finite rings is also provided by Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki in[2] to later be used in Sumcheck Arguments and solving the R1CS problem over Rings. We still face the same problems when it commes to small characteristic Rings.

These next papers don't work on GKR and Sumcheck but instead propose a general SNARK construction for rings $\mathbb{Z}_{2^k}$.

Ganesh et al. [13] proposed Rinocchio, which adapted the Pinnocchio [14] system (a SNARK designed for field arithmetic) to work with ring arithmetic. Nonetheless, when used in $\mathbb{Z}_{2^k}$ Rinnocchio faces significant efficiency loss, as exceptional sets of $\mathbb{Z}_{2^k}$ are very small. While our paper does not tackle the same problem, its efficiency and security don't depend on the chosen prime value.

Similarly, building on the blueprint of VOLE-based zero-knowledge protocols designed for finite fields and inspired by the concept behind SPD$\mathbb{Z}_{2^k}$ [15], Baum and their team proposed two online constructions in "Appenzeller to Brie" [16]. Subsequently, they developed a more efficient online protocol using a PCG construction for VOLE over $\mathbb{Z}_{2^k}$ in Moz$\mathbb{Z}_{2^k}$arella [17]. However, a notable limitation of these constructions is their inherent dependency on the security parameter.

A more recent construction of a designated-verifier zero-knowledge proof scheme presented by Lin, Xing and Yao in [18] removes the undesirable dependence of communication complexity on the security parameter, and achieve (strict) linear communication complexity.

While all these constructions revolve around zero knowledge primitives, no advancement in the field

of interactive proofs for secure computation in rings has been made. As far as we are aware, we propose the first constructions for such protocols.

2 Preliminaries

2.1 Notations

We use bold letters (e.g. $\boldsymbol{x}$, $\boldsymbol{y}$) to denote vectors, and we use the star operator ($\star$) to denote component-wise product of vectors.

During all the paper p will be a prime number. We will also refer to the ring $\mathbb{Z}/p^k\mathbb{Z}$ as $\mathbb{Z}_{p^k}$ for simplicity purposes.

We will define $[n] := \{1, \cdots, n\}$ and $P(S)$ to be the power set of S, for any set S. The power set is the set of all subsets of a set.

2.2 Algebraic Preliminaries

Trying to build cryptographic primitives over rings is a big challenge since a lot of assumptions and theorems underlying the security and soundness are lost when not working in fields. The first problem we will tackle is the construction of a ring version Schwartz Zipple lemma.

Lemma 2.1. *A polynomial of degree c over $\mathbb{Z}_{p^k}$ can have up to rp^{k-1} roots.*

For instance, the polynomial of degree 1 $f(x) = p^{k-1}x$ has p^{k-1} roots.

That lemma gives the following upper-bound for the Schwartz Zipple lemma for any $f \in \mathbb{Z}_{p^k}^{\leq r}[X]$

$$\begin{aligned} Pr[f(r) = 0 | f \neq 0, r \xleftarrow{\$} \mathbb{Z}_{p^k}] \leq \\ \leq c \cdot p^{k-1}/p^k = c/p \end{aligned}$$

which is clearly non-negligible.

To solve this problem we will work on rings with a better upper-bounds on polynomial roots known as Galois Rings [19].

Definition 2.2. *We denote by $GR(p^k, d)$ the Galois Ring over $\mathbb{Z}_{p^k}$ of degree d, which is a ring extension $\mathbb{Z}_{p^k}/(f(X)) = \mathbb{Z}_{p^k}[\xi]$, where $f(X) \in \mathbb{Z}_{p^k}[X]$ is a monic polynomial of degree d over $\mathbb{Z}_{p^k}$ whose reduction modulo p is irreducible over $\mathbb{Z}_p$ and ξ is a root of f in the Galois Ring.*

We can view $GR(p^k, d)$ as a $\mathbb{Z}_{p^k}$-module [20] with basis $\{1, \xi, \xi^2, \cdots, \xi^{d-1}\}$. Therefore, any value $x \in GR(p^k, d)$ can be written as

$$x = \sum_{i=0}^{d-1} a_i \xi^i \quad a_i \in \mathbb{Z}_{p^k}$$

Within this ring we have a better upper-bound on number of polynomial roots.

Lemma 2.3. *([LXY23, lemma 1]) A polynomial of degree c over $GR(p^k, d)$ can have up to $cp^{(k-1)d}$ roots.*

This will enable us to construct a Schwartz Zipple lemma over $GR(p^k, d)$.

Corollary 2.3.1. *For any polynomial $f \in GR(p^k, d)^{\leq c}[X]$ we have*

$$\begin{aligned} Pr[f(r) = 0 | f \neq 0, r \xleftarrow{\$} GR(p^k, d)] \leq \\ \leq c \cdot p^{(k-1)d}/p^{dk} = c/p^d \end{aligned} \tag{1}$$

Corollary 2.3.2 (Schwartz Zipple lemma)**.** *For any polynomials $f, g \in GR(p^k, d)^{\leq c}[X]$ we have*

$$\begin{aligned} Pr[f(r) = g(r) | f \neq g, r \xleftarrow{\$} GR(p^k, d)] = \\ = Pr[(f-g)(r) | f - g \neq 0, r \xleftarrow{\$} GR(p^k, d)] \leq c/p^d \end{aligned}$$

We see here that this probability can be made negligible by increasing the extension degree. That will give us the soundness of all protocols over $\mathbb{Z}_{p^k}$ presented in the following sections.

Since our main goal will be to build protocols over $\mathbb{Z}_{p^k}$ we will now need a tool to bring all computations from $\mathbb{Z}_{p^k}$ to the Galois Ring. For the Sumcheck protocol case we will be using what can be called as the trivial mapping.

Definition 2.4. *We will define the trivial bijective mapping as*

$$\begin{aligned} \Phi : \mathbb{Z}_{p^k}^n \rightarrow & GR(p^k, n) \\ \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} \mapsto & \sum_{i=0}^{n-1} x_i \xi^i \end{aligned}$$

For any $\boldsymbol{x} \in \mathbb{Z}_{p^k}^n$, we will define $\overline{x} := \Phi(x)$.

Consider f to be a multivariate polynomial vector (or vector of multivariate polynomials) with the polynomials in $\mathbb{Z}_{p^k}^{\leq c}[X_1, \cdots, X_l]$. That can also be viewed as a polynomial whose coefficients are in a vector space and the evaluates in $\mathbb{Z}_{p^k}$. Let c be the maximum degree of the polynomials, then we can represent f as

$$f(x_1, x_2 \cdots, x_l) = \sum_{i_1 + i_2 + \cdots + i_l \leq c} a_{i_1, i_2, \cdots, i_l} x_1^{i_1} x_2^{i_2} \cdots x_l^{i_l}$$

where $a_{i_1, i_2, \cdots, i_l} \in \mathbb{Z}_{p^k}^n$ and $x_i \in \mathbb{Z}_{p^k}$.

We will define $\overline{f} := \Phi(f) \in GR(p^k, d)^{\leq c}[X_1, \cdots, X_l]$ for any polynomial vector f as

$$\Phi(f)(x_1, \cdots, x_l) := \sum_{i_1+\cdots+i_l \leq c} \Phi(a_{i_1,\cdots,i_l}) x_1^{i_1} \cdots x_l^{i_l}$$

The trivial mapping will be useful when presenting the basic Sumcheck protocol construction, but once we try to build the GKR protocol, we will need a stronger mapping from $\mathbb{Z}_{p^k}^n$ to $GR(p^k, d)$.

We define the ring $(\mathbb{Z}_{p^k}^n, +, \star)$ where $\star$ is the component wise product of vectors.

$$\boldsymbol{x_1} \star \boldsymbol{x_2} = \begin{pmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,n} \end{pmatrix} \star \begin{pmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,n} \end{pmatrix} = \begin{pmatrix} x_{1,1}x_{2,1} \\ x_{1,2}x_{2,1} \\ \vdots \\ x_{1,n}x_{2,n} \end{pmatrix}$$

Definition 2.5. *Let $d, n \in \mathbb{N}$, a pair of $\mathbb{Z}_{p^k}$-module homomorphisms $\phi : \mathbb{Z}_{p^k}^n \to GR(p^k, d)$ and $\psi : GR(p^k, d) \to \mathbb{Z}_{p^k}^n$ is a degree-D Reverse Multiplication-Friendly Embedding (RMFE) if for all $\boldsymbol{x_1}, \cdots, \boldsymbol{x_D} \in (\mathbb{Z}_{p^k}^n, +, \star)$ it holds that $\prod_{i=1}^D \boldsymbol{x_i} = \psi(\prod_{i=1}^D \phi(\boldsymbol{x_i}))$.*

It's construction can be found in [21].

Some important direct consequences of this definition are presented in the following lemma.

Lemma 2.6. *ϕ is injective and ψ is surjective.*

Proof. To see that ϕ is injective it suffices to show that $\phi(\boldsymbol{a}) = \boldsymbol{0}$ implies that $\boldsymbol{a} = 0$. Indeed, if $\phi(\boldsymbol{a}) = 0$ then $\boldsymbol{0} = \psi(0) = \psi(\phi(\boldsymbol{a}) \star \phi(\boldsymbol{1}) \star \cdots \star \phi(\boldsymbol{1})) = \boldsymbol{a} \star \boldsymbol{1} \star \cdots \star \boldsymbol{1} = \boldsymbol{a}$. Similarly, given $\boldsymbol{a} \in \mathbb{Z}_{p^k}^n$, the preimage of $\boldsymbol{a}$ is given by $\phi(\boldsymbol{a}) \cdot \phi(\boldsymbol{1}) \cdots \phi(\boldsymbol{1})$, which shows that ψ is surjective. □

Lemma 2.7. *In a RMFE pair (ϕ, ψ), both functions are linear.*

Proof. Due to the fact that ϕ is a group homomorphism, we have $\phi(h\boldsymbol{a}) = \phi(\sum_{i=1}^h \boldsymbol{a}) = \sum_{i=0}^h \phi(\boldsymbol{a}) = h\phi(\boldsymbol{a})$. The same argument can be applied to ψ as well. □

Lemma 2.8. *([ELXY23, lemma 2]) Let (ϕ, ψ) be a degree-D RMFE pair. Then there exists a degree-D RMFE pair (ϕ', ψ') with $\phi'(\boldsymbol{1}) = 1$.*

Throughout the paper we will use such RMFE.

We will also define $\hat{f} = \phi(f) \in GR^{\leq c}(p^k, d)[X_1, \cdots, X_l]$ for any polynomial vector f the same way we did $\overline{f}$ with Φ:

$$\phi(f)(x_1, \cdots, x_l) := \sum_{i_1+\cdots+i_l \leq c} \phi(a_{i_1,\cdots,i_l}) x_1^{i_1} \cdots x_l^{i_l}$$

2.3 Sumcheck protocol

The Sumcheck problem is a fundamental problem that has various applications. The problem is to sum a polynomial $f \in \mathbb{F}[X_1, \cdots, X_l]$ over the binary hypercube.

$$H := \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_l \in \{0,1\}} f(b_1, b_2, \cdots, b_l) \quad (2)$$

Calculating this sum directly has an exponential complexity with the dimension l, given that there are 2^l possible combinations for $b_1, b_2, ..., b_l$.

To address this issue, Lund et al. introduced the Sumcheck protocol, described in Protocol 1, which enables a verifier V to delegate the computation to a computationally unbounded prover P. The prover can then convince the verifier V that the result H is the correct sum. The proof size for the Sumcheck protocol is $O(cl)$, where c represents the variable degree of f. In each round, P transmits a univariate polynomial of a single variable from f, and this can be uniquely determined by $c+1$ points. The verifier's time complexity in this protocol is $O(cl)$.

The prover's time complexity is $O(l2^l T)$, where $O(T)$ is the complexity to evaluate f at a random point. It's important to note that the Sumcheck protocol is both complete and sound, with a soundness error of $\frac{cl}{|\mathbb{F}|}$.

We will define the polynomial vector version of the equality as

$$\begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_n \end{pmatrix} = \sum_{b_{\log n+1}, \cdots, b_l \in \{0,1\}} \tilde{f}(b_{\log n+1}, \cdots, b_l) =$$

$$= \sum_{b_{\log n+1}, \cdots, b_l \in \{0,1\}} \begin{pmatrix} f(0, \cdots, 0, 0, b_{\log n+1}, \cdots, b_l) \\ f(0, \cdots, 0, 1, b_{\log n+1}, \cdots, b_l) \\ \vdots \\ f(1, \cdots, 1, 1, b_{\log n+1}, \cdots, b_l) \end{pmatrix} \quad (3)$$

where $\tilde{f}$ is a polynomial vector. If the prover is able to prove the previous equality, then the verifier will get $H = \sum_{i=1}^n H_i$. We will refer to this transformation the batching of the sumcheck, and the first $\log n$ variables of f will be the batching parameters.

2.4 GKR protocol

The GKR protocol is an interactive proof protocol for layered arithmetic circuits proposed by Goldwasser et al. that uses the Sumcheck protocol as a building block.

Protocol 1 (Sumcheck). *The protocol proceeds in l rounds*

- *In the first round, P sends a univariate polynomial*

$$f_1(x_1) := \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_l \in \{0,1\}} f(x_1, b_2, \cdots, b_l)$$

  V checks that $H = f_1(0) + f_1(1)$ and sends a random challenge $r_1 \in \mathbb{F}$ to P.
- *In the i-th round, P sends a univariate polynomial*

$$f_i(x_i) := \sum_{b_{i+1} \in \{0,1\}} \cdots \sum_{b_l \in \{0,1\}} f(r_1, r_2, \cdots, r_{i-1}, x_i, b_{i+1}, \cdots, b_l)$$

  V checks that $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$ and sends a random challenge $r_i \in \mathbb{F}$ to P.
- *In the l-th round P will send a univariate polynomial*

$$f_l(x_l) := f(r_1, \cdots, r_{l-1}, x_l)$$

  V will check that $f_{l-1}(r_{l-1}) = f_l(0) + f_l(1)$ and generates a random challenge $r_l \in \mathbb{F}$. Given oracle access to an evaluation $f(r_1, r_2, \cdots, r_l)$, V will verify that $f_l(r_l) = f(r_1, r_2, \cdots, r_l)$ and accept if the equality holds. The instantiation of the oracle depends on the application of the Sumcheck protocol

Protocol 2 (GKR protocol). *Let $\mathbb{F}$ be a prime field. Let $C : \mathbb{F}^{S_i} \to \mathbb{F}^{S_o}$ be a m-depth layered arithmetic circuit. Without loss of generality assume S_i and S_o are both powers of 2 and we can pad them if not. The protocol proceeds as follows.*

- *Define the multilinear extension of the output array as $\check{W}_o$. V chooses a random challenge $r_o \in \mathbb{F}^{s_o}$ and sends it to P. Both P and V compute $\check{W}_o(r_o)$.*
- *In the first round P and V run a sumcheck protocol on*

$$\check{W}_o(r_o) := \sum_{b,c \in \{0,1\}^{s_o}} \check{add}_o(r_o, b, c)(\check{W}_1(b) + \check{W}_1(c)) + \check{mult}_o(r_o, b, c)(\check{W}_1(b) \cdot \check{W}_1(c))$$

  At the end of the protocol V has to evaluate the whole polynomial. To do so, V receives a univariate polynomial $q(x)$ claimed to be equal to $W_1(l_1(x))$, where $l(x)$ is a linear function with $l(0) = r_b$ and $l_1(1) = r_c$. The verifier will now compute $W_1(r_b) = q_1(0)$, $W_1(r_c) = q_1(1)$, $\check{add}_i(r_o, r_b, r_c)$ and $\check{mult}_i(r_0, r_b, r_c)$ and then checks the last sumcheck protocol equality. V will now choose a new challenge r'_1 and send it to the prover. We will consider $r_1 = l_1(r'_1)$.
- *On the i'th round P and V will run a sumcheck on*

$$\check{W}_i(r_i) := \sum_{b,c \in \{0,1\}^{s_i}} \check{add}_i(r_i, b, c)(\check{W}_{i+1}(b) + \check{W}_{i+1}(c)) + \check{mult}_i(r_i, b, c)(\check{W}_{i+1}(b) \cdot \check{W}_{i+1}(c))$$

  and run the last step as it was done in the first iteration, resulting in a new claimed equality $\check{W}_{i+1}(l(x)) = q_{i+1}(x)$.
- *During the last iteration m the verifier will construct $\check{W}_m$ the multilinear extension of the input array and check that $W_m(l_m(x)) = q_m(x)$ and accept if this is true, reject otherwise.*

The high level idea is that given a layered arithmetic circuit C over a finite field $\mathbb{F}$, the prover will reduce the claimed output of the i'th layer to a claimed output on the previous layer of C. At the end of the protocol the verifier will have a claim about the input of the protocol that can be checked easily.

To do this we will to write the output of the i'th layer as the sum of the output of the $i + 1$'th layer. This will be done using a tool called multilinear extension.

Definition 2.9. *Let $V : \{0,d\}^s \to \mathbb{F}$ be a function. The multilinear extension of V is the unique multilinear polynomial $\check{V} : \mathbb{F}^s \to \mathbb{F}$ such that $\check{V}(x_1, \cdots, x_s) = V(x_1, \cdots, x_s) \quad \forall x_1, \cdots, x_s \in \{0,1\}^s$*

$\check{V}$ can be expressed using the Lagrange polynomial basis as

$$\check{V}(\boldsymbol{x}) = \sum_{b \in \{0,1\}^s} \prod_{i=1}^{s} ((1-x_i)(1-b_i) + x_i b_i) \cdot V(b)$$

Before describing the GKR protocol, we introduce some additional notations. We denote the number of gates in the i'th layer as S_i and let $s_i = \lceil \log S_i \rceil$ (we will assume S_i to be a power of 2 for simplicity purposes). Consider now $W_i : \{0,1\}^{s_i} \to \mathbb{F}$ to be the function that takes as input the label of a gate in the i'th layer in it's binary form and outputs the output of that gate. We also define two additional functions $add_i, mult_i : \{0,1\}^{s_{i-1}+2s_i} \to \{0,1\}$ referred as *wiring predicates* in the literature. add_i ($mult_i$) takes one gate label $z \in \{0,1\}^{s_{i-1}}$ in layer $i-1$ and outputs 1 if and only if gate z is an addition (multiplication) gate that takes the output of gate x, y as input, where x and y are the other two input parameters to the function.

With these new definitions, we have the following layer equality:

$$W_i(z) = \sum_{b,c \in \{0,1\}^{s_i}} add_i(z,b,c)(W_{i+1}(b) + W_{i+1}(c)) + mult_i(z,b,c)(W_{i+1}(b) \cdot W_{i+1}(c)) \quad \forall z \in \{0,1\}^{s_i}$$

We can rewrite the equation in its multilinear extension version

$$\check{W}_i(z) := \sum_{b,c \in \{0,1\}^{s_i}} \check{add}_i(z,b,c)(\check{W}_{i+1}(b) + \check{W}_{i+1}(c)) + \check{mult}_i(z,b,c)(\check{W}_{i+1}(b) \cdot \check{W}_{i+1}(c))$$

where $z \in \mathbb{F}^{s_i}$.

The whole protocol description can be found in Protocol 2. The main idea is to use sumcheck protocol to prove the above equality in the first layer at a random point r_0. That is to prove

$$\check{W}_0(r_0) := \sum_{b,c \in \{0,1\}^{s_0}} \check{add}_0(r_0,b,c)(\check{W}_1(b) + \check{W}_1(c)) + \check{mult}_0(r_0,b,c)(\check{W}_1(b) \cdot \check{W}_1(c))$$

At the last step of the sumcheck the verifier would have to evaluate the polynomial the polynomial at the random point r_b, r_c, which implies he would have to compute $\check{W}_1(r_b)$ and $\check{W}_1(r_c)$. This is not possible as V doesn't have access to the computation trace of C.

To solve this, the prover will provide a univariate polynomial $q_1(x)$ claimed to be equal to $W_1(l_1(x))$ where l_1 is the linear polynomial such that $l_1(0) = r_b$ and $l_1(1) = r_c$. The verifier will be able to compute $\check{W}_1(r_b) = q_1(0)$ and $\check{W}_1(r_c) = q_1(1)$ as long as $q_1(x) = \check{W}_1(l_1(x))$. The verifier will check the last step of the sumcheck and if the equality holds, will proceed to prove that the polynomial equality holds using Schwartz Zipple lemma. V will pick a random challenge r_1' and define $r_1 = l(r_1')$ and the prover will have to prove that $q_1(r_1') = \check{W}_1(r_1)$.

This will be done recursively the same way we reduced the claim on the output to a claim on the last layer. Once the protocol reaches the input layer, the verifier will be able to compute $\check{W}_m$ and check that $\check{W}_m \circ l_m = q_m$.

3 Sumcheck over $\mathbb{Z}_{p^k}$

Throughout this section we will build a protocol that will enable P to prove to V the correctness of the following sum

$$H = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_l \in \{0,1\}} f(b_1, \cdots, b_l) \tag{4}$$

for any $f \in \mathbb{Z}_{p^k}^{\leq c}[X_1, \cdots, X_l]$ known to both P and V.

Naively computing the right hand equation would take the verifier an exponential amount of evaluations of f, whereas this protocol will enable V to delegate the computation to a computationally unbounded prover P.

Since the upper-bound for the Schwartz Zipple lemma is non-negligible in $\mathbb{Z}_{p^k}$, the classic Sumcheck protocol won't work. To solve this issue we will be bringing the computation over to the Galois Ring, and that will add a logarithmic overhead factor on the prover side in comparison to the classic version.

3.1 Protocol construction

We will split the protocol in two main steps, the first one being the Setup phase and the second the interactive phase. The setup phase will bring the sumcheck equation from $\mathbb{Z}_{p^k}$ to $GR(p^k, n)$, and the interactive phase will run the classic sumcheck protocol over this new ring.

The soundness of the protocol will come from the new version of Schwartz Zipple lemma presented in Corollary 2.3.2 and the completeness will come by construction.

Step 1 - Setup phase During this phase we will bring the summation from $\mathbb{Z}_{p^k}$ to $GR(p^k, n)$. This could be done trivially by inclusion since $\mathbb{Z}_{p^k} \subseteq$

Protocol 3 (Sumcheck over $\mathbb{Z}_{p^k}$)**.** *The protocol proceeds in 2 main steps*

Step 1 - Setup phase

- *P will batch the first* $\log d$ *summations together reducing the statement to proving*

$$\begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_n \end{pmatrix} = \sum_{b_{\log n+1}\in\{0,1\}} \cdots \sum_{b_l\in\{0,1\}} \begin{pmatrix} f(0,\cdots,0,0,b_{\log n+1},\cdots,b_l) \\ f(0,\cdots,0,1,b_{\log n+1},\cdots,b_l) \\ \vdots \\ f(1,\cdots,1,1,b_{\log n+1},\cdots,b_l) \end{pmatrix} = \sum_{b_{\log n+1},\cdots,b_l\in\{0,1\}} \tilde{f}(b_{\log n+1},\cdots,b_l)$$

  with $\tilde{f} : \mathbb{Z}_{p^k}^{l-\log n} \to \mathbb{Z}_{p^k}^n$ *a polynomial vector. It can be easily seen that* $H = \sum_{i=1}^n H_i$.
- *Both V and P will compute* $\overline{f} = \Phi(\tilde{f}) \in GR(p^k,n)[X_{\log n-1},\cdots,X_l]$ *for* $\Phi : \mathbb{Z}_{p^k}^n \to GR(p^k,n)$.

Step 2 - Interactive phase

- *Consider* $l' = l - \log n$*. In the first round, P sends a univariate polynomial*

$$\overline{g}_1(x_1) := \sum_{b_2\in\{0,1\}} \cdots \sum_{b_{l'}\in\{0,1\}} \overline{f}(x_1,b_2,\cdots,b_{l'})$$

  V checks that $H = \overline{g}_1(0) + \overline{g}_1(1)$ *and sends a random challenge* $r_1 \in GR(p^k,n)$ *to P.*
- *In the i-th round, P sends a univariate polynomial*

$$\overline{g}_i(x_i) := \sum_{b_{i+1}\in\{0,1\}} \cdots \sum_{b_{l'}\in\{0,1\}} \overline{f}(r_1,r_2,\cdots,r_{i-1},x_i,b_{i+1},\cdots,b_{l'})$$

  V checks that $\overline{g}_{i-1}(r_{i-1}) = \overline{g}_i(0) + \overline{g}_i(1)$ *and sends a random challenge* $r_i \in GR(p^k,n)$ *to P.*
- *In the* l'*-th round P will send a univariate polynomial*

$$\overline{g}_{l'}(x_{l'}) := \overline{f}(r_1,\cdots,r_{l'-1},x_{l'})$$

  V will check that $\overline{g}_{l'-1}(r_{l'-1}) = g_l(0) + g_{l'}(1)$ *and generates a random challenge* $r_{l'} \in \mathbb{F}$*. V will now evaluate* $\overline{f}(r_1,r_2,\cdots,r_{l'})$ *and will verify that* $\overline{g}_{l'}(r_{l'}) = \overline{f}(r_1,r_2,\cdots,r_{l'})$ *and accept if the equality holds or reject otherwise.*

$GR(p^k,n)$, nonetheless this would bring a big overhead to the overall computation. To solve this problem we will batch the first $\log n$ summations together doing n Sumchecks in parallel the same way it was presented in the preliminaries.

Let's say we have

$$H = \sum_{b_1\in\{0,1\}} \cdots \sum_{b_l\in\{0,1\}} f(b_1,\cdots,b_l)$$

Then, proving the previous equality is the same as proving the following two equalities

$$\begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_n \end{pmatrix} = \sum_{b_{\log n+1},\cdots,b_l\in\{0,1\}} \begin{pmatrix} f(0,\cdots,0,0,b_{\log n+1},\cdots,b_l) \\ f(0,\cdots,0,1,b_{\log n+1},\cdots,b_l) \\ \vdots \\ f(1,\cdots,1,1,b_{\log n+1},\cdots,b_l) \end{pmatrix} \tag{5}$$

$$H = \sum_{i=1}^n H_i$$

We can now write 5 as a summation of evaluations of a polynomial vector

$$\boldsymbol{H} = \sum_{b_{\log n+1},\cdots,b_l\in\{0,1\}} \tilde{f}(b_{\log n+1},\cdots,b_l)$$

Which we can now turn into a summation in the Galois Ring using Φ. We will end up with the following equality to prove

$$\begin{aligned} \overline{H} = \Phi(\boldsymbol{H}) &= \sum_{b_{\log n+1},\cdots,b_l\in\{0,1\}} \Phi(\tilde{f}(b_{\log n+1},\cdots,b_l)) \\ &= \sum_{b_{\log n+1},\cdots,b_l\in\{0,1\}} \overline{f}(b_{\log n+1},\cdots,b_l) \end{aligned} \tag{6}$$

with $\overline{H} \in GR(p^k,n)$ and $\overline{f} \in GR(p^k,n)[X_{\log n+1},\cdots,X_l]$. Since Φ is a linear

bijective mapping, proving that 5 holds is equivalent that proving that 6 holds.

The computation of $\overline{f}$ from f will be done by both P and V and it's necessary to ensure soundness of the protocol. If P where to compute $\overline{f}$ and send it to V, then P would be able to generate a new function $\overline{g} \neq \overline{f}$ whose summation is a different value H' and send $\overline{g}$ instead of $\overline{f}$ and the verifier would still accept the second part of the protocol as long as this new function $\overline{g}$ adds to this new value $H' \neq H$.

Step 2 - Interactive phase The prover P now wants to prove the following equality correctness to the verifier

$$\overline{H} = \sum_{b_{\log n+1}, \cdots, b_l \in \{0,1\}} \overline{f}(b_{\log n+1}, \cdots, b_l)$$

with $\overline{H} \in GR(p^k, n)$ and $\overline{f} \in GR(p^k, n)[X_{\log n+1}, \cdots, X_l]$.

For simplicity purposes we will consider $l' = l - \log n$ and work with the following equation:

$$\overline{H} = \sum_{b_1, \cdots, b_{l'} \in \{0,1\}} \overline{f}(b_1, \cdots, b_{l'})$$

This equation looks like the Sumcheck equation, but now on $GR(p^k, n)$. If we apply the classic Sumcheck protocol over this new ring, we will get a soundness error of $\delta_s \leq l/p^d$. This will prove to V the correctness of the sum and therefore the correctness of 4

The complete protocol can be found in Protocol 3.

We will now see the completeness and soundness analysis, as well as the complexity analysis of this protocol and compute the overhead of this new version in comparison to the classic Sumcheck over fields.

3.2 Completeness and soundness analysis

Lemma 3.1. *Given $f \in \mathbb{Z}_{p^k}[X_1, \cdots, X_l]$, then the following two are equivalent:*

1. $H = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_l \in \{0,1\}} f(b_1, \cdots, b_l)$
2. $\overline{H} = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{l'} \in \{0,1\}} \overline{f}(b_1, \cdots, b_{l'})$

Proof. Consider the batched form to be

$$\boldsymbol{H} = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{l'} \in \{0,1\}} \tilde{f}(b_1, \cdots, b_l)$$

This previous form is equivalent to the original one. Since Φ is a linear mapping, we have

$$\begin{aligned}
\Phi(\boldsymbol{H}) = \overline{H} &= \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{l'} \in \{0,1\}} \overline{f}(b_1, \cdots, b_{l'}) \\
&= \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{l'} \in \{0,1\}} \sum_{S \in P([l'])} \Phi(\tilde{f}_S) \prod_{i \in S} b_i \\
&= \Phi\Big(\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} \sum_{S \in P([l])} \tilde{f}_S \prod_{i \in S} b_i \Big) \\
&= \Phi\Big(\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} \tilde{f}(b_1, \cdots, b_l) \Big)
\end{aligned}$$

Since Φ is a bijective mapping, we have that this equality holds if and only if

$$\boldsymbol{H} = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} \tilde{f}(b_1, \cdots, b_l)$$

□

This lemma ensures that as long as P can prove the correctness of sum 2. in the previous lemma, then V can trust that the first sum is H.

During the protocol, the Setup phase was the conversion from the first sum to the second. The second phase nonetheless looks exactly like the original sumcheck protocol, but working on $GR(p^k, n)$ instead of $\mathbb{F}$. This next lemma will give us the soundness and correctness of the Interactive phase.

Lemma 3.2. *Let f be a n-variate polynomial defined over the ring $GR(p^k, n)$. For any $H \in GR(p^k, n)$, let $\mathcal{L}$ be the language of polynomials $\overline{f}$ such that*

$$H = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_l \in \{0,1\}} \overline{f}(b_1, \cdots, b_l)$$

The Sumcheck protocol used in the Interactive phase of Protocol 2 is an IOP system for $\mathcal{L}$ with completeness error $\delta_c = 0$ and soundness error $\delta_s \leq l/p^n$.

Proof. Completeness holds by construction. We will prove soundness by induction on l.

In the case $l = 1$, P sends $h_1(x_1)$ which should be equal to $\overline{f}_1(x_1)$. In order to check that, we evaluate both polynomials in a random point, and thanks to Corollary 2.3.2 (Schwartz Zipple lemma over $GR(p^k, n)$) we have that if $h_1(x_1) \neq \overline{f}(x_1)$ then with probability $1 - \frac{1}{p^n}$, V will reject, giving us a soundness error of $1/p^n$.

We will now assume the induction hypothesis that for $(l-1)$-variate polynomials the soundness error is $(l-1)/p^n$. Consider now $h_1(x_1) = \sum_{b_2 \in 0,1} \cdots \sum_{b_l \in 0,1} f(x_1, b_2, \cdots, b_l)$. Suppose now that P sends $f_1(x_1) \neq h_1(x_1)$. Then because they are distinct polynomials $h_1(r_1) \neq f_1(r_1)$ with probability at least $1 - p^{-n}$. Conditioned to this event happening, P now has to prove the equality $f_1(r_1) = \sum_{b_2 \in 0,1} \cdots \sum_{b_l \in 0,1} f(r_1, b_2, \cdots, b_l)$, which by the induction hypothesis we know that V will reject at

some subsequent round with probability at least $1-(n-1)/p^n$. Therefore V will reject with probability at least

$$1 - Pr[h_1(r_1) = f_1(r_1)] - (1-$$
$$Pr[V \text{ rejects in some round } j > 1 | h_1(r_1) \neq f_1(r_1)]) \geq$$
$$\geq 1 - 1/p^n - (l-1)/p^n = 1 - l/p^n$$

And therefore we end up with a soundness error of l/p^d. □

Theorem 3.3. *Protocol* 3 *is complete and has a soundness error* $\delta_s \leq l/p^d$.

Proof. Given the fact that V has access to the polynomial f and can compute the transformation $\overline{f}$, all the soundness error will come from the interactive step of the protocol. As seen in Lemma 3.2, this soundness error, and therefore the whole protocols soundness error, is $\delta_s \leq l/p^d$.

The completeness of the protocol is directly given by construction. □

3.3 Complexity analysis

The protocol complexity can be computed by adding the complexity of both phases.

During the first phase, both P and V have to construct a polynomial in $GR(p^k, n)$; That means they have to first construct $\tilde{f}$ and then from there find $\overline{f}$. This can be done in $O(nT_1)$, where $O(T_1)$ is the time to evaluate the polynomial f at one point.

During the second phase, the prover will evaluate $\overline{f}$ $O(l'2^{l'})$ times. If we consider $O(T_2)$ to be the time it takes to evaluate $\overline{f}$ in a random point of $GR(p^k, n)$ and taking into account that $l' = l - \log n$, then the prover complexity will be

$$O(l'2^{l'}T_2) = O((l-\log n)2^{l-\log n}T_2) = O(\frac{l2^l}{n}T_2)$$

Evaluation of $\overline{f}$ in a point of the Galois Ring will have the same amount of multiplications (products in the Galois Ring) as evaluations of f in the ring $\mathbb{Z}_{p^k}$. Using Fast Fourier Transform and having the irreducible polynomial we used for the quotient to generate the Galois Ring be a sparse polynomial, we can compute a multiplication of two elements in $GR(p^k, n)$ in time $O(n \log n)$. That means that we have $O(T_2) = O(T_1 n \log n)$

With that we finally have a prover complexity of

$$O(\frac{l2^l}{n}T_2) = O(\frac{l2^l}{n}T_1 n \log n) = O(l2^l T_1 \log n)$$

We can see that in comparison to the classic sumcheck over finite fields which has a prover time of $O(l2^l T_1)$, we only have a $\log n$ overhead factor. In our protocol n is also the bits of security and can be set to 128 or 256, meaning it's not a big value.

When it comes to verifier complexity, during the second phase V has to evaluate l' univariate polynomials in $GR(p^k, n)^{\leq c}[X]$ on a random point, and during the last step of the protocol V evaluates $\overline{f}$ in a random point. This first l' evaluations can be computed in $O(l'cn \log n)$. The final evaluation will take $O(T_2) = O(n \log n T_1)$. We will end up with a verifier complexity of

$$O(n \log n(l'c + T_1))$$

The overall communication of the protocol is l' univariate polynomials of maximum degree c over $GR(p^k, n)$, which is the same as $O(l'cn)$ elements.

We can see all costs summarized in the following table

Proof size	V time	P time
$O(l'cn)$	$O(n \log n(l'c + T_1))$	$O(l2^l T_1 \log n)$

Table 1: *Complexity analysis Protocol* 3

3.4 Quasilinear Prover time multilinear function Sumcheck

The Libra paper proposed an optimization to the Sumcheck protocol for the mulitlinear case that uses the fact that each iteration the amount of monomials is reduced by half as we are evaluating one of the variables.

In the general multilinear case, evaluating $f \in GR(p^k, n)[X_1, \cdots, X_{l'}]$ would take $O(T_1) = O(2^{l'} l' \log l') = O(2^l \log n)$ giving a total complexity on the sumcheck protocol of $O(l2^{2l} \log n)$.

The optimization proposed in Libra can also be done in our case reducing the prover complexity to $O(2^l \log^2 n)$, that is removing a $O(l2^l)$ factor from the trivial approach.

This optimization consists on the precomputation of f in all points of the form $(r_1, \cdots, r_{i-1}, b) \quad \forall i \in [l], b \in \{0,1\}^{l-i+1}$ and r_j the random challenges in the protocol. If this evaluations can be precomputed in time $O(2^{l'} n \log(n)) = O(2^l \log(n))$ then during the protocol we can use them to add the elements without having to compute them. We will add $O(2^{l'} + 2^{l'-1} + \cdots + 1) = O(2^{l'+1}) = O(2^{l'})$ elements in $GR(p^k, n)$ which can be done in $O(2^{l'} n) = O(2^l)$ and therefore the whole algorithm complexity will be $O(2^l \log n)$.

The precomputing algorithm can be seen in Algorithm 1 and taking into account that the elements in $\boldsymbol{A}$ are elements in $GR(p^k, n)$ we end up having an overall complexity of $O(2^{l'+1} n \log n) = O(2^l \log n)$.

Algorithm 1 $\mathcal{F} \leftarrow$ FunctionEvaluations$(f, \boldsymbol{A}, r_1, \cdots, r_{l'})$

Input: Multilinear $f \in GR(p^k, d)[X_1, \cdots, X_{l'}]$, initial bookkeeping table $\boldsymbol{A}$, random $r_1, \cdots, r_{l'}$ in $GR(p^k, d)$
Output: All function evaluations $f(r_1, \cdots, r_{i-1}, b_i, \cdots, b_{l'})$

1: **for** $i = 1, \cdots, l$ **do**
2: **for** $b \in \{0,1\}^{l'-i}$ **do**
3: **for** $t = 0, 1$ **do**
4: Let $f(r_1, \cdots, r_{i-1}, t, b) = \boldsymbol{A}[b] \cdot (1-t) + \boldsymbol{A}[b + 2^{l-i}] \cdot t$
5: $\boldsymbol{A}[b] = \boldsymbol{A}[b] \cdot (1 - r_i) + \boldsymbol{A}[b + 2^{l'-i}] \cdot r_i$
6: Let $\mathcal{F}$ contain all function evaluations $f(.)$ computed at Step 4
7: **return** $\mathcal{F}$

4 Degree D Sumcheck over $\mathbb{Z}_{p^k}$

The previously presented protocol is of great use when working with a multivariate polynomial in it's expanded form, the problem comes when this polynomial is presented as the product of smaller polynomials. Having to compute the resulting polynomial may end up being very expensive, and it also breaks the nice product structure, so a modification of the previous protocol has to be used to tackle this issue.

We want to build a protocol that enables a prover P prove to a verifier V that the following equality holds

$$H = \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_l \in \{0,1\}} \prod_{i=1}^{D} f_i(b_1, \cdots, b_l)$$

where $f_i \in \mathbb{Z}_{p^k}^{\leq c}[X_1, \cdots, X_l]$ and $i \in [D]$.

The trivial approach to compute the image Φ of each polynomial (after doing the batching trick) in the product and solve it the same way as before is not correct, as Φ is not a ring isomorphism and therefore

$$\Phi(\boldsymbol{H}) \neq \sum_{b_1 \in \{0,1\}} \cdots \sum_{b_{l'} \in \{0,1\}} \prod_{i=1}^{D} \Phi(\tilde{f}_i(b_1, \cdots, b_{l'}))$$

Since finding nice (injective) homomorphisms is hard and they may not exist between $\mathbb{Z}_{p^k}$ and $GR(p^k, d)$, we will use a tool called degree-D Reverse Multiplicative Friendly Embedding (RMFE). This tool consists a group homomorphism pair (ϕ, ψ) where $\phi : \mathbb{Z}_{p^k}^n \to GR(p^k, d)$, $\psi : GR(p^k, d) \to \mathbb{Z}_{p^k}^n$ and $\prod_{i=1}^{D} x_i = \psi(\prod_{i=1}^{D} \phi(x_i))$ with $x_1, \cdots, x_D \in \mathbb{Z}_{p^k}^n$.

It's important to see that since this is not a ring homomorphism and therefore the composition of ϕ with ψ will not be the identity in $\mathbb{Z}_{p^k}$, that is $\psi(\phi(\boldsymbol{x})) \neq \boldsymbol{x}$. Nonetheless, since we specified in the preliminaries that we would be using ϕ such that $\phi(\mathbf{1}) = 1$, then

$$\psi(\phi(\boldsymbol{x})) = \psi(\phi(\boldsymbol{x}) \cdot \phi(\mathbf{1}) \cdots \phi(\mathbf{1})) = \boldsymbol{x} \cdot \mathbf{1} \cdots \mathbf{1} = \boldsymbol{x}$$

With this tool and using it's linearity property we will have the following equality

$$\boldsymbol{H} = \psi(\sum_{b_1, \cdots, b_l \in \{0,1\}} \prod_{i=1}^{D} \phi(\tilde{f}_i(b_1, \cdots, b_l)))$$

With that in mind we can dive into the protocol construction.

4.1 Protocol construction

We can split the protocol into 3 steps, the precomputating phase, the setup phase and the interactive phase. The last phase will be identical to the one in Protocol 3, nonetheless the other ones will vary.

Step 1 - Precomputing phase Given a polynomial vector $\tilde{f}$, when computing $\phi(\tilde{f}) = \hat{f}$ we may need to compute many instances of the function ϕ (one for each coefficient). To speed up this process we can use the fact that ϕ is a linear function, and therefore if we find the image of the canonical basis, we can compute the image of any vector in $\mathbb{Z}_{p^k}$ by linearity.

Therefore the precomputing phase will consider $\mathbb{Z}_{p^k}^n$ to be a $\mathbb{Z}_{p^k}$-module and compute the image of it's canonical basis to be later used when computing multiple images of ϕ.

This precomputing phase only depends on the chosen RMFE, and therefore can be reused in multiple separate instances of Sumcheck. That is why it is also interesting to point out that ψ is linear, and therefore we can use the same idea as before and compute the image of the canonical basis.

Step 2 - Setup phase The setup phase will start the same way it did in the previous section, by batching the first $\log n$ summations together and have the sumcheck turn into a polynomial vector equality

$$\boldsymbol{H} = \sum_{b_{\log n+1}, \cdots, b_l \in \{0,1\}} \prod_{i=1}^{D} \tilde{f}_i(b_{\log n+1}, \cdots, b_l)$$

Protocol 4 (Degree D Sumcheck over $\mathbb{Z}_{p^k}$)**.** *The protocol proceeds in 3 main steps*

Step 1 - Precomputing phase

- *Compute the image ϕ of the canonical basis of the $\mathbb{Z}_{p^k}$-module $\mathbb{Z}_{p^k}^n$.*
- *Compute the image ψ of the canonical basis of the $\mathbb{Z}_{p^k}$-module $GR(p^k, d)$.*

Step 2 - Setup phase

- *P will batch the first $\log d$ summations together reducing the statement to proving*

$$\begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_n \end{pmatrix} = \sum_{b_{\log n+1},\cdots,b_l \in \{0,1\}} \prod_{i=1}^{D} \begin{pmatrix} f_i(0,\cdots,0,0,b_{\log n+1},\cdots,b_l) \\ f_i(0,\cdots,0,1,b_{\log n+1},\cdots,b_l) \\ \vdots \\ f_i(1,\cdots,1,1,b_{\log n+1},\cdots,b_l) \end{pmatrix} = \sum_{b_{\log n+1},\cdots,b_l \in \{0,1\}} \prod_{i=1}^{D} \tilde{f}_i(b_{\log n+1},\cdots,b_l)$$

  with $\tilde{f}_i : \mathbb{Z}_{p^k}^{l-\log n} \to \mathbb{Z}_{p^k}^n$. It can be easily seen that $H = \sum_{i=1}^n H_i$.
- *Both V and P will compute $\hat{f} = \phi(\tilde{f}_i)$ for $\phi : \mathbb{Z}_{p^k}^n \to GR(p^k, d)$. We will therefore have several polynomials $\hat{f}_i \in GR(p^k, n)[X_{\log n-1}, \cdots, X_l]$.*
- *P will send a value $\hat{H}$ claimed to be*

$$\hat{H} = \sum_{b_{\log n+1},\cdots,b_l \in \{0,1\}} \prod_{i=1}^{D} \hat{f}_i(b_{\log n+1},\cdots,b_l)$$

  and V will check that $\psi(\hat{H}) = \boldsymbol{H}$ and reject if it's not true.

Step 3 - Interactive phase

- *Consider $l' = l - \log n$. In the first round, P sends a univariate polynomial*

$$\hat{g}_1(x_1) := \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_{l'} \in \{0,1\}} \prod_{i=1}^{D} \hat{f}_i(x_1, b_2, \cdots, b_{l'})$$

  V checks that $\hat{H} = \hat{g}_1(0) + \hat{g}_1(1)$ and sends a random challenge $r_1 \in GR(p^k, n)$ to P.
- *In the i-th round, P sends a univariate polynomial*

$$\hat{g}_i(x_i) := \sum_{b_{i+1} \in \{0,1\}} \cdots \sum_{b_{l'} \in \{0,1\}} \prod_{i=1}^{D} \hat{f}_i(r_1, r_2, \cdots, r_{i-1}, x_i, b_{i+1}, \cdots, b_{l'})$$

  V checks that $\hat{g}_{i-1}(r_{i-1}) = \hat{g}_i(0) + \hat{g}_i(1)$ and sends a random challenge $r_i \in GR(p^k, n)$ to P.
- *In the l'-th round P will send a univariate polynomial*

$$\hat{g}_{l'}(x_{l'}) := \prod_{i=1}^{D} \hat{f}_i(r_1, \cdots, r_{l'-1}, x_{l'})$$

  V will check that $\hat{g}_{l'-1}(r_{l'-1}) = g_l(0) + g_{l'}(1)$ and generates a random challenge $r_{l'} \in \mathbb{F}$. V will now evaluate $\hat{f}_i(r_1, r_2, \cdots, r_{l'})$ for $i \in [D]$ and will verify that $\hat{g}_{l'}(r_{l'}) = \prod_{i=1}^{D} \hat{f}_i(r_1, r_2, \cdots, r_{l'})$ and accept if the equality holds or reject otherwise.

where the product between vectors is the $\star$ product.

Both the prover and the verifier will now compute $\hat{f}_i = \phi(\tilde{f}_i) \in GR(p^k, d)[X_1, \cdots, X_n]$ using the precomputed images of the canonical basis to speed up the computation.

The prover will now send a value $\hat{H}$ claimed to be equal to $\sum_{b_1\in\{0,1\}}\cdots\sum_{b_l\in\{0,1\}}\prod_{i=1}^{D}\phi(\tilde{f}_i(b_1,\cdots,b_l))$.

The verifier will check that

$$\boldsymbol{H}=\psi(\hat{H})$$

And reject if it's not true.

The only thing left for the prover to do is to prove that

$$\hat{H}=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_l\in\{0,1\}}\prod_{i=1}^{D}\hat{f}_i(b_1,\cdots,b_l)$$

which will be done in the interactive phase.

Step 3 - Interactive phase This last step will the the same as the interactive phase in Protocol 3, as we are just proving a sumcheck equality in the Galois Ring.

The whole protocol is shown in Protocol 4.

4.2 Completeness and soundness analysis

Lemma 4.1. *Given D polynomials $f_i \in \mathbb{Z}_{p^k}[X_1,\cdots,X_l]$ for $i\in[D]$, if $\hat{H}=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_{l'}\in\{0,1\}}\prod_{i=1}^{D}\hat{f}_i(b_1,\cdots,b_{l'})$ and $\psi(\hat{H})=\boldsymbol{H}$, then $H=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_l\in\{0,1\}}\prod_{i=1}^{D}f_i(b_1,\cdots,b_l)$.*

Proof. Since ψ is a linear mapping and using the RMFE definition, we have

$$\begin{aligned}\boldsymbol{H}=\psi(\hat{H})&=\psi(\sum_{b_1\in\{0,1\}}\cdots\sum_{b_{l'}\in\{0,1\}}\prod_{i=1}^{D}\hat{f}_i(b_1,\cdots,b_{l'}))\\&=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_{l'}\in\{0,1\}}\psi(\prod_{i=1}^{D}\phi(\tilde{f}_i(b_1,\cdots,b_{l'})))\\&=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_{l'}\in\{0,1\}}\prod_{i=1}^{D}\tilde{f}_i(b_1,\cdots,b_{l'})\end{aligned}$$

And we know by construction that if

$$\boldsymbol{H}=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_{l'}\in\{0,1\}}\prod_{i=1}^{D}\tilde{f}_i(b_1,\cdots,b_{l'})$$

then we have

$$H=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_l\in\{0,1\}}\prod_{i=1}^{D}f_i(b_1,\cdots,b_l)$$

□

The previous lemma ensures us that if P is able to prove the correctness of the sum in the Galois Ring, then the original sum is correct. This will bring the soundness of the first part of the protocol, the correctness is given by the other implication, which is trivial as $\hat{H}$ is defined as

$$\hat{H}:=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_{l'}\in\{0,1\}}\prod_{i=1}^{D}\hat{f}_i(b_1,\cdots,b_{l'})$$

and will always be true no matter what H is.

The completeness and soundness of the interactive phase is given by Lemma 3.2 from the previous section.

Theorem 4.2. *Protocol 4 is complete and has a soundness error $\delta_s\leq l'/p^d$.*

Proof. V has access to the polynomial f as well as the precomputed images ϕ and ψ of the canonical basis and can compute the transformation $\hat{f}$ efficiently. Lemma 3.2 will ensure that

$$\hat{H}=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_{l'}\in\{0,1\}}\prod_{i=1}^{D}\hat{f}_i(b_1,\cdots,b_{l'})$$

with a soundness error of $\delta_s\leq l/p^d$. Since V can verify that $\psi(\hat{H})=\boldsymbol{H}$, Lemma 4.1 proves that with no extra soundness error we have

$$H=\sum_{b_1\in\{0,1\}}\cdots\sum_{b_l\in\{0,1\}}\prod_{i=1}^{D}f_i(b_1,\cdots,b_l)$$

Therefore the whole protocol's soundness error will be $\delta_s\leq l/p^d$.

The completeness of the protocol is again given by construction. □

4.3 Complexity analysis

As expected, working with a more complex map from $\mathbb{Z}_{p^k}^n$ to $GR(p^k,d)$ such as RMFE, will increase the complexity of the whole protocol.

The main overhead with respect to the previously presented Sumcheck over $\mathbb{Z}_{p^k}$ will come from the fact that the cardinality of the new ring will have to be bigger than the one in the original ring. The specific size will be given by Lemma 4.3 first presented in Corollary 5 [https://eprint.iacr.org/2023/173.pdf].

Lemma 4.3. *Let $F/\mathbb{F}_q$ be a function field of genus $\mathfrak{g}$ with n distinct rational places and a place of degree k. Then there exists an $(n,k;D)$-RMFE over $\mathbb{F}_q$ as follows.*

1. *if q is a square, there is a constructive family of $(n,k;D)$-RMFE over $\mathbb{F}_q$ with $n\to\infty$ and $\frac{k}{n}\to D+\frac{2D}{\sqrt{q}-1}$.*
2. *if $q=p^{2m+1}$ for a prime p, there is a constructive family of $(n,k;D)$-RMFE over $\mathbb{F}_q$ with $n\to\infty$ and $\frac{k}{n}\to D+\frac{D(p+1+\epsilon)}{p^{m+1}-1}$, where $\epsilon=\frac{p-1}{p^m-1}$.*

We also have another lemma from Lemma 5 [https://eprint.iacr.org/2023/173.pdf] that brings this RMFE from fields to rings.

Lemma 4.4. *Let $q = p^r$ for a prime p. Then the $(n,k;D)$-RMFE over $\mathbb{F}_q$ constructed in Lemma 5 [https://eprint.iacr.org/2023/173.pdf] can be lifted to an $(n,k;D)$-RMFE over $GR(p^l,r)$ for any $l \geq 1$.*

With that in mind, we have the following Corollary

Corollary 4.4.1. *There exists a constructive family of $(n,k;D)$-RMFE over $\mathbb{Z}_{p^l}$ for all $l = p^{2m} \geq 1$ with $n \to \infty$ and $\frac{k}{n} \to D + \frac{2D}{p^m-1}$.*

With this lemma we can now see that the Galois Ring will be $O(D + \frac{2D}{p^m-1})$ times bigger than the original field. We will therefore have $O(d) = O(n(D + \frac{2D}{p^m-1}))$. For our purposes m will be large enough to be able to consider the second term negligible, therefore we have $O(d) = O(nD)$.

We can now analyze the complexity of each phase the same way we did with the previous protocol.

During the setup phase, both P and V build the polynomial in the Galois Ring using the precomputed images of the canonical basis. They will first have to construct $\tilde{f}_i$ for $i \in [D]$ and this can be done in $O(n\sum_{i=0}^{D} T_i)$ where $O(T_i)$ is the time to evaluate f_i at one point. We will define $T = \sum_{i=1}^{D} T_i$. Then they will have to find $\hat{f}_i$ for $i \in [D]$ which takes $O(n^2T)$ assuming that $O(T_i)$ is also the number of coefficients f_i has. P will then have to compute the new sum which can be done in time $O(2^{l'}(T' + d\log d)) = O(\frac{2^l(T'+d\log d)}{n})$ where $T' = \sum_{i=1}^{D} T'_i$ with $O(T'_i)$ the time to evaluate $\hat{f}$ in a point of the $GR(p^k,d)$. That means the second phase has a verifier complexity of $O(n^2T')$ and a prover complexity of $O(\frac{2^l(T'+d\log d)}{n} + n^2T)$.

Any operation in $GR(p^k,d)$ using FFT takes $O(d\log d)$ operations in $\mathbb{Z}_{p^k}$. Since $\hat{f}_i$ and f_i maintain the same structure, we can assume that $O(T'_i) = O(T_i d\log d)$ and the previous prover complexity will end up being $O(\frac{2^l Td\log d}{n} + n^2T)$. Now using that $O(d) = O(n(D + \frac{2D}{p^m-1}))$ and that n^2T is much smaller than $\frac{2^l Td\log d}{n}$ we end up with a prover complexity for the second phase of

$$O(2^l TD\log d)$$

The last phase will have the same complexity as the previous algorithm, but with the new d. That means that the interactive phase will have a prover complexity of

$$O(l'2^{l'}T') = O(\frac{l2^l Td\log d}{n}) = O(l2^l TD\log d)$$

That means the overall prover complexity will end up being $O(l2^l TD\log d)$. Taking into account that the classical sumcheck has a prover complexity of $O(ls^l T)$, we can see that this new version adds an overhead of $O(D\log d)$ to the overall cost.

The verifier during the second phase will have to do l' evaluations of these given polynomials, involving multiplications in $GR(p^k,d)$. This will give us a verifier time of $O(l'cd\log d)$, where c is the max degree on the individual variables of the multivariate polynomial. Finally, the end step will be to evaluate the whole polynomial in a single point and evaluating it at one random point takes $O(d\log(d)T)$ time.

The total communication of the protocol will be then $O(l'cd)$, the l' univariate polynomials of c terms in $GR(p^k,d)$ each.

We can see all costs summarized in the following table

Proof size	V time	P time
$O(l'cd)$	$O(d\log d(l'c+T))$	$O(l2^l TD\log d)$

Table 2: *Complexity analysis Protocol* 4

4.4 Quasilinear Prover time Sumcheck for products of multilinear functions

The same way we did on the previous section, if we consider the case where all functions are multilinear then we have $O(T_i) = O(2^{l'}d\log d) = O(2^l D\log d)$ making the overall prover complexity of the protocol $O(l2^{2l}D^2\log^2 d)$ when we run the protocol in the naive way.

The algorithm used to reduce the prover complexity on the previous section won't work now since the product of multilinear polynomials is not multilinear. Nonetheless we can use the same idea to build a new protocol also proposed in the Libra paper with prover complexity of $O(2^l D^2\log d)$ removing a $O(l2^l D)$ factor from the cost and making it quasilinear.

The way this is done is by precomputing in $O(D2^{l'}d\log d) = O(2^l D^2\log d)$ time the images of each function in the product using Algorithm 1 but now working in a bigger ring. Once we have them we can run the sumcheck protocol using those preimages in time $O(2^{l'}d\log d) = O(2^l D\log d)$.

The overall prover complexity ends up being

$$O(2^l D^2\log d)$$

5 GKR over $\mathbb{Z}_{p^k}$

We now have seen two tools to send packed values from the ring $\mathbb{Z}_{p^k}$ to $GR(p^k,d)$, the trivial bijective

Protocol 5 (GKR protocol over $\mathbb{Z}_{p^k}$)**.** *Let $C : \mathbb{Z}_{p^k}^{S_i} \to \mathbb{Z}_{p^k}^{S_o}$ be a m-depth layered arithmetic circuit. Without loss of generality assume S_i and S_o are both powers of 2 and we can pad them if not. The protocol proceeds as follows.*

1. *Define the multilinear extension of the output array as $\check{W}_0$. Both P and V will run a slightly modified degree 3 Sumcheck to prove the correctness of*
$$\check{W}_0(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1}} \check{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_1(\boldsymbol{b}) + \check{W}_1(\boldsymbol{c})) + \check{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_1(\boldsymbol{b}) \cdot \check{W}_1(\boldsymbol{c})) \quad \forall \boldsymbol{z} \in \mathbb{Z}_{p^k}^{s_0}$$
2. *P and V will now turn that into an equality in the Galois Ring by considering the trivial inclusion $\mathbb{Z}_{p^k} \in GR(p^k,d)$. The polynomial will be the same as the coefficients will still be in $\mathbb{Z}_{p^k}$, but we will think of them as elements in $GR(p^k,d)$.*
3. *The verifier will choose a random challenge $\boldsymbol{r_0} \in GR(p^k,d)^{s_0'}$ and send it to the prover. The prover and verifier will compute $W_0(\boldsymbol{r_0})$.*
4. *They will now run a classic sumcheck over Galois Rings to prove that*
$$\hat{W}_0(\boldsymbol{r_0}) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1}} \hat{add}_0(\boldsymbol{r_0},\boldsymbol{b},\boldsymbol{c})(\hat{W}_1(\boldsymbol{b}) + \hat{W}_1(\boldsymbol{c})) + \hat{mult}_0(\boldsymbol{r_0},\boldsymbol{b},\boldsymbol{c})(\hat{W}_1(\boldsymbol{b}) \cdot \hat{W}_1(\boldsymbol{c}))$$
5. *At the end of the sumcheck protocol the verifier will have to evaluate the right hand side of the equation in the random point $(\boldsymbol{r_b},\boldsymbol{r_c})$ but V doesn't have access to $\hat{W}_1(z)$. The prover will send the claimed values $c_b = \hat{W}_1(r_b)$ and $c_c = \hat{W}_1(r_c)$ and the verifier will check the last equality with those values.*
6. *The prover and verifier will continue with the protocol to prove the correctness of the claimed evaluations of W_1. Consider the linear function $l_1 : GR(p^k,d) \to GR(p^k,d)^{s_1}$ such that $l_1(0) = r_b$ and $l_1(1) = r_c$. Then the prover will send the univariate polynomial $q_1 = W_1 \circ l_1 : GR(p^k,d) \to GR(p^k,d)$ to the verifier. The verifier will check that $q(0) = c_b$ and $q(1) = c_c$.*
7. *The verifier will choose a new random challenge $r_1' \in GR(p^k,d)$ and define $r_1 = l_1(r_1')$ and send it to the prover. They will now recurse the protocol to prove that*
$$\hat{W}_1(\boldsymbol{r_1}) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1}} \hat{add}_0(\boldsymbol{r_1},\boldsymbol{b},\boldsymbol{c})(\hat{W}_2(\boldsymbol{b}) + \hat{W}_2(\boldsymbol{c})) + \hat{mult}_0(\boldsymbol{r_1},\boldsymbol{b},\boldsymbol{c})(\hat{W}_2(\boldsymbol{b}) \cdot \hat{W}_2(\boldsymbol{c}))$$
8. *On the last round of the GKR protocol the verifier will receive $\hat{W}_m(\boldsymbol{r_m})$ which can also be computed from the input. V will compute it and compare if they are equal, accept if they are and reject otherwise.*

mapping from Definition 2.4 and the RMFE from Definition 2.5.

We will now present a third tool called the trivial inclusion that sends a value to itself, but in the Galois Ring.

Definition 5.1. *We will define the trivial inclusion mapping as*

$$\begin{aligned}\Psi : \mathbb{Z}_{p^k} &\to GR(p^k,n)\\ x &\mapsto x\end{aligned}$$

This is well defined as $\mathbb{Z}_{p^k} \subseteq GR(p^k,d)$.

We haven't used this mapping as in the previous cases, even though it would have worked, it would bring a $O(n)$ overhead factor to the whole protocol. In the GKR case, even though it has an $O(\log n)$ overhead factor in comparison to the protocol presented in the next section, it presents such a simple construction that is worth considering.

5.1 Protocol construction

The protocol will be the same as the classic one, but the sent challenges will be from the Galois Ring instead of $\mathbb{Z}_{p^k}$.

The prover during each of it's rounds will prove to the verifier the correctness of the polynomial equality

$$\check{W}_i(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1}} \check{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_{i+1}(\boldsymbol{b}) + \check{W}_1(\boldsymbol{c})) + \check{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_{i+1}(\boldsymbol{b}) \cdot \check{W}_1(\boldsymbol{c}))$$

Since we have that $f \in \mathbb{Z}_{p^k}[X_1,\cdots,X_{s_i}]$, then $f \in GR(p^k,d)[X_1,\cdots,X_{s_i}]$. We then know that we can prove the previous equality using the Galois Ring

version of the Schwartz Zipple lemma evaluating in a random point.

At the beginning of the protocol the verifier will choose a random challenge $r_0 \in GR(p^k, d)$ and send it to the prover. The prover and the verifier will now proceed to run the sumcheck protocol sending random challenges from $GR(p^k, d)$ to prove that

$$\check{W}_0(\boldsymbol{r_0}) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{s_1}} \check{add}_0(\boldsymbol{r_0}, \boldsymbol{b}, \boldsymbol{c})(\check{W}_1(\boldsymbol{b}) + \check{W}_1(\boldsymbol{c})) + \check{mult}_0(\boldsymbol{r_0}, \boldsymbol{b}, \boldsymbol{c})(\check{W}_1(\boldsymbol{b}) \cdot \check{W}_1(\boldsymbol{c}))$$

At the last step of the protocol the verifier will have to evaluate the right hand side of the equation. Nonetheless V doesn't have access to $W_1(\boldsymbol{z})$ and therefore can't compute the values $W_1(\boldsymbol{r_b})$ and $W_1(\boldsymbol{r_c})$. The prover will send the claimed values $c_b = W_1(\boldsymbol{r_b})$ and $c_c = W_1(\boldsymbol{r_b})$ and the verifier will use those values to check the last equality of the sumcheck protocol.

The verifier and the prover will now proceed to prove the correctness of the claimed evaluations of $W_1(\boldsymbol{z})$. To do that P will build the linear function $l_1 : GR(p^k, d) \to GR(p^k, d)^{s_1}$ such that $l_1(0) = r_b$ and $l_1(1) = r_c$ and send $q_1 = W_1 \circ l_1$ to the verifier. The verifier will check that $q_1(0) = c_b$ and $q_1(1) = c_c$.

The verifier will then choose a random challenge $r_1' \in GR(p^k, d)$ and define $r_1 = l_1(r_1')$. They will now recurse the protocol to prove

$$q_1(r_1') = \check{W}_1(\boldsymbol{r_1}) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{s_2}} \check{add}_1(\boldsymbol{r_1}, \boldsymbol{b}, \boldsymbol{c})(\check{W}_2(\boldsymbol{b}) + \check{W}_1(\boldsymbol{c})) + \check{mult}_1(\boldsymbol{r_1}, \boldsymbol{b}, \boldsymbol{c})(\check{W}_2(\boldsymbol{b}) \cdot \check{W}_1(\boldsymbol{c}))$$

Once the protocol gets to the last round, the verifier will receive $\hat{W}_m(\boldsymbol{r_m})$ which is a value he can compute as he has access to the input. He will compute this value and compare to the claimed one, accept if it's equal, reject otherwise.

The soundness of the protocol won't be discussed as it's proofs are the same as the ones in the original GKR protocol, but now using the new Schwartz Zipple lemma and similar proofs have also been done in previous sections.

5.2 Complexity analysis

We will first analyse the complexity of the protocol if we where to run it naively and then using techniques from the Libra paper.

Each sumcheck of the protocol can be done in $O(2s_i 2^{2s_i} T)$ where $O(T)$ is the time to evaluate the right hand side of the equation. We know that the right hand side is the product of multilinear polynomials, one of them being of $2s_i$ variables (in $GR(p^k, d)$) and therefore we can assume that if the evaluations where computed naively $O(T) = O(2^{2s_i} d \log d)$.

If we consider $M_s = max_i s_i$ and assume $O(s_i) = O(M_s)$, then the overall complexity of the protocol will be

$$O(m M_s 2^{4M_s} d \log d)$$

We can greatly reduce the complexity of the protocol by applying the techniques presented in the Libra paper. Since they will be later used in a more complex scenario, we won't present them here.

For this case we use the same algorithms, just having the random values from the Galois Ring instead of a finite field, increasing the complexity from the Libra GKR by $O(d \log d)$, where d is the security parameter. We end up having an overall complexity of

$$O(m 2^{M_s} d \log d)$$

which is $O(M_s 2^{3M_s})$ times better than the naive approach.

On the verifier side during each sumcheck he will receive s_i values in $GR(p^k, d)$ and one polynomial of degree s_i at the end. V will have to evaluate this polynomial on three points along with the evaluations of $mult_i$ and add_i which we assume that can be done in $O(T)$ time. This gives an overall verifier complexity of

$$O(mTM_s \log M_s)$$

The total proof size will be

$$O(mM_s)$$

We can summarize all the costs in the following table:

Proof size	$O(mM_s)$
V time	$O(mTM_s \log M_s)$
P time	$O(m2^{M_s} d \log d)$

Table 3: *Complexity analysis Protocol* 5

6 GKR over $\mathbb{Z}_{p^k}$ via RMFE

Having seen both the Sumcheck and degree D Sumcheck over $\mathbb{Z}_{p^k}$ we can now present it's first use case in the construction of the GKR protocol over $\mathbb{Z}_{p^k}$.

This protocol presents a challenge, since it relies on recursivity by reducing a claim from one layer to a claim in the previous one. In the original GKR protocol, this claim is the evaluation of a polynomial in two points that is needed in the last step of the Sumcheck, nonetheless in our case this evaluation is on the Galois Ring, but the next polynomial has to fully be in $\mathbb{Z}_{p^k}$ to apply the previous presented

Protocol 6 (GKR protocol via RMFE). *Let $C : \mathbb{Z}_{p^k}^{S_{input}} \to \mathbb{Z}_{p^k}^{S_{output}}$ be a m-depth layered arithmetic circuit. Without loss of generality assume S_i are all powers of 2 and we can pad them if not. Define $s_i = \log S_i$ and $s_i' = s_i - \frac{\log n}{2}$. Let (ϕ, ψ) be the RMFE pair that will be used throughout the protocol. The protocol proceeds as follows.*

1. *Define the multilinear extension of the output array as $\check{W}_0$. Both P and V will run a slightly modified degree 3 Sumcheck to prove the correctness of*

$$\check{W}_0(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1}} \check{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_1'(\boldsymbol{b}) + \check{W}_1'(c)) + \check{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_1'(\boldsymbol{b}) \cdot \check{W}_1'(\boldsymbol{c})) \quad \forall \boldsymbol{z} \in \mathbb{Z}_{p^k}^{s_0}$$

2. *During the setup phase, we will split the batching parameters equally between the 2 variables $\boldsymbol{b}$ and $\boldsymbol{c}$. That means we will have $\hat{W}_1' \in GR(p^k,d)[X_1, \cdots, X_{s_1'}]$. P will compute the $GR(p^k,d)$ polynomial in $\boldsymbol{z}$*

$$H_0(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1'}} \hat{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\hat{W}_1'(\boldsymbol{b}) + \hat{W}_1'(\boldsymbol{c})) + \hat{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\hat{W}_1'(\boldsymbol{b}) \cdot \hat{W}_1'(\boldsymbol{c}))$$

   and both P and V will compute $\hat{W}_0(\boldsymbol{z})$.
3. *The verifier will choose a polynomial vector $l_0 : GR(p^k,d) \to GR(p^k,d)^{s_0'}$ with the coefficients in $\mathbb{Z}_{p^k}$ and send it to the prover. The prover will compute $H_0 \circ l_0$ and send it to the verifier. The verifier will check that $\psi(H_0 \circ l_0) = \psi(\hat{W}_0 \circ l_0)$ by checking that the ψ image of each coefficient in the polynomials is equal.*
4. *They will proceed with the interactive phase of the protocol now proving*

$$H_0(l_0(z)) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1'}} \hat{add}_0(l_0(z),\boldsymbol{b},\boldsymbol{c})(\hat{W}_1'(\boldsymbol{b}) + \hat{W}_1'(\boldsymbol{c})) + \hat{mult}_0(l_0(z),\boldsymbol{b},\boldsymbol{c})(\hat{W}_1'(\boldsymbol{b}) \cdot \hat{W}_1'(\boldsymbol{c}))$$

5. *They will then run the Sumcheck Protocol the same way it's done in Protocol 4. We will define $\boldsymbol{r_b}$ and $\boldsymbol{r_c}$ to be the random challenges in $GR(p^k,d)$ sent by the verifier. The Sumcheck protocol will run until the point where V has to evaluate the polynomial in the last iteration.*
6. *During this last step of the Sumcheck the prover will send the needed evaluations $c_b = W_1'(\boldsymbol{r_b})$ and $c_c = W_1'(\boldsymbol{r_c})$ to the verifier and V will check that the equality holds.*
7. *They will now proceed to prove the correctness of these evaluations. The prover will construct the two polynomial vectors l_{1b} and l_{1c} such that $l_{1b}(\xi) = \boldsymbol{r_b}$ and $l_{1c}(\xi) = \boldsymbol{r_c}$. P will send the polynomials $\hat{W}_1'(l_{1b})$, $\hat{W}_1'(l_{1c})$, $\hat{W}_1(\boldsymbol{y}, l_{1b})$ and $\hat{W}_1(\boldsymbol{y}, l_{1c})$ to the verifier and V will check the polynomial equality*

$$\sum_{i=0}^{n} \psi(\hat{W}_1(\cdot, l_{1*}(z)))(\boldsymbol{y})[i] = \psi(\hat{W}_1'(l_{1*}(z)))[\boldsymbol{y}] \quad \forall \boldsymbol{y} \in \{0,1\}^{\log n}$$

   to ensure that all four sent polynomials come from the same W_1.
8. *V will choose two random polynomials $\alpha_1, \beta_1 \xleftarrow{\$} \mathbb{Z}_{p^k}^{<d}[X]$ and a random vector polynomial $l_1 : \mathbb{Z}_{p^k}^{n} \to \mathbb{Z}_{p^k}^{\log n}$ with each polynomial of degree $< d$. then P will now compute the polynomial $G_1(z) = \alpha_1(z)H_1(l_1(z), l_{1b}(z)) + \beta_1(z)H_1(l_1(z), l_{1c}(z))$ defined as follows*

$$G_1(z) := \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1'}} \left(\alpha_1(z)\hat{add}_o(l_1(z), l_{1b}(z), \boldsymbol{b}, \boldsymbol{c}) + \beta_1(z)\hat{add}_o(l_1(z), l_{1c}(z), \boldsymbol{b}, \boldsymbol{c})\right)(\hat{W}_2'(\boldsymbol{b}) + \hat{W}_2'(\boldsymbol{c})) + \left(\alpha_1(z)\hat{mult}_o(l_1(z), l_{1b}(z), \boldsymbol{b}, \boldsymbol{c}) + \beta_1(z)\hat{mult}_o(l_1(z), l_{1c}(z), \boldsymbol{b}, \boldsymbol{c})\right)(\hat{W}_2'(\boldsymbol{b}) \cdot \hat{W}_2'(\boldsymbol{c})) \tag{7}$$

   and send it to the verifier. P and V will also compute $\alpha_1(z)\hat{W}_1(l_1(z), l_{1b}(z)) + \beta_1(z)\hat{W}_1(l_1(z), l_{1c}(z))$. The verifier will check that $\psi(\alpha_1(z)\hat{W}_1(l_1(z), l_{1b}(z)) + \beta_1(z)\hat{W}_1(l_1(z), l_{1c}(z))) = \psi(\hat{G}_1(z))$ by checking the equality in each coefficient.
9. *P will now run a sumcheck protocol to prove 7. At the last step V will need the evaluations of $\hat{W}_2$ in two points. This will be solved the same way as before and the algorithm will run recursively until it reaches the last layer where the verifier will compute $\hat{W}_m$ from the known input layer and check the last equality.*

Sumchecks. To solve this we will modify the idea presented by Chiesa et al. in [2] where the claim of the evaluations of the polynomial on two points is combined into one sumcheck by using a linear combination.

Consider C to be an m-layered circuit over $\mathbb{Z}_{p^k}$ with each layer having S_i gates. Without loss of generality we will consider S_i to be a power of 2 and if that is not the case we will pad it until it is with dummy gates. We will define $s_i = \log S_i$. Consider $W_i : \{0,1\}^{s_i} \to \mathbb{Z}_{p^k}$ to be the function that gets as input a gate label in layer i and outputs the output of that gate. Let $\check{W}_i : \mathbb{Z}_{p^k}^{s_i} \to \mathbb{Z}_{p^k}$ be it's multilinear extension. Let $\check{add}_i$ and $\check{mult}_i$ be the multilinear extensions of add_i and $mult_i$.

Given a function $f : \mathbb{Z}_{p^k}^l \to \mathbb{Z}_{p^k}$ we previously saw that

$$H = \sum_{b_1,\cdots,b_l \in \{0,1\}} f(b_1, \cdots, b_l)$$

is equivalent to

$$\begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_n \end{pmatrix} = \sum_{b_1,\cdots,b_l \in \{0,1\}} \begin{pmatrix} f(0,\cdots,0,0,b_{\log n+1},\cdots,b_l) \\ f(0,\cdots,0,1,b_{\log n+1},\cdots,b_l) \\ \vdots \\ f(1,\cdots,1,1,b_{\log n+1},\cdots,b_l) \end{pmatrix}$$

and $H = \sum_{i=1}^{n} H_i$.

This batching in the GKR case will bring some problems when recursing the protocol. During the protocol we will be working with polynomials of the form $add_i : \{0,1\}^{s_i} \times \{0,1\}^{s_{i+1}} \times \{0,1\}^{s_{i+1}} \to \mathbb{Z}_{p^k}$ and when transforming this function into a polynomial vector we will be splitting the batching parameters into the 2 last variables equally for simplicity purposes. We will need n such that $2|\log n$. We define $s_i' = s_i - \frac{\log n}{2}$ for $i \in [m]$. After the batching we will end up with polynomial vectors of the form $add_i : \{0,1\}^{s_i} \times \{0,1\}^{s_{i+1}'} \times \{0,1\}^{s_{i+1}'} \to \mathbb{Z}_{p^k}^n$.

Let's see this batching more in depth with the specific functions from the GKR. Assume we have the equality that we want to batch

$$\check{W}_0(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{s_1}} \check{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_1(\boldsymbol{b}) + \check{W}_1(\boldsymbol{c})) + \check{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_1(\boldsymbol{b}) \cdot \check{W}_1(\boldsymbol{c})) \quad \forall \boldsymbol{z} \in \mathbb{Z}_{p^k}^{s_0}$$

then we will batch the variables $\boldsymbol{b}$ and $\boldsymbol{c}$ and end up with

$$H_0(\boldsymbol{z}) := \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{s_1'}} \hat{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\hat{W}_1'(\boldsymbol{b}) + \hat{W}_1'(\boldsymbol{c})) + \hat{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\hat{W}_1'(\boldsymbol{b}) \cdot \hat{W}_1'(\boldsymbol{c})) \quad \boldsymbol{z} \in GR(p^k,d)^{s_i}$$

with $\hat{W}_1' : GR(p^k,d)^{s_1'} \to GR(p^k,d)^n$ being

$$\hat{W}_1'(\boldsymbol{z}') = \phi\left(\begin{pmatrix} \check{W}_1(0,\cdots,0,0,\boldsymbol{z}') \\ \check{W}_1(0,\cdots,0,1,\boldsymbol{z}') \\ \vdots \\ \check{W}_1(1,\cdots,1,1,\boldsymbol{z}') \end{pmatrix}\right)$$

We are using $\hat{W}_1'$ instead of $\hat{W}_1$ since we will later define $\hat{W}_1$ as the ϕ image of

$$\check{W}_1(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{s_2}} \check{add}_1(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_2(\boldsymbol{b}) + \check{W}_2(\boldsymbol{c})) + \check{mult}_1(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_2(\boldsymbol{b}) \cdot \check{W}_2(\boldsymbol{c}))$$

During the GKR protocol we will be using a (ϕ,ψ) Degree 3 RMFE pair since we will need to do that many Galois Ring multiplications before using ψ at some point. Finally, we define $M_s = \max_{i \in [m]} s_i$.

With this in mind and using the notation previously presented we can now view the protocol construction.

6.1 Protocol construction

The main skeleton of the protocol will be equal to the one in the GKR protocol presented by Chiesa et al. in [2].

P and V will first compute the polynomial W_0 from the output of the circuit as well as W_1 from the last layer. They will also compute add_0 and $mult_0$ and then extend all 4 functions to multilinear polynomials $\check{W}_0$, $\check{W}_1$, $\check{add}_0$ and $\check{mult}_0$.

These 4 functions will follow the following equality

$$\check{W}_0(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{s_1}} \check{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_1(\boldsymbol{b}) + \check{W}_1(\boldsymbol{c})) + \check{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_1(\boldsymbol{b}) \cdot \check{W}_1(\boldsymbol{c})) \quad \forall \boldsymbol{z} \in \mathbb{Z}_{p^k}^{s_0}$$

The prover and verifier will now run a modified version of a degree 3 Sumcheck protocol.

They will first compute the polynomial vector version of the equality by splitting the batching parameters into the 2 variables b and c. If we define $s_i' = s_i - \frac{\log n}{2}$ then this will result in this new equality of polynomial vectors:

$$\tilde{W}_0(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{s_1'}} \tilde{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\tilde{W}_1'(\boldsymbol{b}) + \tilde{W}_1'(\boldsymbol{c})) + \tilde{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\tilde{W}_1'(\boldsymbol{b}) \cdot \tilde{W}_1'(\boldsymbol{c})) \quad \forall \boldsymbol{z} \in \mathbb{Z}_{p^k}^{s_0}$$

Using ϕ from the Degree 3 RMFE pair, P will transform this to a summation of a polynomial over $GR(p^k,d)$ the same way it was done in Protocol 4. V will transform $\tilde{W}_0(\boldsymbol{z})$ to $\hat{W}_0(\boldsymbol{z})$. We will therefore have

$$\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1'}} \hat{add}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\hat{W}_1'(\boldsymbol{b})+\hat{W}_1'(\boldsymbol{c})) + \hat{mult}_0(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\hat{W}_1'(\boldsymbol{b})\cdot\hat{W}_1'(\boldsymbol{c}))$$

and $\hat{W}_0(\boldsymbol{z}) \in GR(p^k,d)[X_1,\cdots,X_{s_0'}]$.

In the classic GKR protocol, the verifier would now send a random challenge in $GR(p^k,d)$. In our case, we can't do that in all layers because of the recursion problem, we need to send something we can transform via ψ. We will therefore send a polynomial with integer coefficients that goes through a random point in $GR(p^k,d)$. This way, by proving the polynomial equality, we will have proven the equality in this specific random point.

We will first prove that such polynomial exists for any point in $GR(p^k,d)$.

Lemma 6.1. *Any value in $\boldsymbol{x} \in GR(p^k,d)^n$ can be the result of the evaluation at point ξ of a vector polynomial $l : GR(p^k,d) \to GR(p^k,d)^n$ where each polynomial has all the coefficients in $\mathbb{Z}_{p^k}$ and is of degree less than d.*

Proof. We know by construction that $\{1,\xi,\cdots,\xi^{d-1}\}$ is a basis of the $\mathbb{Z}_{p^k}$ module $GR(p^k,d)$. Therefore any value $x \in GR(p^k,d)$ can be written as $x = \sum_{i=0}^{d-1} a_i\xi^i \quad a_i \in \mathbb{Z}_{p^k}$. If we consider the polynomial $f(y) = \sum_{i=0}^{d-1} a_i y^i$, then this polynomial evaluates to x at point ξ. We can do the same with a vector polynomial $l : GR(p^k,d) \to GR(p^k,d)^n$ by having a polynomial in each coordinate such that at point ξ it evaluates to x_i, and therefore $l(\xi) = \boldsymbol{x}$. □

The verifier will then choose a random point in $GR(p^k,d)^{s_0}$ and send the vector polynomial l_0 with coefficients in $\mathbb{Z}_{p^k}$ that goes through that point when evaluated at ξ to the prover.

The prover and the verifier will compute $\hat{W}_o \circ l_0$ and the prover will send a polynomial $H_0 \circ l_0$ claimed to be

$$H_0(l_0(z)) := \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1'}} \hat{add}_0(l_0(z),\boldsymbol{b},\boldsymbol{c})(\hat{W}_1(\boldsymbol{b})+\hat{W}_1(\boldsymbol{c})) + \hat{mult}_0(l_0(z),\boldsymbol{b},\boldsymbol{c})(\hat{W}_1(\boldsymbol{b})\cdot\hat{W}_1(\boldsymbol{c}))$$

The verifier will now check that the equality $\psi(H_0 \circ l_0) = \psi(W_0 \circ l_0)$ holds by checking the equality in each of the at most $d \cdot s_0$ coefficients of the univariate polynomial.

Those polynomials are degree $d \cdot s_0$ polynomials, nonetheless there is a trick we can use to reduce them to degree d or less after the previous check has been done. We just use the polynomials because we want to prove their equality at point ξ. If q is a degree d irreducible polynomial over $\mathbb{F}_p$ and $GR(p^k,d) = \mathbb{Z}_{p^k}[X]/(q)$ then we can reduce the degree $d \cdot s_0$ polynomials modulo q. Even though the resulting polynomial won't evaluate to the same as the original one at all points, it will at point ξ, as ξ is a root of polynomial q.

This reduction needs to be done after the verification made by the verifier, or else the equality won't hold. If q is sparse enough, this reduction can be done in linear time and both V and P will do it for the polynomial $H_0 \circ l_0$. With this reduction, the prover time of the protocol will decrease in a factor of $O(d)$.

They will now proceed to do a Sumcheck Protocol with this polynomials, that is that in each round, the prover will be sending a degree d polynomial in z instead of a value.

During this sumcheck protocol, the verifier will be sending random challenges in $GR(p^k,d)$. Let $\boldsymbol{r_b}$ and $\boldsymbol{r_c}$ be the random challenges sent(both elements in $GR(p^k,d)^{s_1'}$).

The protocol will run until the last step, where the verifier will have to evaluate the right hand side of the equation. Nonetheless V doesn't have access to $\hat{W}_1'$ and therefore can't compute $\hat{W}_1'(\boldsymbol{r_b})$ and $\hat{W}_1'(\boldsymbol{r_c})$.

The prover will send the two evaluations to the verifier $c_b := \hat{W}_1'(\boldsymbol{r_b})$ and $c_c := \hat{W}_1'(\boldsymbol{r_c})$ and the verifier will check the last step of the Sumcheck with those values.

They will then proceed to prove the correctness of those evaluations.

In the classic sumcheck this would be done by using a line that goes through those two random values, and then use the composition with W_1. Nonetheless it's not always possible to find a vector polynomial with integer coefficients that goes through two specific values in $GR(p^k,d)^{s_1'}$. This is why we had to move from the classic GKR to the approach provided in [2].

In this approach we merge together both claims using a linear combination and doing both Sumchecks at once. We will therefore choose two random values in $GR(p^k,d)$. We will see in the soundness section that if α_1', β_1' are random values in $GR(p^k,d)$ then it's enough for the prover to show the correct evaluation of $\alpha_1' H_1(\boldsymbol{r_b}) + \beta_1' H_1(\boldsymbol{r_c})$ (H_1 defined the same way H_0 was defined) to prove the correctness of the evaluations of $H_1(\boldsymbol{r_b})$ and $H_1(\boldsymbol{r_c})$.

If we just did this, we would encounter two problems that would make the polynomials in which we want to apply the recursion have values in $GR(p^k,d)$ that are not ϕ images of values in $\mathbb{Z}_{p^k}^{s_1'}$. The first would be the multiplying random values α_1' and β_1'. The second would be the random values $\boldsymbol{r_b}$ and $\boldsymbol{r_c}$ evaluated in $\hat{add}_1$ and $\hat{mult}_1$.

We use a similar idea in both cases. For the first problem, thanks to Lemma 6.1 we can build two polynomials $\alpha_1, \beta_1 \in \mathbb{Z}_{p^k}^{<d}[X]$ such that $\alpha_1(\xi) = \alpha_1'$

and $\beta_1(\xi) = \beta_1'$. Instead of using the values, we will prove the equality as polynomials with coefficients in $\mathbb{Z}_{p^k}$.

For the second problem we will build two polynomial vectors l_{1b} and l_{1c} with coefficients in $\mathbb{Z}_{p^k}$ such that $l_{1b}(\xi) = \boldsymbol{r_b}$ and $l_{1c}(\xi) = \boldsymbol{r_c}$ and use those instead of the values.

Before the verifier sends the random challenges α_i and β_i, the prover will send the polynomials $\hat{W}_1'(l_{1b}(z))$ and $\hat{W}_1'(l_{1c}(z))$. The prover will also send the polynomials $\hat{W}_1(\boldsymbol{y}, l_{1b}(z))$ and $\hat{W}_1(\boldsymbol{y}, l_{1c}(z))$ for $\boldsymbol{y} \in GR(p^k, d)^{\frac{\log n}{2}}$ ($\hat{W}_1$ defined the same way it was done with $\hat{W}_0$). The verifier will have to check that the polynomial $\hat{W}_1' \circ l_{1*}$ comes from $\hat{W}_1(\boldsymbol{y}, l_{1*}(z))$.

Since we have that

$$\hat{W}_1'(l_{1*}(z)) = \phi\left(\begin{pmatrix} \check{W}_1(0,\cdots,0,0,l_{1*}(z)) \\ \check{W}_1(0,\cdots,0,1,l_{1*}(z)) \\ \vdots \\ \check{W}_1(1,\cdots,1,1,l_{1*}(z)) \end{pmatrix}\right)$$

then the verifier will simply have to check that

$$\sum_{i=0}^{n} \psi(\hat{W}_1(\cdot, l_{1*}(z)))(\boldsymbol{y})[i] = \psi(\hat{W}_1'(l_{1*}(z)))[\boldsymbol{y}]$$

for all $\boldsymbol{y} \in \{0,1\}^{\log n}$. This comes from the fact that both the left and right hand terms equal to $W_1(\boldsymbol{y}, l_{1*}(z))$. With that the verifier can ensure that both $\hat{W}_1' \circ l_{1*}$ and $\hat{W}_1(\boldsymbol{y}, l_{1*}(z))$ come from the same W_1.

The verifier will now need to send another random vector polynomial to fill the first $\log n$ empty variables. V will send to P the vector polynomial $l_1 : \mathbb{Z}_{p^k}^n \to \mathbb{Z}_{p^k}^{\log n}$.

We will end up proving the correctness of the evaluations of H_1 by proving the following equality:

$$\begin{aligned} &\psi(\alpha_1(z)H_1(l_1(z)l_{1b}(z) + \beta_1(z)H_1(l_1(z), l_{1c}(z)) = \\ &\psi(\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_1'}} [\alpha_1 \hat{add}_1(l_1(z), l_{1b}(z), \boldsymbol{b}, \boldsymbol{c}) + \\ &\beta_1 \hat{add}_1(l_1(z), l_{1c}(z), \boldsymbol{b}, \boldsymbol{c})] \cdot (\hat{W}_2'(\boldsymbol{b}) + \hat{W}_2'(\boldsymbol{c})) + \\ &[\alpha_1 \hat{mult}_1(l_1(z), l_{1b}(z), \boldsymbol{b}, \boldsymbol{c}) + \\ &\beta_1 \hat{mult}_1(l_1(z), l_{1c}(z), \boldsymbol{b}, \boldsymbol{c})] \cdot (\hat{W}_2'(\boldsymbol{b}) \cdot \hat{W}_2'(\boldsymbol{c}))) \end{aligned} \tag{8}$$

The protocol will go as follows. If we define

$$\begin{aligned} H_i(\boldsymbol{z}) := &\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_{i+1}'}} \hat{add}_i(\boldsymbol{z}, \boldsymbol{b}, \boldsymbol{c})(\hat{W}_{i+1}'(\boldsymbol{b}) + \\ &\hat{W}_{i+1}(\boldsymbol{c})) + \hat{mult}_i(\boldsymbol{z}, \boldsymbol{b}, \boldsymbol{c})(\hat{W}_{i+1}'(\boldsymbol{b}) \cdot \hat{W}_{i+1}(\boldsymbol{c})) \end{aligned}$$

then the prover will send to the verifier the polynomial $G(z) = \alpha_1(z)H_1(l_1(z), l_{1b}(z)) + \beta_1(z)H_1(l_1(z), l_{1c}(z))$. The verifier will now check that $\psi(G(z)) = \psi(\alpha_1(z)\hat{W}_1(l_1(z), l_{1b}(z)) + \alpha_1(z)\hat{W}_1(l_1(z), l_{1c}(z)))$ (checking the equality in each coefficient) and that $G(\xi) = \alpha_i(\xi)c_b + \beta_i(\xi)c_c$.

After this checks are done, the prover and verifier can now reduce again the degree of the univariate polynomial $G(z)$ by computing modulo q.

They will then proceed to prove the correctness of sum 8 using the Sumcheck protocol. At the end we will face the same problem of needing the evaluation of the polynomial which will be solved the same way as the first time.

They will recurse the protocol until the claim is the evaluations of the first layer, the input layer. Since the verifier has access to the input values, V can compute $\hat{W}_m'(\boldsymbol{r_b})$ and $\hat{W}_m'(\boldsymbol{r_c})$ to check the equality and finish the last Sumcheck.

Even though this protocol may seem very complex, using the ideas presented in the Libra paper [9] the prover can run it with just $O(nS)$ time, that means quasilinear in the size of the circuit, with just an overhead factor of the security parameter.

6.2 Completeness and soundness analysis

We will see first the protocol completeness, that is that if the prover is honest, the verifier will always accept. To do so we need to ensure that all checks from the verifier pass.

The first check the verifier does is $\psi(H_0 \circ l_0) = \psi(\hat{W}_0 \circ l_0)$ by checking the equality of the image ψ of the polynomial coefficients. In this equation and all the ones we encounter throughout the protocol, at most 3 different polynomials are multiplied together making it so that we only need a degree 3 RMFE.

To prove the correctness of the previous equality we will present this simple lemma. This lemma will also help us prove the correctness of the other equality checks we encounter throughout the protocol of the form

$$\psi(\alpha_i(z)\hat{W}_i(l_{ib}(z)) + \beta_i(z)\hat{W}_i(l_{ic}(z))) = \psi(G_i(\boldsymbol{z}))$$

Lemma 6.2. *With the construction presented in Protocol 6, the equality*

$$\psi(H_i \circ l) = \psi(\hat{W}_i \circ l)$$

always holds.

Proof. We have by construction that if

$$\begin{aligned} \tilde{H}_i(\boldsymbol{z}) := &\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s_{i+1}'}} \tilde{add}_i(\boldsymbol{z}, \boldsymbol{b}, \boldsymbol{c})(\tilde{W}_{i+1}'(\boldsymbol{b}) + \\ &\tilde{W}_{i+1}'(\boldsymbol{c})) + \tilde{mult}_i(\boldsymbol{z}, \boldsymbol{b}, \boldsymbol{c})(\tilde{W}_{i+1}'(\boldsymbol{b}) \cdot \tilde{W}_{i+1}'(\boldsymbol{c})) \end{aligned}$$

then

$$\tilde{W}_i(\boldsymbol{z}) = \tilde{H}_i(\boldsymbol{z})$$

which also implies that

$$\tilde{W}_i(l_i(z)) = \tilde{H}_i(l_i(z))$$

Since ϕ is a degree 3 RMFE and we have at most 3 polynomials multiplied together in each side of the equation, we will have

$$\begin{gathered}\psi(\hat{W}_i(l_i(z))) = \psi(\phi(\tilde{W}_i(l_i(z)))) = \\ \psi(\sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s'_{i+1}}} \phi(\tilde{add}_i(l_i(z),\boldsymbol{b},\boldsymbol{c}))\cdot \\ (\phi(\tilde{W}'_{i+1}(\boldsymbol{b})) + \phi(\tilde{W}'_{i+1}(\boldsymbol{c})))+ \\ \phi(\tilde{mult}_i(l_i(z),\boldsymbol{b},\boldsymbol{c}))(\phi(\tilde{W}'_{i+1}(\boldsymbol{b}))\cdot\phi(\tilde{W}'_{i+1}(\boldsymbol{c})))) = \\ \psi(H_i(l_i(z)))\end{gathered}$$

which is what we wanted to prove:

$$\psi(H_i \circ l) = \psi(\hat{W}_i \circ l)$$

□

Lemma 6.3. *With the construction presented in Protocol 6, the equality*

$$\psi(\alpha_i(z)\hat{W}_i(l_{ib}(z)) + \beta_i(z)\hat{W}_i(l_{ic}(z))) = \psi(G_i(\boldsymbol{z}))$$

always holds.

Proof. By construction we have

$$G_i(z) = \alpha_i(z)H_i(l_{ib}(z)) + \beta_i(z)H_i(l_{ic}(z))$$

Using Lemma 6.2 we also have $\psi(H_i \circ l_{ib}) = \psi(\hat{W}_i \circ l_{ib})$ and $\psi(H_i \circ l_{ic}) = \psi(\hat{W}_i \circ l_{ic})$.

Since ψ is a linear function and α_i and β_i polynomials with integer coefficients and using Lemma 6.2 we have

$$\begin{gathered}\psi(\alpha_i(z)\hat{W}_i(l_{ib}(z)) + \beta_i(z)\hat{W}_i(l_{ic}(z))) = \\ \alpha_i(z)\psi(\hat{W}_i(l_{ib}(z))) + \beta_i(z)\psi(\hat{W}_i(l_{ic}(z))) = \\ \alpha_i(z)\psi(H_i(l_{ib}(z))) + \beta_i(z)\psi(H_i(l_{ic}(z))) = \psi(G_i(z))\end{gathered}$$

□

Theorem 6.4. *If the prover is honest, then when running Protocol 6 the verifier will always accept, and therefore the protocol is complete.*

Proof. The verifier will reject in two cases:

- The equality $\psi(H_0 \circ l_0) = \psi(\hat{W}_0 \circ l_0)$ doesn't hold.
- The equality $\psi(\alpha_i\hat{W}_i(l_{ib}(\boldsymbol{z})) + \beta_i\hat{W}_i(l_{ic}(\boldsymbol{z}))) = \psi(G_i(\boldsymbol{z}))$ doesn't hold.
- One of the Sumcheck protocols fails at some point.

We saw in Lemma 6.2 and Lemma 6.3 that the first and second case won't happen as long as the prover is honest. The third case won't happen either as we saw that the Sumcheck protocol is complete in a previous section.

Therefore Protocol 6 is complete. □

The only thing left for us to prove is the soundness of the protocol.

Lemma 6.5. *If $\psi(H_i \circ l_i) = \psi(\hat{W}_i \circ l_i)$ then $\psi(H_i) = \psi(\hat{W}_i)$.*

Proof. This comes from the fact that both H_i and $\hat{W}_i$ are multilinear polynomials of at most M_s variables, and therefore we just need equality in 2^{M_s} points from an interpolation subset which we have since the domain of l_i is $GR(p^k, d)$ and this domain has p^d elements in it's interpolation subset. □

The previous lemma provides a proof of 0 soundness error with the conversion from a multilinear polynomial to a univariate polynomial. We now need to check how does soundness behave when doing a linear combination of two claims.

Lemma 6.6. *Given α_i and β_i random polynomials in $\mathbb{Z}_{p^k}^{<d}[X]$ if*

$$\psi(\alpha_i(z)\hat{W}_i(l_{ib}(\boldsymbol{z})) + \beta_i(z)\hat{W}_i(l_{ic}(\boldsymbol{z}))) = \psi(G_i(\boldsymbol{z}))$$

then with soundness error of $O(p^{-k})$ we have that

$$\psi(\hat{W}_i) = \psi(H_i)$$

Proof. Using linearity of ψ the first equation can be written as

$$\alpha_i(z)\psi(\hat{W}_i(l_{ib}(\boldsymbol{z}))) + \beta_i(z)\psi(\hat{W}_i(l_{ic}(\boldsymbol{z}))) = \psi(G_i(\boldsymbol{z}))$$

which using Lemma 6.5 we can reduce the claim to prove that given $\alpha_i(z)$ and $\beta_i(z)$ random polynomials

$$\alpha_i(z)\cdot a + \beta_i(z)\cdot b = \alpha_i(z)\cdot c + \beta_i(z)\cdot d$$

then with a soundness error of p^{-k}

$$a = c \quad b = d$$

If we assume $a \neq c$, then we will have

$$\alpha_i(z)(a-c) = \beta_i(z)(d-b)$$

In particular the equality will hold when evaluating the polynomial at point ξ. Since the evaluation of polynomials in $\mathbb{Z}_{p^k}^{<d}[X]$ is a bijective mapping between $\mathbb{Z}_{p^k}^{<d}[X]$ and $GR(p^k, d)$, we can imagine the equality with α_i and β_i being random elements in $GR(p^k, d)$.

$$\alpha_i(a-c) = \beta_i(d-b)$$

We can transform this to an evaluation of a polynomial $f(x,y) = x(a-c) + y(b-d)$ at two random points in $GR(p^k,d)$ which is zero with probability p^{-d} thanks to the Galois Rings version of the Schwartz Zipple Lemma presented in Corollary 2.3.1.

That concludes the proof and gives a soundness error of p^{-d} to this part of the protocol. □

6.3 Complexity analysis

We will do a naive complexity analysis like we did on the previous sections and then we will see how we can improve the prover complexity to be quasilinear in the circuit size by using ideas presented in the Libra paper [9].

Observe that during the protocol the prover will be sending to the verifier the composition $\hat{W}_i \circ l_i$ with l_i a degree d polynomial, therefore the composition will be a univariate polynomial of degree d^2 in $GR(p^k,d)$. Nonetheless this will only have to be done once per layer, as the sumcheck can be done with a degree d polynomial in z. This reduction can be done in $O(d)$ time if the quotient polynomial is chosen to be sparse.

We can now imagine doing a Sumcheck with a degree d polynomial as doing d Sumchecks together, and therefore the complexity will be the one of doing one sumcheck multiplied by d.

Taking into account that $O(d) = O(n)$ (that is because $d \approx 3n$) and assuming that $O(s'_i) = O(M'_s) = O(M_s)$, each Sumcheck of the protocol can be done in time

$$O(ds'_i 2^{2s'_i} T) = O(M_s 2^{2M_s - \log n} T) = O(M_s 2^{2M_s} \frac{T}{n})$$

Then we will do m Sumchecks of $2s_i$ variables each with those degree d polynomials and using a degree 3 RMFE. This can be done naively in time $O(mM_s 2^{2M_s} \frac{T}{n})$.

Observe that we can also compute $O(T)$ since we know that the functions are multilinear polynomials of at most $2M'_s$ variables and therefore at most $2^{2M'_s}$ monomials. Taking into account that those monomials are in $GR(p^k,d)$, we end up with $O(T) = O(2^{2M_s} n \log n)$ and that gives us a total prover time of

$$O(mM_s 2^{4M_s} \log n)$$

On the verifier side, V will run a total of m Sumchecks with each sumcheck having $2s_i$ variables and sending polynomials of degree d in each round. The verifier will also have to evaluate the polynomials add_i and $mult_i$ to check on the last step the equality which can be done in time $O(T)$. V will also have to compute once each round the ψ image of a degree d^2 polynomial which can be done in $O(d^4)$ time. The verifier will therefore have to do d products of elements in $GR(p^k,d)$ in each round, $2s'_i$ rounds and an evaluation at the last round ending up with a total verifier complexity of

$$O(m(n \log n(n+T) + n^4))$$

which is much faster than computing the whole circuit. Compared to the classic sumcheck, the verifier will generally have an overhead factor of $O(n \log n)$.

Finally the proof size will be m layers, each layer having a sumcheck and each sumcheck having $2s'_i$ rounds. In each round a polynomial of degree $O(d)$ in $GR(p^k,d)$ is sent giving us a total proof size of

$$O(mM_s d)$$

We can also consider S to be the size of the circuit and we would then have $O(2^M_s) = O(S)$.

We can see all costs summarized in the following table

Proof size	$O(mM_s d)$
V time	$O(m(n \log n(n+T) + n^4))$
P time	$O(mM_s S^2 T \log n)$

Table 4: *Complexity analysis Protocol 6*

The prover time is really large, being quadratic on the circuit size multiplied by the time to evaluate a degree $2s_i$ polynomial. Nonetheless we can greatly reduce it to a quasilinear time in the circuit size by using the ideas presented in the Libra paper [9].

6.4 Quasilinear time sumcheck for GKR functions in $GR(p^k,d)$

If we now think to apply the quasilinear time sumcheck presented in section 4, we would still not end up with a quasilinear time sumcheck in the layer size of the GKR, since we still have multilinear polynomials of $2s'_i$ variables. We would therefore have an algorithm close to quadratic in the circuit size.

Nonetheless, the same way it was done in the Libra paper [9], we can use the fact that if we have

$$\check{W}_i(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c} \in \{0,1\}^{s_{i+1}}} \check{add}_i(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_{i+1}(\boldsymbol{b}) + \check{W}_{i+1}(\boldsymbol{c})) + \check{mult}_i(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\check{W}_{i+1}(\boldsymbol{b}) \cdot \check{W}_{i+1}(\boldsymbol{c})) \quad \forall \boldsymbol{z} \in \mathbb{Z}_{p^k}^{s_i}$$

then both $\check{add}_i(z,x,y)$ and $\check{mult}_i(z,x,y)$ are the multilinear extension of very sparse arrays with $O(2^{s_i})$ (out of $O(2^{s_i+2s_{i+1}})$ possible elements) nonzero elements.

Once we send this functions to the Galois Ring via ϕ, we will be batching them together in groups

of n, and therefore, since ϕ is linear and $\phi(\mathbf{0}) = 0$, we will have around $O(2^{s_i})$ nonzero elements in $\hat{add}_i(z,x,y)$ and $\hat{mult}_i(z,x,y)$ out of the $O(2^{s_i+2s'_{i+1}})$ possible elements.

We will first consider the case with no addition gates and rewrite the sumcheck equation in the following way:

$$\hat{W}_i(\boldsymbol{z}) = \sum_{\boldsymbol{b},\boldsymbol{c}\in\{0,1\}^{s'_{i+1}}} \hat{mult}_i(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})(\hat{W}_{i+1}(\boldsymbol{b})\cdot\hat{W}_{i+1}(\boldsymbol{c})) =$$
$$= \sum_{\boldsymbol{b}\in\{0,1\}^{s'_{i+1}}} \hat{W}_{i+1}(\boldsymbol{b}) \sum_{\boldsymbol{c}\in\{0,1\}^{s'_{i+1}}} \hat{mult}_i(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})\cdot\hat{W}_{i+1}(\boldsymbol{c}) =$$
$$= \sum_{\boldsymbol{b}\in\{0,1\}^{s'_{i+1}}} \hat{W}_{i+1}(\boldsymbol{b})\hat{h}_{i+1}(z,\boldsymbol{c})$$

with

$$\hat{h}_{i+1}(z,\boldsymbol{c}) = \sum_{\boldsymbol{c}\in\{0,1\}^{s'_{i+1}}} \hat{mult}_i(\boldsymbol{z},\boldsymbol{b},\boldsymbol{c})\cdot\hat{W}_{i+1}(\boldsymbol{c})$$

Then we will do two Sumchecks, each of them in quasilinear time.

Phase one With this first sumcheck we prove the equality on the b variable

$$\hat{W}_i(\boldsymbol{z}) = \sum_{\boldsymbol{b}\in\{0,1\}^{s'_{i+1}}} \hat{W}_{i+1}(\boldsymbol{b})\hat{h}_{i+1}(z,\boldsymbol{c})$$

Initializing the bookkeeping table for $\hat{W}_{i+1}$ in $O(2^{s'_{i+1}}n\log n) = O(2^{s_{i+1}}\sqrt{n}\log n)$ time is easy since ϕ is a linear function and the images of $\tilde{W}_{i+1}$ in $\{0,1\}^{s'_{i+1}}$ are the ones given by the array from which the multilinear extension was generated from. The images of $\hat{W}_{i+1}$ in the hypercube will just be the ϕ images of the computed values. To send those images to the ring we would need $O(2^{s'_{i+1}}n\log n) = O(2^{s_{i+1}}\sqrt{n}\log n)$ time.

Once we have the bookkeeping table we for $\hat{W}_{i+1}$ we can find the evaluations needed for the Sumcheck in $O(2^{s_{i+1}}\sqrt{n}\log n)$ time the same way it was done in Algorithm 1.

Computing the bookkeeping table for $\hat{h}_{i+1}(z,\boldsymbol{c})$ in quasilinear time is not that easy. The two problems we face here are:

1. We need 2^{2s_i} evaluations of $\hat{mult}_i$. This problem is solved in Libra using the sparsity of the array it comes from.
2. When computing the ϕ image of the batched, we will end up with 2^{s_i} nonzero elements, not $2^{s'_i}$ which might be a problem. This will be solved later on.

Let's see first how we adapt Libra's solution to our case first and then we will solve the second problem

To do so we rewrite this polynomial in the following way

$$\hat{h}_{i+1}(z,\boldsymbol{b}) = \sum_{\boldsymbol{c}\in\{0,1\}^{s'_{i+1}}} \hat{mult}_i(l(z),\boldsymbol{b},\boldsymbol{c})\hat{W}_{i+1}(\boldsymbol{c}) =$$
$$\sum_{\boldsymbol{s},\boldsymbol{c}\in\{0,1\}^{s'_{i+1}}} I(l(z),\boldsymbol{s})\hat{mult}_i(s,\boldsymbol{b},\boldsymbol{c})\hat{W}_{i+1}(\boldsymbol{c})$$

where we have

$$I(\boldsymbol{w},\boldsymbol{s}) = \prod_{i=1}^{s'_{i+1}}((1-w_i)(1-s_i)+w_is_i)$$

the identity polynomial.

The equality is given trivially by uniqueness of multilinear extensions.

Now we have that the images of $mult_i(\boldsymbol{s},\boldsymbol{b},\boldsymbol{c})$ on $\{0,1\}^{s_i\times s^2_{i+1}}$ can be trivially computed in time $O(2^{s_i})$ as they come come from a sparse array with only 2^{s_i} nonzero elements. Therefore we can also compute the images of $\tilde{mult}_i(\boldsymbol{s},\boldsymbol{b},\boldsymbol{c})$ on $\{0,1\}^{s_i\times s'^2_{i+1}}$ in time $O(2^{s_i})$. This function $\tilde{mult}_i$ will still have $O(2^{s_i})$ nonzero elements, not $O(2^{s'_i})$, as each batched parameter evaluated has at most one nonzero element out of all the elements. Therefore:

$$Im(\tilde{mult}_i|_{\{0,1\}^{s_i\times s'^2_{i+1}}}) = \{\mathbf{0}, e_1, \cdots, e_n\}$$

where e_i is the vector with all zeros but in position i where there is a 1.

If we where now to send the images of $\tilde{mult}_i$ to the Galois Ring trivially, we would have to evaluate ϕ a total of 2^{s_i} times, since it has that many nonzero elements. This would give us an overall complexity of $O(2^{s_i}n\log n)$ which has too big of an overhead for the protocol to be useful.

This problem can be solved by using the fact that $|Im(\tilde{mult}_i|_{\{0,1\}^{s_i\times s'^2_{i+1}}})| = n+1$ and therefore we can precompute the ϕ images of those functions in time $O(n^2\log n)$ and then use them to compute the ϕ images of the images of $\tilde{mult}_i$ in time $O(2^{s_i})$. This would solve problem 2 mentioned previously.

We already have the images of W_{i+1} from the previous computation, so we now just need the images of $I(l(z),\boldsymbol{s})$ for $\boldsymbol{s}\in\{0,1\}^{s_i}$. We can do this with the procedure $\boldsymbol{L}$ presented in Algorithm 2. This bookkeeping table will store polynomials on z with integer coefficients (since l had integer coefficients) and therefore the algorithm will run in $O(2^{s_i}n\log n)$.

With all this we can compute the bookkeeping table $\boldsymbol{A}_{h_l}$ in time $O(2^{s_i}n\log n)$ as we know that there are at most $O(2^{s_i})$ elements in $\{0,1\}^{s_i\times s'^2_{i+1}}$ such that $mult(\boldsymbol{s},\boldsymbol{b},\boldsymbol{c})$ is non-zero and each element in $\boldsymbol{L}$ will be a polynomial of degree less than d with coefficients in $\mathbb{Z}_{p^k}$.

Algorithm 2 $\boldsymbol{A}_{h_l} \leftarrow$ Initialize_PhaseOne$(\hat{W}_{i+1}, \boldsymbol{A}_{f_3}, l(z))$

Input: Multilinear $\hat{W}_{i+1} \in GR(p^k,d)[X_1, \cdots, X_{s'_{i+1}}]$, initial bookkeeping table $\boldsymbol{A}_{\hat{W}_{i+1}}$, random polynomial vector $l : GR(p^k,d) \to GR(p^k,d)^{s_i}$.
Output: Bookkeeping table $\boldsymbol{A}_{h_{i+1}}$

procedure $\boldsymbol{L}(z) \leftarrow Precompute(l(z))$
  Set $\boldsymbol{L}[0](z) = 1$
  for $i = 0, \cdots, s_i - 1$ **do**
    for $\boldsymbol{x} \in \{0,1\}^i$ **do**
      $\boldsymbol{L}[\boldsymbol{x}, 0](z) = \boldsymbol{L}[\boldsymbol{x}](z) \cdot (1 - l_i(z))$
      $\boldsymbol{L}[\boldsymbol{x}, 1](z) = \boldsymbol{L}[\boldsymbol{x}](z) \cdot l_i(z)$
$\forall \boldsymbol{b} \in \{0,1\}^{s'_{i+1}}$, set $\boldsymbol{A}_{h_{i+1}}[\boldsymbol{b}](z) = 0$
for every $(\boldsymbol{s}, \boldsymbol{b}, \boldsymbol{c}) \in \{0,1\}^{s_i \times s'^2_{i+1}}$ such that $\hat{mult}(\boldsymbol{s}, \boldsymbol{b}, \boldsymbol{c})$ is non-zero **do**
  $\boldsymbol{A}_{h_{i+1}}[\boldsymbol{b}](z) = \boldsymbol{A}_{h_{i+1}}[\boldsymbol{b}](z) + \boldsymbol{L}[\boldsymbol{s}](z) \cdot \hat{mult}(\boldsymbol{s}, \boldsymbol{b}, \boldsymbol{c}) \cdot \boldsymbol{A}_{W_{i+1}}[\boldsymbol{c}]$
return $\boldsymbol{A}_{h_l}$

Algorithm 3 $\boldsymbol{A}_{h_l} \leftarrow$ Initialize_PhaseTwo$(f_1, l(z), r_b)$

Input: Multilinear $f_1 \in GR(p^k,d)[X_1, \cdots, X_{l'}]$ random polynomial vector $l : GR(p^k,d) \to GR(p^k,d)^{l'}$ and random $r_b \in GR(p^k,d)^{l'}$.
Output: Bookkeeping table $\boldsymbol{A}_{f_1}$

$\boldsymbol{L}(z) \leftarrow$ Precompute$(l(z))$
$\boldsymbol{R_b} \leftarrow$ Precompute(R_b)
$\forall y \in \{0,1\}^{l'}$, set $\boldsymbol{A}_{f_1}[y](z) = 0$.
for every (s, x, y) such that $f_1(s, x, y)$ is non-zero **do**
  $\boldsymbol{A}_{f_1}[y](z) = \boldsymbol{A}_{f_1}[y](z) + \boldsymbol{L}[s](z) \cdot \boldsymbol{R_b}[x] \cdot f_1(s, x, y)$
return $\boldsymbol{A}_{f_1}$

Phase two With phase one we have the tools to prove in $O(2^{M'_s} n)$ time the correctness of the sum

$$\sum_{x \in \{0,1\}^{M'_s}} \hat{f}_2(x) \hat{h}_l(x, z)$$

by using the the bookkeeping tables $\boldsymbol{A}_{f_2}$ and $\boldsymbol{A}_{h_l}$. Now we just need to prove the correctness of

$$\hat{h}_l(r_b, z) = \sum_{y \in \{0,1\}^{M'_s}} \hat{f}_1(l(z), r_b, y) \hat{f}_3(y)$$

for the $r_b \in GR(p^k,d)^{M'_s}$ chosen by the verifier. To do this, we just need to compute the bookkeeping tables for each function in the sum.

Computing $\boldsymbol{A}_{f_3}$ is easy as it's again the multilinear extension of an array and therefore we can do the same we did for f_2.

We will now need to compute the bookkeeping table that stores $f_1(l(z), r_b, y)$ for all $y \in \{0,1\}^{M'_s}$. To do this we will rewrite

$$f_1(l(z), r_b, y) = \sum_{(s,x) \in \{0,1\}^{M'_s}} I(l(z), s) I(r_b, x) f_1(s, x, y)$$

which is true by uniqueness of the multilinear extension.

Now we just need to compute $\boldsymbol{R}_b$ and $\boldsymbol{L}(z)$, the arrays that have the evaluations of $I(r_b, x) \forall x \in \{0,1\}^{M'_s}$ and $I(l(z), s) \forall s \in \{0,1\}^{M'_s}$ respectively. Computing $\boldsymbol{L}(z)$ has already been done in Algorithm 2 and we can compute $\boldsymbol{R}_b$ the same way.

Once we have all this, computing the bookkeeping table for f_1 is easy and can be seen in Algorithm 3

With all the bookkeeping tables, we can run Algorithm 1 to compute in time $O(2^{M'_s} n \log n) = O(2^{M_s} \log n)$ all the values needed to later run the sumcheck in $O(2^{M'_s} n \log n) = O(2^{M_s} \log n)$ time.

Complexity analysis With this new version of the protocol we will have reduced greatly the prover time for each sumcheck down to $O(2^{M_s} n)$ giving us an overall prover complexity of the GKR protocol of

$$O(m 2^{M_s} n)$$

This new complexity reduces a factor of $O(\frac{2^{3M_s} M_s \log^2 n}{n})$ the cost of the naive approach and has an overhead factor in comparison to the classic GKR over Fields of $O(n)$, the security parameter.

7 Discussion

In this section, we discuss some potential improvements for Sailor and some ideas on how to continue the construction of protocols over rings.

Improving GKR prover time: The current prover time for the GKR over $\mathbb{Z}_{p^k}$ is $O(nS)$, which has a $O(n)$ overhead factor in respect to the version over fields.

This $O(n)$ factor could be reduced to $O(\log n)$ if a faster way to evaluate ϕ and ψ from the RMFE is found. The current method is using the functions linearity and after computing the image of the canonical basis, a matrix vector product is needed. This means that after the precomputing phase, one image can be computed in $O(n^2)$ time. Matrix vector product optimization is a well known problem and has been reduced to $O(n^2/(\varepsilon \log n)^2)$ in [22], nonetheless it seems like removing this quadratic factor on the numerator might not be possible.

Another approach could be to work over the Galois Ring in a non canonical basis, such that the matrix representation of the RMFE in that basis was diagonal. This would enable a linear computation of RMFE images, but this will turn element-wise multiplication more expensive as we will have to work in this basis during the protocol. It will also increase the ψ computation complexity, although that might not be an issue as this is at most computed $O(nm)$ times throughout the protocol.

Finally another approach would be to find an RMFE with a very sparse ϕ matrix representation. Since it's construction is fairly complex, finding a sparse one might be challenging.

Improving GKR verifier time: Even though in secure computation protocols prover time and succinctness are prioritized over the verifier time, it is also important to try to reduce it as much as possible. Our protocol introduces a big overhead when it comes to this aspect, as the verifier is not able to use the batching of the parameters to reduce a $O(n)$ factor it's complexity, like the prover can do. It is left as a future problem to reduce this verifier complexity overhead to a minimum.

Other ZKP protocols over $\mathbb{Z}_{2^k}$: While bilinear pairings might not be possible to construct over $\mathbb{Z}_{2^k}$ or Galois Rings, and therefore making it challenging to build some of the most used protocols, it is possible to have a Galois Ring version of the discrete logarithm.

If we have $GR(2^k, d)$ such that $2^d - 1$ is a prime number (known as the Mersenne primes), then the set $S = \{1, \xi, \xi^2, \cdots, \xi^{2^d-2}\}$ is a multiplicative subgroup of order $2^d - 1$. This might give us a discrete logarithm for large primes, nonetheless, it's structure coming from the Galois Rings might make this assumption weaker in comparison to any construction over elliptic curves.

With this assumption it is possible to construct a polynomial commitment scheme over $GR(p^k, d)$ with a similar structure as the IPA Based presented in Bulletproof [23].

On the other side, many cryptographic protocols rely on randomness to provide soundness, and this is a major source of problems when working with RMFE as we have seen in this paper. If the randomness is chosen in the original ring, then soundness is generally lost, and if it is chosen in the Galois Ring, then we can't use the RMFE to bring back any values to the original ring. In Sailor a solution to this problem is presented by choosing the randomness as a polynomial with coefficients in the original ring such that it goes through the desired point in the Galois Ring. This can be easily extrapolated to many scenarios and might help in future protocol constructions over rings of the form $\mathbb{Z}_{p^k}$.

References

[1] Carsten Lund et al. "Algebraic Methods for Interactive Proof Systems". In: *J. ACM* 39.4 (Oct. 1992), pp. 859–868. ISSN: 0004-5411. DOI: 10.1145/146585.146605. URL: https://doi.org/10.1145/146585.146605.

[2] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. *Sumcheck Arguments and their Applications*. Cryptology ePrint Archive, Paper 2021/333. https://eprint.iacr.org/2021/333. 2021. URL: https://eprint.iacr.org/2021/333.

[3] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *J. ACM* 62.4 (Sept. 2015). ISSN: 0004-5411. DOI: 10.1145/2699436. URL: https://doi.org/10.1145/2699436.

[4] Ignacio Cascudo et al. *Amortized Complexity of Information-Theoretically Secure MPC Revisited*. Cryptology ePrint Archive, Paper 2018/429. https://eprint.iacr.org/2018/429. 2018. URL: https://eprint.iacr.org/2018/429.

[5] Ignacio Cascudo and Jaron Skovsted Gundersen. *A Secret-Sharing Based MPC Protocol for Boolean Circuits with Good Amortized Complexity*. Cryptology ePrint Archive, Paper 2020/162. https://eprint.iacr.org/2020/162. 2020. URL: https://eprint.iacr.org/2020/162.

[6] Ronald Cramer, Matthieu Rambaud, and Chaoping Xing. *Asymptotically-Good Arithmetic Secret Sharing over $\mathbb{Z}/(p^l\mathbb{Z})$ with Strong Multiplication and Its Applications to Efficient MPC*. Cryptology ePrint Archive, Paper 2019/832. https://eprint.iacr.org/2019/832. 2019. URL: https://eprint.iacr.org/2019/832.

[7] Daniel Escudero, Chaoping Xing, and Chen Yuan. *More Efficient Dishonest Majority Secure Computation over $\mathbb{Z}_{2^k}$ via Galois Rings*. Cryptology ePrint Archive, Paper 2022/815. https://eprint.iacr.org/2022/815. 2022. URL: https://eprint.iacr.org/2022/815.

[8] Mark Abspoel et al. *Asymptotically Good Multiplicative LSSS over Galois Rings and Applications to MPC over Z/p^kZ*. Cryptology ePrint Archive, Paper 2020/1256. https://eprint.iacr.org/2020/1256. 2020. URL: https://eprint.iacr.org/2020/1256.

[9] Tiancheng Xie et al. *Libra: Succinct Zero-Knowledge Proofs with Optimal Prover Computation*. Cryptology ePrint Archive, Paper 2019/317. https://eprint.iacr.org/2019/317. 2019. URL: https://eprint.iacr.org/2019/317.

[10] Shuo Chen et al. *Verifiable Computing for Approximate Computation*. Cryptology ePrint Archive, Paper 2019/762. https://eprint.iacr.org/2019/762. 2019. URL: https://eprint.iacr.org/2019/762.

[11] Nikolay Vassiliev and O. Kanzheleva. "Polynomial Interpolation over the Residue Rings Z n". In: *Journal of Mathematical Sciences* 209 (Sept. 2015), pp. 845–850. DOI: 10.1007/s10958-015-2531-1.

[12] Eduardo Soria-Vazquez. *Doubly Efficient Interactive Proofs over Infinite and Non-Commutative Rings*. Cryptology ePrint Archive, Paper 2022/587. https://eprint.iacr.org/2022/587. 2022. URL: https://eprint.iacr.org/2022/587.

[13] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. *Rinocchio: SNARKs for Ring Arithmetic*. Cryptology ePrint Archive, Paper 2021/322. https://eprint.iacr.org/2021/322. 2021. URL: https://eprint.iacr.org/2021/322.

[14] Bryan Parno et al. "Pinocchio: Nearly practical verifiable computation". In: *Communications of the ACM* 59.2 (2016), pp. 103–112.

[15] Ronald Cramer et al. "SPDZ2k: Efficient MPC mod 2^k for Dishonest Majority". In: *Annual International Cryptology Conference. Springer*. 2018, pp. 769–798.

[16] Carsten Baum et al. *Appenzeller to Brie: Efficient Zero-Knowledge Proofs for Mixed-Mode Arithmetic and $\mathbb{Z}_{2^k}$*. Cryptology ePrint Archive, Paper 2021/750. https://eprint.iacr.org/2021/750. 2021. DOI: 10.1145/3460120.3484812. URL: https://eprint.iacr.org/2021/750.

[17] Carsten Baum et al. *Moz$\mathbb{Z}_{2^k}$arella: Efficient Vector-OLE and Zero-Knowledge Proofs Over $\mathbb{Z}_{2^k}$*. Cryptology ePrint Archive, Paper 2022/819. https://eprint.iacr.org/2022/819. 2022. DOI: 10.1007/978-3-031-15985-5_12. URL: https://eprint.iacr.org/2022/819.

[18] Fuchun Lin, Chaoping Xing, and Yizhou Yao. *More Efficient Zero-Knowledge Protocols over $\mathbb{Z}_{2^k}$ via Galois Rings*. Cryptology ePrint Archive, Paper 2023/150. https://eprint.iacr.org/2023/150. 2023. URL: https://eprint.iacr.org/2023/150.

[19] Z. Wan. *Lectures On Finite Fields And Galois Rings*. World Scientific Publishing Company, 2003. ISBN: 9789813102262. URL: https://books.google.com/books?id=F_A7DQAAQBAJ.

[20] M.F. Atiyah and I.G. MacDonald. *Introduction To Commutative Algebra*. Addison-Wesley series in mathematics. Avalon Publishing, 1994. ISBN: 9780813345444. URL: https://books.google.com/books?id=HOASFid4x18C.

[21] Daniel Escudero et al. *Degree-D Reverse Multiplication-Friendly Embeddings: Constructions and Applications*. Cryptology ePrint Archive, Paper 2023/173. https://eprint.iacr.org/2023/173. 2023. URL: https://eprint.iacr.org/2023/173.

[22] Richard Williams. "Matrix-vector multiplication in sub-quadratic time (some preprocessing required)". In: Jan. 2007, pp. 995–1001. DOI: 10.1145/1283383.1283490.

[23] Benedikt Bünz et al. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.