Honest Majority Multiparty Computation over Rings with Constant Online Communication

Minghua Zhao*
zhaominghua@fmsh.com.cn
Shanghai Fudan Microelectronics Group Company Limited
Shanghai, China

ABSTRACT

Multiparty computation (MPC) over rings such as $\mathbb{Z}_{2^{64}}$ has received a great deal of attention recently due to its ease of implementation and attractive performance. We assume there are n parties and the adversary corrupts t of them, where $2t < n$. In this work, we leverage RMFEs (Reverse Multiplication Friendly Embeddings) to build an MPC protocol over $\mathbb{Z}_{p^k}$ with constant online communication. An (l, d)-RMFE packs l elements from $\mathbb{Z}_{p^k}$ into an element in $GR(p^k, d) = \mathbb{Z}_{p^k}[X]/h(X)$, where $\deg h = d$. The existence of asymptotically good RMFE ensures that d/l is bounded by a constant independent of n.

We show that the method from Turbopack applies to honest majority secure multiparty computation over $\mathbb{Z}_{p^k}$ in the preprocessing model, achieving perfect security and the best online communication complexity. In the semi-honest setting, we build a secure multiparty computation protocol over $\mathbb{Z}_{p^k}$ based on the packed Shamir secret-sharing over an extension ring and RMFEs. The online communication requires $12d/l$ elements per multiplication gate. The communication cost per multiplication gate in the preprocessing phase is $\Theta(n \log n)$. This is the first protocol over $\mathbb{Z}_{p^k}$ such that an arithmetic circuit can be evaluated with overall communication complexity $O(1)$ elements per multiplication gate.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

Secure Multiparty Computation, Shamir Secret-Sharing, Galois ring

ACM Reference Format:
Minghua Zhao. 2024. Honest Majority Multiparty Computation over Rings with Constant Online Communication. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3634737.3661136

1 INTRODUCTION

Secure multiparty computation (MPC) enables a set of parties without a trusted third party, each having its own input, to compute a given function on these without leaking anything besides the output. Security requires that, even if an adversary corrupts t out of the n parties, this adversary learns nothing about the inputs of honest parties beyond what is possibly leaked by the output of the computation.

There is a long line of works studying the efficiency of honest majority information-theoretic MPC in the settings of passive security and active security with abort, for an arbitrary number of parties. In 1979, Shamir [26] used polynomials over fields to share secrets. Since then, efforts are paid to apply Shamir's secret-sharing to multiparty computation [6, 12, 15]. The first protocol in achieving linear communication complexity in this setting is DN07 [15]. Dispute control appears as an important technique for robust protocols [4, 5].

In 1992, Franklin and Yung [21] used packed Shamir secret-sharing over fields to reduce the communication complexity. But the secrets within a single packed secret sharing are binded together, i.e. they must be added, multiplied and opened parallelly. This makes the original way of using packed Shamir secret-sharing difficult to be applied in practice. Benhamouda et al. [7] used random masks associated with lines in the circuit so that they could unpack secrets in the calculation and still preserve security. Later, Escudero et al. [18] improved this technique and built a secure multiparty protocol called "Turbopack" in the case where $n = 2t + 1$. The online phase of their protocol requires communicating 12 field elements in total per multiplication gate with circuit-dependent preprocessing. But Turbopack requires the field size to be larger than $n + K$, where K is the parameter of the packed secret sharing scheme. Later, Hu et al. [24] built an MPC protocol for boolean circuits with constant online communication.

Most progress has focused on arithmetic circuits over finite fields. However, modern CPUs are designed over rings like $\mathbb{Z}_{2^{64}}$. It is expected that protocols over rings match the data format more closely and have faster computation than those over fields. Cramer et al. [11] constructed the SPD$\mathbb{Z}_{2^k}$ over $\mathbb{Z}_{2^k}$ for dishonest majority in the preprocessing model that has efficiency comparable to the well-known SPDZ protocol [16]. There are also other works [9, 10, 25] considering MPC over $\mathbb{Z}_{2^k}$. In [20], the authors even constructed MPC protocols based on a generalization of Shamir's secret sharing to non-commutative rings.

In 2003, Cramer et al. [12] have showed monotone span programs over the rings with large enough Lenstra constants. Their construction is based on Shamir secret-sharing. Recently, Abspoel et al. [1] have developed the theory of information-theoretic secure multiparty computation over $\mathbb{Z}_{p^k}$ via Galois rings. Although they pointed out that the theory might be developed for packed Shamir secret-sharing, the construction in [1] did not pack secrets. The extension of rings incurs on a multiplicative overhead of $\log n$ in communication complexity when compared with the Shamir secret-sharing over large fields.

To reduce the overhead caused by the extension of rings, Reverse Multiplication Friendly Embeddings (RMFEs) [8, 13] translates a single multiplication in $GR(2^k, d)$ into a component-wise multiplication in $\mathbb{Z}_{2^k}^l$. Dalskov et al. [14] have showed that Circuit Amortization Friendly Encodings (CAFEs), a generalization of RMFEs, allow to compute certain circuits over $\mathbb{Z}_{p^k}$ at the cost of a single secure multiplication in its extension ring. Abspoel et al. [2] observed that some algebra techniques could be applied to reducing communication complexity of the previous theory. Through these techniques, the magnitude of the communication complexity of multiplication based on Shamir secret-sharing over $\mathbb{Z}_{p^k}$ is the same with that over $\mathbb{F}_{p^k}$.

1.1 Our Contribution

In this work, we use packed Shamir secret-sharing to develop the theory of information-theoretic secure multiparty computation over $\mathbb{Z}_{p^k}$ via Galois rings. We construct the first semi-honest MPC protocol over $\mathbb{Z}_{p^k}$ such that the amortized online communication per multiplication gate is constant.

First, we use Galois rings to obtain large exceptional sets. A Galois ring is of the form $R = \mathbb{Z}_{p^k}[Y]/(h(Y))$, where $h(Y)$ is a non-constant, monic polynomial such that its reduction module p is irreducible in $\mathbb{F}_p[Y]$. If $d = \deg h$ is large enough, then R will contain a large enough exceptional set for packed Shamir secret-sharing. Denote R by $GR(p^k, d)$.

Next, we use the concept of RMFE [8] to embed $(\mathbb{Z}_{p^k})^l$ into R as $\mathbb{Z}_{p^k}$-modules. An RMFE is a pair of $\mathbb{Z}_{p^k}$-module homomorphisms (ϕ, ψ), $\phi : (\mathbb{Z}_{p^k})^l \longrightarrow R$, $\psi : R \longrightarrow (\mathbb{Z}_{p^k})^l$,such that $\mathbf{x} * \mathbf{y} = \psi(\phi(\mathbf{x})\phi(\mathbf{y}))$ for all $\mathbf{x}, \mathbf{y} \in (\mathbb{Z}_{p^k})^l$, where $*$ denotes the coordinate-wise product. The enlarging factor d/l is an important parameter. We show a construction of RMFEs that for $n < 2^{128}$, $d/l \leq 8$. For $n < 2^{15}$, $d/l \leq 4$. Because of the existence of asymptotically-good RMFEs [8, 13], d/l is bounded by a constant.

With RMFEs, a batch of l multiplication gates is computed via one multiplication over R. For every multiplication gate in R, RMFEs require ReEncode [8], causing more communication rounds. We solve this problem by merging ReEncode into the multiplication operation.

Then we build our MPC protocol following Turbopack [18], the state-of-the-art, which is an MPC protocol over fields achieving constant online communication complexity per multiplication gate. The concurrent work [24] leverages RMFEs and the pattern of Turbopack to build an MPC protocol for boolean circuits with constant online communication. Our protocol over $\mathbb{Z}_{p^k}$ consists of three phases:

(1) A circuit and input-independent phase, requiring communication of $7nd/l$ elements in $\mathbb{Z}_{p^k}$ per multiplication gate.
(2) A circuit-dependent but input-independent phase, requiring communication of $\frac{8}{3}nd^2/l + \frac{13}{3}nd/l$ elements in $\mathbb{Z}_{p^k}$ per multiplication gate.
(3) A circuit and input-dependent phase, requiring communication of $12d/l$ elements in $\mathbb{Z}_{p^k}$ per multiplication gate.

We will explain later that d can be chosen to be $\Theta(\log n)$. So the communication complexity of phase (2) per multiplication gate is $\Theta(n \log n)$. The communication complexity of phase (3) per gate is actually $O(1)$.

Our MPC protocol over $\mathbb{Z}_{p^k}$ is based on Shamir's sharing scheme over R. One of our contributions is showing how to deal with the random elements in $im\,\phi$ and R together. More concretely, a random value $\lambda_\alpha \in \mathbb{Z}_{p^k}$ is assigned to each wire α in the circuit. For each addition gate with input wires α, β and output wire γ, $\lambda_\gamma = \lambda_\alpha + \lambda_\beta$. Assume that if the circuit is computed in plaintext, then each wire α is assigned with v_α. A designated party P_1 always learns $\mu_\alpha = v_\alpha - \lambda_\alpha$ for each wire α in the online phase.

For each addition gate with input wires α, β and output wire γ, P_1 locally compute $\mu_\gamma = \mu_\alpha + \mu_\beta$. Multiplication gates in each layer of the circuit are divided into small groups and big groups. For each big group of Kl multiplication gates, let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{Kl})$ denote the batch of the first input wires, $\boldsymbol{\beta}$ denote the batch of the second input wires and $\boldsymbol{\gamma}$ denote the batch of the output wires. We use Beaver triples [3] to compute the multiplication based on Shamir sharings. A mask like $(\phi(\lambda_{\gamma_1}, \ldots, \lambda_{\gamma_l}), \ldots, \phi(\lambda_{\gamma_{Kl-l+1}}, \ldots, \lambda_{\gamma_{Kl}})) \in R^K$ can not hide $(\phi(v_{\alpha_1}, \ldots, v_{\alpha_l})\phi(v_{\beta_1}, \ldots, v_{\beta_l}), \ldots)$ well, because the product of two elements in $im\,\phi$ does not necessarily lie in $im\,\phi$. We use a random mask $\boldsymbol{\lambda_\gamma} \in R^K$ such that applying ψ to all entries gives $(\lambda_{\gamma_1}, \ldots, \lambda_{\gamma_{Kl}})$.

We observe that $R = im\,\phi \oplus \ker\psi$ as $\mathbb{Z}_{p^k}$-modules. The random masks built from masks in $\mathbb{Z}_{p^k}$ lie in $(im\,\phi)^K$. Then we can add random elements in $(\ker\psi)^K$ to make the masks random in R^K and the result of applying ψ remains unchanged.

Based on these observations, we build a semi-honest MPC protocol over $\mathbb{Z}_{p^k}$ with constant online communication. Finally, we remark that our results extend to MPC over $\mathbb{Z}_N$ for any integer N via the Chinese Remainder Theorem.

1.2 Outline of this Article

The first section gives an introduction of multiparty computation and our contribution. In Section 2 we introduce the preliminaries for the rest of the work. This includes basic notations, Galois rings, RMFEs and Shamir secret-sharing over commutative rings. In Section 3, we present some building blocks for our main protocol. In Section 4, we present our semi-honest MPC protocol over $\mathbb{Z}_{p^k}$ with honest majority. In Section 5, we conclude this work.

2 PRELIMINARIES

2.1 Notation

In this paper, rings mean commutative rings with identity. Let $\mathbb{Z}$ denote the ring of integers. For an integer m, $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$. For a ring R, let $R[X]$ denote the ring of polynomials in the indeterminate X with coefficients in R. Let $M_m(R)$ denote the ring of $m \times m$ matrices over R with the identity I_m. For an integer $m \geq 0$, let $R[X]_{\leq m} \subset R[X]$ denote the set of polynomials in $R[X]$ of degree at most m; it is an R-module. We denote by R^* the multiplicative subgroup of invertible elements in R.

We use bold characters for vectors and $*$ for componentwise product. Denote the vector whose entries are all 1 by $\mathbf{1}$. Let $V_{n,d}(\alpha_1,\ldots,\alpha_n)$ be an $n \times d$ Vandermonde matrix with the entry being α_i^{j-1} in the i-th row and the j-th column, $1 \le i \le n$, $1 \le j \le d$. For a set S, $x \leftarrow S$ means that x is sample randomly from S. The subscript of $\log_2$ is often omitted. We use ker to denote the kernel and *im* to denote the image of a function.

The arithmetic circuits in this paper are over $\mathbb{Z}_{p^k}$ and there are four types of gates: input gates, addition gates, multiplication gates and output gates. Input gates have no input wire and one output wire. Addition gates and multiplication gates have two input wires and one output wire. Output gates have one input wire and no output wire.

2.2 Model

In this work, we focus on honest majority computation over $\mathbb{Z}_{p^k}$, where p is a prime and $k \ge 1$. We assume secure channels between any two nodes in the network. Furthermore, we assume the availability of broadcast channels.

We use the client-server model for the secure multi-party computation. In the client-server model, clients provide inputs to the functionality and receive outputs, and servers can participate in the computation but do not have inputs nor get outputs. Each party may have different roles in the computation. Note that, if every party plays a single client and a single server, this corresponds to a protocol in the standard MPC model. Let c denote the number of clients and n denote the number of servers. One benefit of the client-server model is that it is sufficient to only consider maximum adversaries [22]. We use $\{P_1,\ldots,P_n\}$ to denote the n servers, and refer to the servers as parties. In this paper, we assume $n \ge 3, t = \lfloor \frac{n-1}{2} \rfloor$ and the adversary corrupts exactly t servers.

Clients and servers are allowed to abort at any time. If one aborts, then all abort and output abort.

The output of the real-world execution consists of the inputs and outputs of honest clients and servers and the view of the adversary. The output of the ideal-world execution consists of the inputs and outputs of honest clients and servers and the view of the adversary. We say that a protocol Π computing a function F is perfectly (statistically) secure if, for all adversary A, there exists a simulator, such that the distribution of the output of the real-world execution is the same as (statistically indistinguishable from) that of the ideal-world execution.

2.3 Galois Rings

Recall that Shamir's secret-sharing can be interpreted by Lagrange interpolating polynomials or Vandermonde matrices. The security of Shamir's secret-sharing actually relies on super-invertible matrices, which are easy to construct over fields.

Definition 2.1. Assume $r \ge c$. A matrix $M \in R^{r\times c}$ is super-invertible if for any row index set $W \subset \{1,\ldots,r\}$ with $|W| = c > 0$, the submatrix M_W is invertible, where M_W denotes the submatrix of M with rows indexed by W.

The definition for the case $r < c$ is similar.

To construct a super-invertible Vandermonde matrix over rings, the following definition plays an important role.

Definition 2.2. Let $\alpha_1,\ldots,\alpha_n \in R$. These points form an exceptional set if $\forall i \ne j$, $\alpha_i - \alpha_j \in R^*$. Define the Lenstra constant of R to be the maximal size of an exceptional set in R.

Even for large k, the Lenstra constant of $\mathbb{Z}_{p^k}$ is p. In particular, the Lenstra constant of $\mathbb{Z}_{2^{64}}$ is 2, which makes it difficult to construct secure Shamir secret-sharing over this ring. To obtain larger Lenstra constants, we use Galois rings as in [1].

Definition 2.3. A Galois ring is a ring of the form

$$R := \mathbb{Z}_{p^k}[Y]/(h(Y)),$$

where $h(Y) \in \mathbb{Z}_{p^k}[Y]$ is a non-constant, monic polynomial such that its reduction module p is irreducible in $\mathbb{F}_p[Y]$.

Proposition 2.4 ([27]). *Let R be as in the above definition. It has the following properties:*

(1) R is a local ring, i.e. it has a unique maximal ideal (p). R^* consists of the elements not in (p). We have $R/(p) \cong \mathbb{F}_{p^d}$, where $d = \deg h$. The "reduction module p" is a ring homomorphism $\Pi : R \longrightarrow \mathbb{F}_{p^d}$.

(2) The Lenstra constant of R is p^d.

(3) For any prime p, positive integer k, and positive integer d, there exists a Galois ring as defined above, and it is unique up to isomorphism. Write $R = GR(p^k, d)$.

(4) If e is any positive integer, then R is a subring of $\hat{R} = GR(p^k, d\cdot e)$. There is a non-constant, monic polynomial $\hat{h} \in R[X]$ that is irreducible module p, such that $\hat{R} = R[X]/(\hat{h}(X))$.

For more properties and proofs of the assertions about Galois rings, refer to [27].

Note that R is a free $\mathbb{Z}_{p^k}$-module of rank $d = \deg h$. One basis is $1, Y, \ldots, Y^{d-1}$ module $(h(Y))$, also denoted by $1, \overline{Y}, \ldots, \overline{Y}^{d-1}$. We write $a \in R$ as $a_0 + a_1\overline{Y} + \cdots + a_{d-1}\overline{Y}^{d-1}$. Assume $h(Y) = h_0 + h_1 Y + \cdots + h_{d-1}Y^{d-1} + Y^d$, then its formal derivative is $h'(Y) = h_1 + \cdots + (d-1)h_{d-1}Y^{d-2} + dY^{d-1}$.

2.4 Reverse Multiplication Friendly Embedding

We want to pack $K = \lfloor \frac{n-t+1}{2} \rfloor$ secrets into one polynomial. Note that $K \approx n/4$. Choose the integer d such that $p^d > n + K$. So $d = O(\log n)$.

So embedding $\mathbb{Z}_{p^k}$ into $GR(p^k, d)$ via only the constant term will increase the communication cost by a $\log n$ factor. So we need the RMFE technique to reduce the communication cost as in [8].

Definition 2.5. Let j be a positive integer. A pair (ϕ, ψ) is called an (l, m)-Reverse Multiplication Friendly Embedding (RMFE) if $\phi : GR(p^k, j)^l \longrightarrow GR(p^k, jm)$ and $\psi : GR(p^k, jm) \longrightarrow GR(p^k, j)^l$ are two $GR(p^k, j)$-linear maps satisfying

$$\mathbf{x} * \mathbf{y} = \psi(\phi(\mathbf{x}) \cdot \phi(\mathbf{y})), for\ all\ \mathbf{x}, \mathbf{y} \in GR(p^k, j)^l.$$

From the definition, $\mathbf{x} = \psi(\phi(\mathbf{x})\cdot\phi(1,\ldots,1))$ for all $x \in GR(p^k, j)^l$. So we can recover $\mathbf{x}$ from $\phi(\mathbf{x})$. Therefore, ϕ is injective and ψ is surjective. Replacing ψ by $\psi_2 : GR(p^k, jm) \longrightarrow GR(p^k, j)^l, x \mapsto \psi(x \cdot \phi(1,\ldots,1))$, we assume $\psi \circ \phi(\mathbf{x}) = \mathbf{x}$ for all $\mathbf{x} \in GR(p^k, j)^l$

in the rest of the article. Then an RMFE gives a $GR(p^k, j)$-module embedding

$$emb : GR(p^k, j) \longrightarrow GR(p^k, jm), x \mapsto \phi(x, x, \dots, x).$$

One useful property is that $\phi \circ \psi$ is a projection onto $im\,\phi$, i.e. $\phi \circ \psi \circ \phi \circ \psi = \phi \circ \psi$. Therefore, $\forall x \in GR(p^k, jm)$, $x = \phi \circ \psi(x) + (x - \phi \circ \psi(x))$ is a unique representation as the sum of elements from $im\,\phi$ and $\ker\psi$, that is, $GR(p^k, jm) \cong im\,\phi \oplus \ker\psi$.

For $j = 1$, $GR(p^k, 1) = \mathbb{Z}_{p^k}$. We construct an RMFE as follows.

For a positive integer δ, we show an RMFE $(\phi_\delta, \psi_\delta)$ for $GR(p^k, \delta) \subset GR(p^k, 2p^\delta\delta)$. By Proposition 2.4, there exists $\hat{h}(X) \in GR(p^k, \delta)[X]$ such that $\deg \hat{h}(X) = 2p^\delta$, $GR(p^k, 2p^\delta\delta) = GR(p^k, \delta)[X]/(\hat{h}(X))$ and $GR(p^k, \delta)$ contains an exceptional set $\{q_1, \dots, q_{p^\delta}\}$. Define

$$\phi_\delta : GR(p^k, \delta)^{p^\delta} \longrightarrow GR(p^k, 2p^\delta\delta);\ \mathbf{x} \mapsto f,$$

where $f = F[X]$ module $\hat{h}(X)$ such that $\deg F < p^\delta$ and $F(q_i) = x_i, 1 \le i \le p^\delta$. Define

$$\psi_\delta : GR(p^k, 2p^\delta\delta) \longrightarrow GR(p^k, \delta)^{p^\delta};\ f \mapsto \mathbf{x},$$

where $f = F[X]$ module $\hat{h}(X)$ and $\mathbf{x} = (F(q_1), \dots, F(q_{p^\delta}))^T$. Since the degree of product of two low degree polynomials does not exceed $\deg \hat{h}$, $(\phi_\delta, \psi_\delta)$ is a $(p^\delta, 2p^\delta)$-RMFE. Specifically, $\phi_\delta(\mathbf{x})$, $\phi_\delta(\mathbf{y})$ represent polynomials $F[X], G[X] \in GR(p^k, \delta)[X]$ with degrees less than p^δ. Then the product $\phi_\delta(\mathbf{x}) \cdot \phi_\delta(\mathbf{y}) = F[X]G[X] \mod \hat{h}(X)$. Since $\deg F[X]G[X] < \deg \hat{h}(X)$,

$$\psi_\delta(\phi_\delta(\mathbf{x}) \cdot \phi_\delta(\mathbf{y})) = (F(q_1)G(q_1), \dots, F(q_{p^\delta})G(q_{p^\delta})) = \mathbf{x} * \mathbf{y}$$

This construction has appeared in [19].

One sequence of the ring extensions is

$$\mathbb{Z}_{p^k} \subset GR(p^k, 2p) \subset GR(p^k, 4p^{2p+1}) \subset \dots$$

This is a fast extension: for $p = 2$, three times of extension yields $\delta > 2^{128}$. When p^δ approaches $n + K$, we can change the extension to be $GR(p^k, \delta) \subset GR(p^k, 2m\delta)$ for some small integer m.

We continue the above construction until we reach a ring $GR(p^k, d)$ where $p^d > n + K$. In practice, $n < 2^{128}$, then the number of extension times ≤ 3. For practical application, we can control the extension rate so that p^d is not too large. We assume $d = \Theta(\log n)$.

To concatenate RMFEs, we need Lemma 5 from [8] or the composition lemma from [19]. The concatenation gives $\phi : \mathbb{Z}^l_{p^k} \longrightarrow GR(p^k, d)$ and $\psi : GR(p^k, d) \longrightarrow \mathbb{Z}^l_{p^k}$. For example, ϕ is defined as

$$\mathbf{x} \mapsto (\phi_1(x_1, \dots, x_p), \dots, \phi_1(\dots, x_l)) \mapsto$$
$$(\phi_{2p}(\phi_1(x_1, \dots, x_p), \dots), \dots, \phi_{2p}(\dots, \phi_1(\dots, x_l)))) \mapsto \dots$$

Now we know (ϕ, ψ) is an (l, d)-RMFE, where $d/l = 2^T$ and T is the number of extension times. From the fast extension, T is a small function of n, asymptotically smaller than any finite composite of logarithm functions. For $n < 2^{128}$, (ϕ, ψ) is an (l, d)-RMFE where $d/l \le 8$. For $n < 2^{15}$, (ϕ, ψ) is an (l, d)-RMFE where $d/l \le 4$.

For asymptotically-good RMFEs, see [8, 13]. Asymptotically-good RMFEs satisfy that $\lim_{n\to\infty} d/l$ is a positive constant. For $p = 2$, there exists a family of (l, d)-RMFEs such that $\lim_{l\to\infty} d/l \approx 4.92$ [19]. Because of the existence of asymptotically-good RMFEs, we can assume d/l is bounded by a constant. Although the concrete construction above is not asymptotically-good, it works well for $n < 2^{128}$.

The embedding

$$emb : \mathbb{Z}_{p^k} \longrightarrow GR(p^k, d), x \mapsto \phi(x, \dots, x)$$

is a $\mathbb{Z}_{p^k}$-module injection. Note that $GR(p^k, d) \cong \mathbb{Z}^d_{p^k}$ and $emb^{-1} : im\,emb \longrightarrow \mathbb{Z}_{p^k}$ is a $\mathbb{Z}_{p^k}$-module isomorphism. Therefore, the $\mathbb{Z}_{p^k}$-module homomorphisms ϕ,ψ,emb and emb^{-1} can be computed efficiently.

Hereafter, we only need to focus on Shamir secret-sharing over the ring $R = GR(p^k, d)$. Choose an exceptional set $\{p_1, \dots, p_K, \alpha_1, \dots, \alpha_n\} \subset R$. The secrets are stored at positions $p_1, \dots, p_K$ and the shares are stored at positions $\alpha_1, \dots, \alpha_n$.

2.5 Shamir secret-sharing

In Figure 1, the protocol Π_{Share} allows a party to share K secrets $x_1, \dots, x_K \in R$ via one polynomial with degree $\le v$. For concrete computation, we state the protocol using Vandermonde matrices instead of polynomials.

$\Pi_{Share}(P_i, v, x_1, \dots, x_K)$

(1) P_i samples $s_1, \dots s_{v+1-K} \leftarrow R$ and computes

$$(f_0, \dots, f_v)^T = (V_{v+1,v+1}(p_1, \dots, p_K, \alpha_1, \dots, \alpha_{v+1-K}))^{-1} \cdot (x_1, \dots, x_K, s_1, \dots s_{v+1-K})^T;$$

(2) P_i computes

$$(s_{v+2-K}, \dots, s_n)^T = (V_{n-v-1+K,v+1}(\alpha_1, \dots, \alpha_n)) \cdot (f_0, \dots, f_v)^T$$

and sends s_j to P_j for $j \ne i$.

Figure 1: Packed Shamir Secret-Sharing Protocol

Note that the inverse of the Vandermonde matrices can be computed in the pre-processing phase. To share Kl secrets $x_1, \dots, x_{Kl} \in \mathbb{Z}_{p^k}$, P_i uses ϕ from section 2.4 to encode them into R before invoking Π_{Share}. Since ϕ is not surjective, parties need to check if some secrets are in $im\phi$ in the malicious model.

We use $[\mathbf{x}]_v$ to denote $(s_1, \dots, s_n)$ generated in Π_{Share} for $\mathbf{x} \in R^K$. We call $[\mathbf{x}]_v$ a degree-v packed Shamir sharing. If the subscript can be omitted, then we write $[\mathbf{x}]$. Since $K + t \le n - K + 1$, the K secrets of a degree-$(n-K)$ packed Shamir sharing are independently random for an adversary holding t shares. So a degree-$(n - K)$ packed Shamir sharing is private against t corrupted parties. If $x \in R$ is stored at p_i and is shared, then we denote this by $[x|_i]$.

The addition of the sharings can be computed locally: $[\mathbf{x}+\mathbf{y}]_v = [\mathbf{x}]_v + [\mathbf{y}]_v$. For $d_1+d_2 < n$, local multiplication increases the degree: $[\mathbf{x} * \mathbf{y}]_{d_1+d_2} = [\mathbf{x}]_{d_1}[\mathbf{y}]_{d_2}$. One multiplication in R translates into l multiplication in $\mathbb{Z}_{p^k}$ via (ϕ, ψ).

For public vector $\mathbf{c} \in R^K$, $\mathbf{c}$ decides a unique polynomial with degree $\le K-1$. So parties can compute $[\mathbf{c} * \mathbf{x}]_{v+K-1} = [\mathbf{c}]_{K-1}[\mathbf{x}]_v$. For a constant $c \in R$, $c \cdot \mathbf{1} \in R^K$ and $[c \cdot \mathbf{1}]_0$ is a degree-0 packed Shamir sharing.

If P_i wants to reconstruct K secrets $x_1, \dots, x_K \in R$ from $[\mathbf{x}]_v$, then P_i collects all shares and uses the Vandermonde matrix to recover a polynomial $f(X) \in R[X]$ such that $f(\alpha_j)$ is the share from P_j for all $1 \le j \le n$. If such a polynomial does not exist, then P_i aborts. The secrets are $x_j = f(p_j)$ for $1 \le j \le K$.

For a fixed v and $K \le v < n$, the set $C = \{[\mathbf{x}]_v | \mathbf{x} \in (im\phi)^K\}$ is a free $\mathbb{Z}_{p^k}$-module of rank $Kl + (v+1-K)d$. This is because $im\phi$ is a free $\mathbb{Z}_{p^k}$-module of rank l and R is a free $\mathbb{Z}_{p^k}$-module of rank d.

There is an important lemma for the security of Shamir's secret-sharing.

Lemma 2.6. *Let G be a finite group and X, Y are independent random variables with sample space G. If X is uniform, then $X \cdot Y$ is a uniform random variable.*

Lemma 2.6 is often used implicitly to explain uniform randomness of random variables generated collectively by parties. Since modules are Abelian groups, this lemma applies to modules.

3 BUILDING BLOCKS

In this section, we prepare some building blocks for our semi-honest MPC. We choose P_1 as a special party. P_1 learns some invariants during the MPC protocol and plays an important role in the computation.

To provide random sharings, we follow the technique in [15] as described in the protocol Π_{Randsh} in Figure 2. The input of Π_{Randsh} is a sharing scheme or a set of some sharings, denoted by Σ.

$\Pi_{Randsh}(\Sigma)$

(1) Parties agree on an $(n-t) \times n$ super-invertible matrix M over R;
(2) Each party P_i samples a random Σ-sharing $S^{(i)}$ and distributes the shares to other parties;
(3) All parties locally compute and output

$$(R^{(1)}, \dots, R^{(n-t)})^T = M(S^{(1)}, \dots, S^{(n)})^T.$$

Figure 2: Random Sharing Protocol

If Σ is the sharing scheme corresponding to $[r|_i]_{n-K}$, $r \leftarrow R$, then the results are shares of uniform random variables due to the invertibility of any $(n-t)\times(n-t)$ submatrix of M. If Σ is the sharing scheme corresponding to $([r|_i]_t, [r|_i]_{2t})$, $r \leftarrow R$, then $\Pi_{Randsh}(\Sigma)$ generates random double sharings. The amortized communication of one random share generated by Π_{Randsh} is $\frac{n(n-1)}{n-K} \approx \frac{4}{3}n$ elements in R, which are $\frac{4}{3}nd$ elements in $\mathbb{Z}_{p^k}$.

We prepare the protocol $\Pi_{SingleMult}$ in Figure 3 for the single multiplication of secrets, where one sharing hides one secret.

In the semi-honest model, the correctness of $\Pi_{SingleMult}$ is obvious. For the security, we refer to [15, 18]. In the malicious model, $\Pi_{SingleMult}$ is known to be attacked easily. The special party P_1 can add any value to the result but the adversary does not know the result. The communication of $\Pi_{SingleMult}$ is $\frac{14}{3}n - 2$ elements in R.

Since $emb : \mathbb{Z}_{p^k} \longrightarrow R, x \mapsto \phi(x, x, \dots, x)$ is an embeding of $\mathbb{Z}_{p^k}$-modules, we have a $\mathbb{Z}_{p^k}$-morphism $\Phi : (im\, emb)^l \longrightarrow R$ induced by ϕ. Then for $\mathbf{x} \in \mathbb{Z}_{p^k}^l$,

$$\psi \circ \Phi(emb(x_1), \dots, emb(x_l)) = (x_1, \dots, x_l)$$

$\Pi_{SingleMult}(i, [x|_i]_t, [y|_i]_t)$

(1) Use Π_{Randsh} to generate random double sharings. Prepare one unused random double sharing $([r|_i]_t, [r|_i]_{2t})$.
(2) All parties locally compute $[e|_i]_{2t} = [x|_i]_t[y|_i]_t + [r|_i]_{2t}$.
(3) P_1 collects shares and reconstructs e. Then P_1 randomly generates a Shamir sharing $[e|_i]_t$ and distributes the shares to other parties.
(4) All parties locally compute $[z|_i]_t = [e|_i]_t - [r|_i]_t$.

Figure 3: Single Multiplication Protocol

The natural isomorphism of modules $\mathbb{Z}_{p^k}^d \cong R, (a_0, \dots, a_{d-1}) \mapsto a_0 + \dots + a_{d-1}\overline{Y}^{d-1}$ induces an isomorphism $iso : (im\, emb)^d \longrightarrow R$. Then $iso^{-1} \circ \Phi$ is essentially a $\mathbb{Z}_{p^k}$-homomorphism from $\mathbb{Z}_{p^k}^l$ to $\mathbb{Z}_{p^k}^d$. The key point is that for $\mathbf{x} \in (im\, emb)^l$, every entry of $iso^{-1} \circ \Phi(\mathbf{x})$ is a $\mathbb{Z}_{p^k}$-linear combination of $x_1, \dots, x_l$. So all parties can locally compute

$$([y_1 \cdot \mathbf{1}]_{n-K}, \dots, [y_d \cdot \mathbf{1}]_{n-K}) = iso^{-1} \circ \Phi([x_1 \cdot \mathbf{1}]_{n-K}, \dots, [x_l \cdot \mathbf{1}]_{n-K}).$$

For simplicity, we may omit emb in our protocol and think $\mathbb{Z}_{p^k}$ as a submodule of R via emb. The protocol $\Pi_{ConvertToRing}$ for computing Φ is presented in Figure 4.

$\Pi_{ConvertToRing}([x_1 \cdot \mathbf{1}]_{n-K}, \dots, [x_l \cdot \mathbf{1}]_{n-K})$

input : $[x_1 \cdot \mathbf{1}]_{n-K}, \dots, [x_l \cdot \mathbf{1}]_{n-K}$, where $x_1, \dots, x_l \in im\, emb$
output: $[y \cdot \mathbf{1}]_{n-K}$, where $y = \Phi(x_1, \dots, x_l)$

(1) Let Σ be the sharing scheme corresponding to $([r_1 \cdot \mathbf{1}]_{n-K}, \dots, [r_d \cdot \mathbf{1}]_{n-K}, [r \cdot \mathbf{1}]_{n-K})$, $r \leftarrow R$, $(r_1, \dots, r_d) = iso^{-1}(r)$. Use $\Pi_{Randsh}(\Sigma)$ to generate enough correlated random tuples.
(2) Parties locally compute $([y_1 \cdot \mathbf{1}]_{n-K}, \dots, [y_d \cdot \mathbf{1}]_{n-K}) = iso^{-1} \circ \Phi([x_1 \cdot \mathbf{1}]_{n-K}, \dots, [x_l \cdot \mathbf{1}]_{n-K})$.
(3) Use an unused tuple $([r_1 \cdot \mathbf{1}]_{n-K}, \dots, [r_d \cdot \mathbf{1}]_{n-K}, [r \cdot \mathbf{1}]_{n-K})$, $r \leftarrow R$, $(r_1, \dots, r_d) = iso^{-1}(r)$. P_1 collects shares of $[(y_1 - r_1) \cdot \mathbf{1}]_{n-K}, \dots, [(y_d - r_d) \cdot \mathbf{1}]_{n-K}$ and reconstructs them.
(4) P_1 computes $u = iso(y_1 - r_1, \dots, y_d - r_d)$ and generates a random Shamir sharing $[u \cdot \mathbf{1}]_{n-K}$ and distributes the shares to other parties. Then all parties locally compute and output $[y \cdot \mathbf{1}]_{n-K} = [u \cdot \mathbf{1}]_{n-K} + [r \cdot \mathbf{1}]_{n-K}$.

Figure 4: Vector Convert To Ring Element Protocol

For the correctness of $\Pi_{ConvertToRing}$, note that $(y_1, \dots, y_d) = iso^{-1} \circ \Phi(x_1, \dots, x_l)$. Then

$$u = iso(y_1 - r_1, \dots, y_d - r_d) = \Phi(x_1, \dots, x_l) - r.$$

So $y = u + r = \Phi(x_1, \dots, x_l)$ as required.

The communication of $\Pi_{ConvertToRing}$ is approximately $\frac{8}{3}nd^2 + \frac{7}{3}nd$ elements in $\mathbb{Z}_{p^k}$.

Our construction of RMFEs actually satisfies that $emb(x)$ is just the constant term in R. This can lead to the availability of local computation of Φ. But we ignore this property and keep the protocol general.

4 SEMI-HONEST MPC VIA PACKED SHAMIR SECRET-SHARING

4.1 Overview

In this section, we present our semi-honest MPC protocols over $\mathbb{Z}_{p^k}$ via packed Shamir secret-sharing over R. We only consider the circuit-dependent preprocessing model for simplicity. The protocol is similar with that over fields from [18].

Recall that there are n parties, $t = \lfloor \frac{n-1}{2} \rfloor$ and $K = \lfloor \frac{n-t+1}{2} \rfloor$. Approximately, $K \approx n/4$. The arithmetic circuit to compute is over $\mathbb{Z}_{p^k}$ and is layered. The topology of circuits is not the subject of this paper. We assume the input gates are in the first layer and the output gates are in the last layer. Recall that $\phi : \mathbb{Z}_{p^k}^l \longrightarrow R$ and $\psi \circ \phi = id_{\mathbb{Z}_{p^k}^l}$. An RMFE gives a $\mathbb{Z}_{p^k}$-module embedding: $emb : \mathbb{Z}_{p^k} \longrightarrow R$, $x \mapsto \phi(x, x, \ldots, x)$. $\Phi : (im\, emb)^l \longrightarrow R$ is induced by ϕ.

We assume that l gates of the same type in the same layer form a small group. Furthermore, K small groups of the same type in the same layer form a big group. Every group of input gates or output gates belong to one client. We may add dummy gates to ensure that gates can be divided into groups as above. The wires of small groups are indexed with Greek letters.

In general, we provide random values in R. For $\mathbf{x} \in \mathbb{Z}_{p^k}^l$ and $r \in R$, we view the masked value $\phi(\mathbf{x}) - r$ in R as $\mathbf{x} - \psi(r)$ in $\mathbb{Z}_{p^k}^l$. Then there are different values $r_1, r_2 \in R$ such that $\psi(r_1) = \psi(r_2)$ and they provide the same mask in $\mathbb{Z}_{p^k}^l$.

Assume that if the circuit is executed in plaintext, then a value $v_\alpha \in \mathbb{Z}_{p^k}$, called a true value, is assigned to each wire α. For multi-party computation, each wire α is assigned with a random masked $\lambda_\alpha \in \mathbb{Z}_{p^k}$. P_1 always maintains $\mu_\alpha = v_\alpha - \lambda_\alpha$ for each wire α. Note that learning $emb(\mu_\alpha)$ is equivalent to learning μ_α. For output wires $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{Kl})$ of a big group, $\boldsymbol{\lambda_\alpha} \leftarrow R^K$ is a random mask. We need to ensure that $(\lambda_{\alpha_1}, \ldots, \lambda_{\alpha_{Kl}})$ is the result of applying ψ to all entries of $\boldsymbol{\lambda_\alpha}$. Let

$$\begin{aligned}\mathbf{v}_{\boldsymbol{\alpha}} =&(\Phi(emb(v_{\alpha_1}), \ldots, emb(v_{\alpha_l})), \ldots,\\ &\Phi(emb(v_{\alpha_{Kl-l+1}}), \ldots, emb(v_{\alpha_{Kl}})))\\ =&(\phi(v_{\alpha_1}, \ldots, v_{\alpha_l}), \ldots, \phi(v_{\alpha_{Kl-l+1}}, \ldots, v_{\alpha_K})) \in R^K.\end{aligned}$$

Learning $\boldsymbol{v_\alpha}$ is the same as learning $v_{\alpha_1}, \ldots, v_{\alpha_{Kl}}$. Define $\boldsymbol{\mu_\alpha}$ to be the result of applying $\phi \circ \psi$ to all entries of $\boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$. Then applying ψ to all entries of $\boldsymbol{\mu_\alpha}$ gives $\mu_{\alpha_1}, \ldots, \mu_{\alpha_{Kl}}$.

In our protocol, $\lambda_{\alpha_1}, \ldots, \lambda_{\alpha_{Kl}}$ are generated randomly and $\boldsymbol{\lambda_\alpha}$ is computed from them. In some cases, $\boldsymbol{\lambda_\alpha} \in (im\,\phi)^K$ works well. But the product of elements in $im\,\phi$ does not lie in $im\,\phi$. To mask the product, we need to add to $\boldsymbol{\lambda_\alpha}$ elements in $(\ker\psi)^K$ to get a random value in R^K.

4.2 Circuit-Independent Preprocessing

In the circuit-independent preprocessing phase, the numbers of gates of different types are known but the topology remains unknown. We present the ideal functionality $\mathcal{F}_{PrepInd}$ for circuit-independent preprocessing in Figure 5.

$\mathcal{F}_{PrepInd}$

(1) **Preparing Random Packed Sharings**: $\mathcal{F}_{PrepInd}$ receives the set of corrupted parties, denoted by $Corr$. For each output wire α of input gates and multiplication gates:
  (a) $\mathcal{F}_{PrepInd}$ receives from the adversary a set of shares $\{u_j\}_{j \in Corr}$. $\mathcal{F}_{PrepInd}$ samples a random value $\lambda_\alpha \leftarrow \mathbb{Z}_{p^k}$ and computes a degree-$(n-k)$ packed Shamir sharing $[emb(\lambda_\alpha) \cdot \mathbf{1}]_{n-K}$ such that for all $j \in Corr$, P_j's share is u_j.
  (b) $\mathcal{F}_{PrepInd}$ distributes the shares of $[emb(\lambda_\alpha) \cdot \mathbf{1}]_{n-K}$ to honest parties.

(2) **Preparing Packed Beaver Triples**: For each big group of Kl multiplication gates:
  (a) $\mathcal{F}_{PrepInd}$ receives from the adversary a set of shares $\{(u_j^{(1)}, u_j^{(2)}, u_j^{(3)})\}_{j \in Corr}$. $\mathcal{F}_{PrepInd}$ samples two random vectors $\mathbf{a}, \mathbf{b} \in R^K$ and computes $\mathbf{c} = \mathbf{a} * \mathbf{b}$. Then $\mathcal{F}_{PrepInd}$ computes three degree-$(n-K)$ packed Shamir sharings $[\mathbf{a}]_{n-K}, [\mathbf{b}]_{n-K}, [\mathbf{c}]_{n-K}$ such that for all $j \in Corr$, P_j's share is $(u_j^{(1)}, u_j^{(2)}, u_j^{(3)})$.
  (b) $\mathcal{F}_{PrepInd}$ distributes the shares of $[\mathbf{a}]_{n-K}, [\mathbf{b}]_{n-K}, [\mathbf{c}]_{n-K}$ to honest parties.

(3) **Preparing Random Masked Sharings for Multiplication Gates**: For each big group of Kl multiplication gates:
  (a) $\mathcal{F}_{PrepInd}$ receives from the adversary a set of shares $\{(u_j^{(1)}, u_j^{(2)}, u_j^{(3)})\}_{j \in Corr}$. $\mathcal{F}_{PrepInd}$ sets $\mathbf{o}^{(1)} = \mathbf{o}^{(2)} = \mathbf{0} \in R^K$, and samples $\mathbf{o}^{(3)} \in (\ker\psi)^K$. Then $\mathcal{F}_{PrepInd}$ samples three random degree-$(n-1)$ packed Shamir sharings
  $$[\mathbf{o}^{(1)}]_{n-1}, [\mathbf{o}^{(2)}]_{n-1}, [\mathbf{o}^{(3)}]_{n-1}$$
  such that for all $j \in Corr$, P_j's share is $(u_j^{(1)}, u_j^{(2)}, u_j^{(3)})$;
  (b) $\mathcal{F}_{PrepInd}$ distributes the shares of $[\mathbf{o}^{(1)}]_{n-1}$, $[\mathbf{o}^{(2)}]_{n-1}$, and $[\mathbf{o}^{(3)}]_{n-1}$ to honest parties.

(4) **Preparing Random Masked Sharings for Input and Output Gates**: For each big group of Kl input gates or output gates, $\mathcal{F}_{PrepInd}$ samples $\mathbf{o} \in (\ker\psi)^K$. $\mathcal{F}_{PrepInd}$ prepares a random degree-$(n-1)$ packed Shamir sharing $[\mathbf{o}]_{n-1}$, in the same way as above.

Figure 5: Preprocessing Circuit-Independent Functionality

The correalted randomness can be generated by Π_{Randsh}. Then we need $\Pi_{SingleMult}$ to prepare packed Beaver triples. The protocol $\Pi_{PrepInd}$ implements $\mathcal{F}_{PrepInd}$ and is presented in Figure 6.

In Step (2).(c), $[\mathbf{e}_i]_{K-1} * [a_i|_i]_t$ is a degree-$(K-1+t)$ Shamir sharing, it is also a degree-$(n-K)$ Shamir sharing because $n - K \geq$

$K-1+t$. If all parties follow the protocol, then $\mathbf{a}=(a_1,\dots,a_K)$, and $\mathbf{b},\mathbf{c}$ have similar expressions.

$\Pi_{PrepInd}$

(1) **Preparing Random Packed Sharings**: Let N_1 be the number of input gates and multiplication gates. Let Σ_1 be the secret sharing scheme corresponding to $[emb(x)\cdot\mathbf{1}]_{n-K}, x\leftarrow\mathbb{Z}_{p^k}$. All parties invoke $N_1/(n-t)$ times of $\Pi_{Randsh}(\Sigma_1)$ to prepare N_1 random sharings in the form of $[emb(x)\cdot\mathbf{1}]_{n-K}$. Each output wire α of input gates and multiplication gates is assigned with a mask $\lambda_\alpha=x$ in such a way that parties hold $[emb(x)\cdot\mathbf{1}]_{n-K}$.

(2) **Preparing Packed Beaver Triples**: Let N_2 denote the number of big groups of multiplication gates. For $i\in\{1,\dots,K\}$, let $\Sigma_{2,i}$ be the secret sharing scheme corresponding to $[r|_i]_t, r\leftarrow R$. All parties invoke $2N_2/(n-t)$ times of $\Pi_{Randsh}(\Sigma_{2,i})$ to prepare $2N_2$ random sharings in the form of $[r|_i]_t$.
   (a) For each big group of multiplication gates, let
   $$\{[a_i|_i]_t,[b_i|_i]_t\}_{i=1}^K$$
   be the unused random sharings.
   (b) For all $i\in\{1,2,...,K\}$, all parties invoke $\Pi_{SingleMult}$ on $(i,[a_i|_i]_t,[b_i|_i]_t)$ and receive $[c_i|_i]_t$.
   (c) Let $\mathbf{e}_i\in R^K$ be the i-th unit vector. All parties locally transform $\mathbf{e}_i$ to the degree-$(K-1)$ packed Shamir sharing $[\mathbf{e}_i]_{K-1}$. Then, all parties locally compute
   $$[\mathbf{a}]_{n-K}=\sum_{i=1}^K[\mathbf{e}_i]_{K-1}*[a_i|_i]_t,$$
   $$[\mathbf{b}]_{n-K}=\sum_{i=1}^K[\mathbf{e}_i]_{K-1}*[b_i|_i]_t,$$
   $$[\mathbf{c}]_{n-K}=\sum_{i=1}^K[\mathbf{e}_i]_{K-1}*[c_i|_i]_t.$$

(3) **Preparing Random Masked Sharings for Multiplication Gates**: Let Σ_3 be the secret sharing scheme corresponding to $[\mathbf{0}]_{n-1}$. All parties invoke $2N_2/(n-t)$ times of $\Pi_{Randsh}(\Sigma_3)$ to prepare $2N_2$ random sharings in the form of $[\mathbf{0}]_{n-1}$. Let Σ_4 be the secret sharing scheme corresponding to $[\mathbf{o}]_{n-1}$, $\mathbf{o}\in(\ker\psi)^K$. All parties invoke $N_2/(n-t)$ times of $\Pi_{Randsh}(\Sigma_4)$ to prepare N_2 random sharings in the form of $[\mathbf{o}]_{n-1}$.

(4) **Preparing Random Masked Sharings for Input and Output Gates**: Let N_3 be the number of big groups of inputs gates and output gates. All parties invoke $N_3/(n-t)$ times of $\Pi_{Randsh}(\Sigma_4)$ to prepare N_3 random sharings in the form of $[\mathbf{o}]_{n-1}$.

Figure 6: Preprocessing Circuit-Independent Protocol

Proposition 4.1. *Protocol $\Pi_{PredInd}$ computes $\mathcal{F}_{PrepInd}$ with perfect security against a semi-honest adversary who controls $t=\lfloor\frac{n-1}{2}\rfloor$ servers. The amortized communication complexity per multiplication gate in this phase is approximately $7nd/l$ elements in $\mathbb{Z}_{p^k}$.*

Proof. Let $Corr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

As stated in the previous section, the amortized communication of one random share generated by Π_{Randsh} is $\frac{n(n-1)}{n-K}\approx\frac{4}{3}n$ elements in R. For each multiplication gate, the communication in Step (1) is $\frac{4}{3}nd$ elements in $\mathbb{Z}_{p^k}$. For each big group of multiplication gates, the communication in Step (2) is approximately $\frac{22}{3}Kn$ elements in R. So the amortized communication complexity per multiplication gate in this phase is approximately $\frac{22}{3}nd/l$ elements in $\mathbb{Z}_{p^k}$.

The simulator $\mathcal{S}$ works as follows.

Simulating Π_{Randsh}.

(1) In Step (1), $\mathcal{S}$ follows the protocol to agree on a super-invertible matrix M.
(2) In Step (2), for each honest party P_i, $\mathcal{S}$ generates a random Σ-sharing and distributes the shares to corrupted parties. For each corrupted party P_i, $\mathcal{S}$ learns the sharing generated by P_i.
(3) In Step (3), $\mathcal{S}$ computes the shares of all parties for each Σ-sharing $R^{(i)}$.

Simulating $\Pi_{PredInd}$.

(1) In Step (1), $\mathcal{S}$ simulates $\Pi_{Randsh}(\Sigma_1)$ and sends the shares of corrupted parties to $\mathcal{F}_{PrepInd}$.
(2) In Step (2), $\mathcal{S}$ simulates $\Pi_{Randsh}(\Sigma_{2,i})$ for all $i\in\{1,\dots,K\}$ as above. Then $\mathcal{S}$ emulates $\Pi_{SingleMult}$. Finally, for each big group of multiplication gates, $\mathcal{S}$ computes the shares of $([\mathbf{a}]_{n-K},[\mathbf{b}]_{n-K},[\mathbf{c}]_{n-K})$ of corrupted parties, and sends them to $\mathcal{F}_{PrepInd}$.
(3) In Step (3) and (4), $\mathcal{S}$ simulates $\Pi_{Randsh}(\Sigma_3)$ and $\Pi_{Randsh}(\Sigma_4)$ as described above. Then $\mathcal{S}$ sends the shares of corrupted parties to $\mathcal{F}_{PrepInd}$.

This completes the description of $\mathcal{S}$.

We note that the only place where honest parties need to send messages to corrupted parties is in Step (2) of Π_{Randsh}. The simulator S honestly generates a random Σ-sharing as that in the real world. Thus, S perfectly simulates the behaviors of honest parties.

Then, we show that the output of honest parties in both worlds have the same distribution. Any $(n-t)\times(n-t)$ submatrix of M is invertible. Given the Σ-sharings prepared by corrupted parties and the shares of corrupted parties, there is a one-to-one map between $\{S^{(i)}\}_{i\in\mathcal{H}}$ and $\{R^{(i)}\}_{i=1}^{n-t}$. Since $\{S^{(i)}\}_{i\in\mathcal{H}}$ are random Σ-sharings, so are $\{R^{(i)}\}_{i=1}^{n-t}$. Thus, the random sharings generated in Step (2) and Step (3) have the same distribution in both worlds.

For Step 2 in $\Pi_{PrepInd}$, following from the same argument as above, all parties hold random degree-t Shamir sharings from $\Pi_{Randsh}(\Sigma_{2,i})$ for $i\in\{1,\dots,K\}$. For each big group of multiplication gates, given the shares of $([a_i|_i]_t,[b_i|_i]_t)_{i=1}^K$ of corrupted parties, a_i,b_i are uniformly random values in the real world. Then, all parties receive $[c_i|_i]_t$ from $\Pi_{SingleMult}$ such that $c_i=a_ib_i$. Finally, all parties locally compute

$$[\mathbf{a}]_{n-K}=\sum_{i=1}^K[\mathbf{e}_i]_{K-1}*[a_i|_i]_t,\ [\mathbf{b}]_{n-K}=\sum_{i=1}^K[\mathbf{e}_i]_{K-1}*[b_i|_i]_t$$

and

$$[\mathbf{c}]_{n-K} = \sum_{i=1}^{K} [\mathbf{e}_i]_{K-1} * [c_i|_i]_t.$$

In the ideal world, $\mathbf{a}$ is a uniform vector sampled by $\mathcal{F}_{PrepInd}$ and the shares are provided by the simulator $\mathcal{S}$. Recall that $\mathcal{S}$ simply follows the protocol to compute the shares of corrupted parties. Thus, $[\mathbf{a}]_{n-K}$ has the same distribution in both worlds. The same argument works for $[\mathbf{b}]_{n-K}$ and $[\mathbf{c}]_{n-K}$. □

4.3 Circuit-Dependent Preprocessing

In Figure 7, we present the ideal functionality $\mathcal{F}_{Prep}$ that prepares correlated randomness for the online phase.

Some correlated randomness is prepared in $\mathcal{F}_{PrepInd}$. The random values over $\mathbb{Z}_{p^k}$ are assembled into R by $\Pi_{ConvertToRing}$. In Figure 8, we present the protocol Π_{Prep} that implements $\mathcal{F}_{Prep}$.

In Step (3), $y_i = \Phi(emb(\lambda_{\alpha_{il-l+1}}) \cdot \mathbf{1}, \dots, emb(\lambda_{\alpha_{il}}) \cdot \mathbf{1})$, $i \in \{1 \dots, K\}$. Then $\boldsymbol{\lambda_\alpha} = (y_1, \dots, y_K)$ satisfies that applying ψ to every entry gives $(\lambda_{\alpha_1}, \dots, \lambda_{\alpha_{Kl}})$. Adding $[\mathbf{o}^{(1)}]_{n-1}$ makes $[\boldsymbol{\lambda_\alpha} + \mathbf{a}]_{n-1}$ a random degree-$(n-1)$ Shamir sharing.

In Step (4), $y_i = \Phi(emb(\lambda_{\alpha_{il-l+1}}) \cdot \mathbf{1}, \dots, emb(\lambda_{\alpha_{il}}) \cdot \mathbf{1})$, $i \in \{1 \dots, K\}$. Then $\boldsymbol{\lambda_\alpha} = (y_1, \dots, y_K) + \mathbf{o}$ satisfies that applying ψ to every entry gives $(\lambda_{\alpha_1}, \dots, \lambda_{\alpha_{Kl}})$. $\mathbf{o} \in (\ker \psi)^K$ ensures that for the output wires of multiplication gates, $\boldsymbol{\lambda_\alpha} \in R^K$ hides the products well.

Proposition 4.2. *Protocol Π_{Prep} computes $\mathcal{F}_{Prep}$ with perfect security in the $\mathcal{F}_{PrepInd}$-hybrid model against a semi-honest adversary who controls t servers. The amortized communication complexity per multiplication gate in this phase is approximately $\frac{8}{3}nd^2/l + \frac{13}{3}nd/l$ elements in $\mathbb{Z}_{p^k}$.*

Proof. Let $Corr$ denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

The communication of $\Pi_{ConvertToRing}$ is $\frac{8}{3}nd^2 + \frac{7}{3}nd$ elements in $\mathbb{Z}_{p^k}$. Also, the communication in Step (3).(b) is approximately $2(n-1)$ elements in R. So the amortized communication of one multiplication gate over $\mathbb{Z}_{p^k}$ is approximately $\frac{8}{3}nd^2/l + \frac{13}{3}nd/l$.

The simulator $\mathcal{S}$ works as follows. The part of simulating Π_{Randsh} is similar with that in the proof of Proposition 2.

Simulating $\Pi_{ConvertToRing}$.

(1) In Step (1), $\mathcal{S}$ simulates Π_{Randsh} and obtains an unused tuple $([r_1 \cdot \mathbf{1}]_{n-K}, \dots, [r_d \cdot \mathbf{1}]_{n-K}, [r \cdot \mathbf{1}]_{n-K})$, where $r = iso(r_1, \dots, r_d)$.
(2) $\mathcal{S}$ knows the corrupted parties' shares of inputs. $\mathcal{S}$ samples $x_1, \dots, x_l \in im(emb)$ and generates $[x_1 \cdot \mathbf{1}]_{n-K}, \dots, [x_l \cdot \mathbf{1}]_{n-K}$ given the corrupted parties' shares. In Step (2), $\mathcal{S}$ computes the shares of $([y_1 \cdot \mathbf{1}]_{n-K}, \dots, [y_d \cdot \mathbf{1}]_{n-K})$.
(3) If P_1 is honest, then $\mathcal{S}$ reconstructs $y_1 - r_1, \dots, y_d - r_d$ and computes $u = iso(y_1 - r_1, \dots, y_d - r_d)$. $\mathcal{S}$ generates a random Shamir sharing $[u \cdot \mathbf{1}]_{n-K}$ and distributes the shares of corrupted parties to them.
If P_1 is corrpted, then $\mathcal{S}$ sends the shares of $[(y_1 - r_1) \cdot \mathbf{1}]_{n-K}$, $[(y_d - r_d) \cdot \mathbf{1}]_{n-K}$ to P_1. $\mathcal{S}$ receives the shares of $[u \cdot \mathbf{1}]_{n-K}$ of honest parties from P_1.
Finally, $\mathcal{S}$ computes the shares of $[y \cdot \mathbf{1}]_{n-K}$ of all parties.

Simulating Π_{Prep}.

$\mathcal{F}_{Prep}$

(1) **Assign Random Values to Wires**: $\mathcal{F}_{Prep}$ receives the set of corrupted parties, denoted by $Corr$. $\mathcal{F}_{Prep}$ receives the circuit C from all parties.
  (a) For each output wire α of an input gate or a multiplication gate, $\mathcal{F}_{Prep}$ samples $\lambda_\alpha \leftarrow \mathbb{Z}_{p^k}$ and associates it with the wire α.
  (b) Starting from the first layer of C to the last layer, for each addition gate with input wires α, β and output wire γ, $\mathcal{F}_{Prep}$ sets $\lambda_\gamma = \lambda_\alpha + \lambda_\beta$.
(2) **Preparing Masks**:For each intermediate layer in C, all multiplication gates are divided into small groups of size l. And small groups are divided into big groups of size K. For each big group of multiplication gates with input wires $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$: write $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_{Kl})$ and let $\boldsymbol{\lambda_\alpha} \in R^K$ be a mask such that applying ψ to every entry gives $\lambda_{\alpha_1}, \dots, \lambda_{\alpha_{Kl}}$. The definition of $\boldsymbol{\lambda_\beta}$ is similar.
(3) **Preparing Packed Beaver Triples**: For each big group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$:
  (a) $\mathcal{F}_{Prep}$ receives from the adversary a set of shares $\{(u_j^{(1)}, u_j^{(2)}, u_j^{(3)})\}_{j \in Corr}$. $\mathcal{F}_{Prep}$ samples random $\mathbf{a}, \mathbf{b} \in R^K$ and computes $\mathbf{c} = \mathbf{a} * \mathbf{b}$. Then $\mathcal{F}_{Prep}$ computes three degree-$(n-K)$ packed Shamir sharings $[\mathbf{a}]_{n-K}, [\mathbf{b}]_{n-K}, [\mathbf{c}]_{n-K}$ such that for all $j \in Corr$, P_j's share of $([\mathbf{a}]_{n-K}, [\mathbf{b}]_{n-K}, [\mathbf{c}]_{n-K})$ is $(u_j^{(1)}, u_j^{(2)}, u_j^{(3)})$.
  (b) $\mathcal{F}_{Prep}$ distributes the shares of $([\mathbf{a}]_{n-K}, [\mathbf{b}]_{n-K}, [\mathbf{c}]_{n-K})$ to honest parties.
  (c) $\mathcal{F}_{Prep}$ computes $\boldsymbol{\lambda_\alpha} + \mathbf{a}$ and $\boldsymbol{\lambda_\beta} + \mathbf{b}$ and sends them to P_1.
(4) **Preparing Degree-$(n-1)$ Packed Shamir Sharings**: For the input layer, all input gates are divided into small groups of size l such that the input gates of each small group belong to the same client. The small groups are divided into big groups of size K such that the input gates of each big group belong to the same client. For each group of input gates with output wires $\boldsymbol{\alpha}$:
  (a) $\mathcal{F}_{Prep}$ receives from the adversary a set of shares $\{u_j\}_{j \in Corr}$. $\mathcal{F}_{Prep}$ samples a random degree-$n-1$ packed Shamir sharing $[\boldsymbol{\lambda_\alpha}]_{n-1}$ such that for all $j \in Corr$, P_j's share of $[\boldsymbol{\lambda_\alpha}]_{n-1}$ is u_j.
  (b) $\mathcal{F}_{Prep}$ distributes the shares of $[\boldsymbol{\lambda_\alpha}]_{n-1}$ to honest parties.

  Similarly, for the output layer in C, all output gates are divided into small groups and big groups such that the output gates of each group belong to the same client. For each group of output gates with input wires $\boldsymbol{\lambda}$, $\mathcal{F}_{Prep}$ prepares and distributes $[\boldsymbol{\lambda_\alpha}]_{n-1}$ in the same way as above. For each big group of multiplication gates and output wires $\boldsymbol{\gamma}$, $\mathcal{F}_{Prep}$ prepares and distributes $[\boldsymbol{\lambda_\gamma}]_{n-1}$ in the same way as above.

Figure 7: Preprocessing Functionality

Π_{Prep}

1. **Circuit-Independent Preprocessing Phase**: All parties invoke $\mathcal{F}_{PrepInd}$ to receive correlated randomness.
2. **Assign Random Values to Wires**: For each output wire α of input gates and multiplication gates, all parties receive $[emb(\lambda_\alpha) \cdot \mathbf{1}]_{n-K}$ from $\mathcal{F}_{PrepInd}$. For every addition gate , all parties locally compute the addition of these masks as in Step (1) of $\mathcal{F}_{Prep}$.
3. **Preparing Packed Beaver Triples**: For each big group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$. Let $\mathbf{e}_i \in R^K$ be the i-th unit vector. All parties invoke $\Pi_{ConvertToRing}([emb(\lambda_{\alpha_{il-l+1}}) \cdot \mathbf{1}]_{n-K}, \ldots, [emb(\lambda_{\alpha_{il}}) \cdot \mathbf{1}]_{n-K})$ and obtain $[y_i \cdot \mathbf{1}]_{n-K}, i \in \{1, \ldots, K\}$. All parties locally compute $[\boldsymbol{\lambda_\alpha}]_{n-1} = \sum_{i=1}^{K} \mathbf{e}_i * [y_i \cdot \mathbf{1}]_{n-K}$. Similarly, all parties compute $[\boldsymbol{\lambda_\beta}]_{n-1}$. All parties use $([\mathbf{a}]_{n-K}, [\mathbf{o}^{(1)}]_{n-1})$ prepared in $\mathcal{F}_{PrepInd}$ to reconstruct $\boldsymbol{\lambda_\alpha} + \mathbf{a}$ to P_1:
   (a) All parties locally compute $[\boldsymbol{\lambda_\alpha} + \mathbf{a}]_{n-1} = [\boldsymbol{\lambda_\alpha}]_{n-1} + [\mathbf{a}]_{n-K} + [\mathbf{o}^{(1)}]_{n-1}$.
   (b) P_1 collects the whole sharing $[\boldsymbol{\lambda_\alpha} + \mathbf{a}]_{n-1}$ and reconstructs $\boldsymbol{\lambda_\alpha} + \mathbf{a}$.
   All parties compute $[\boldsymbol{\lambda_\beta} + \mathbf{b}]_{n-1}$ and reconstruct the secret to P_1 in a similar way. Note that $\mathbf{o}^{(1)} = \mathbf{o}^{(2)} = \mathbf{0}$ here.
4. **Preparing Degree-$(n-1)$ Packed Shamir Sharings**: For each big group of input gates, let $\boldsymbol{\alpha}$ be the output wires of these gates. Recall $[\mathbf{o}]_{n-1}$ is prepared in $\mathcal{F}_{PrepInd}$, $\mathbf{o} \in (\ker \psi)^K$. All parties invoke $\Pi_{ConvertToRing}([emb(\lambda_{\alpha_{il-l+1}}) \cdot \mathbf{1}]_{n-K}, \ldots, [emb(\lambda_{\alpha_{il}}) \cdot \mathbf{1}]_{n-K})$ and obtain $[y_i \cdot \mathbf{1}]_{n-K}, i \in \{1, \ldots, K\}$. All parties locally compute $[\boldsymbol{\lambda_\alpha}]_{n-1} = \sum_{i=1}^{K} \mathbf{e}_i * [y_i \cdot \mathbf{1}]_{n-K} + [\mathbf{o}]_{n-1}$. The same step is done for the input wires of each big group of output gates and output wires of each big group of multiplication gates.

Figure 8: Preprocessing Protocol

1. In Step (1), $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{PrepInd}$ and learns the shares of corrupted parties.
2. The simulator for Π_{Randsh} is the same with the simulator defined in the proof of Proposition 2 for Π_{Randsh}. In Step (2) of Π_{Prep}, $\mathcal{S}$ follows the protocol to compute the shares of corrupted parties for $[emb(\lambda_\alpha) \cdot \mathbf{1}]_{n-K}$.
3. In Step (3), for each big group of multiplication gates, $\mathcal{S}$ follows the protocol to compute the shares of $[\boldsymbol{\lambda_\alpha}]_{n-1}$ and $[\boldsymbol{\lambda_\beta}]_{n-1}$ of corrupted parties. In Step (3).(b), $\mathcal{S}$ follows the protocol to compute the shares of $[\boldsymbol{\lambda_\alpha} + \mathbf{a}]_{n-1}$ and $[\boldsymbol{\lambda_\beta} + \mathbf{b}]_{n-1}$ of corrupted parties and sets the shares of honest parties to be uniform elements in R. If P_1 is honest, then $\mathcal{S}$ honestly follows the protocol. Otherwise, $\mathcal{S}$ sends the shares of $[\boldsymbol{\lambda_\alpha} + \mathbf{a}]_{n-1}$ and $[\boldsymbol{\lambda_\beta} + \mathbf{b}]_{n-1}$ of honest parties to P_1.
4. In Step (4), for each big group of input gates, $\mathcal{S}$ follows the protocol to compute the shares of $[\boldsymbol{\lambda_\alpha}]_{n-1}$ held by corrupted parties and sends them to $\mathcal{F}_{Prep}$. $\mathcal{S}$ does the same for the input wires of each big group of output gates and multiplication gates.

This completes the description of $\mathcal{S}$.

For the distributions of variables of $\Pi_{ConvertToRing}$, the secrets $x_1, \ldots, x_l \in im\, emb$ are uniformly random given the shares of corrupted parties. After sampling $x_1, \ldots, x_l \in im\, emb$, $\mathcal{S}$ follows $\Pi_{ConvertToRing}$. So the distributions of variables of $\Pi_{ConvertToRing}$ in two worlds are the same.

Now we show the distribution of Π_{Prep} in the real world and the distribution of $\mathcal{F}_{Prep}$ in the ideal world are prefectly indistinguishable. The argument for Step 2 is similar to the argument in Proposition 2. The honest parties need to send messages to corrupted parties in Step (3).(b). Note that $\mathbf{a}, \mathbf{b} \in R^K$ are uniformly random given the shares of $[\mathbf{a}]_{n-K}, [\mathbf{b}]_{n-K}$ of corrupted parties. The random degree-$(n-1)$ packed Shamir sharing $[\mathbf{o}^{(1)}]_{n-1}$ makes $[\boldsymbol{\lambda_\alpha} + \mathbf{a}]_{n-1}$ a random degree -$(n-1)$ packed Shamir sharing, which satisfies that the shares of honest parties are uniformly random given the shares of corrupted parties. Thus, the shares of honest parties have the same distribution in both worlds. The output $\boldsymbol{\lambda_\alpha} + \mathbf{a}$ and $\boldsymbol{\lambda_\beta} + \mathbf{b}$ have the same distribution in both worlds.

In Step (4), $\mathcal{S}$ computes the shares of corrupted parties. In the real world, since $[\mathbf{o}]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing of an element in $(\psi^{-1}(\mathbf{0}), \ldots, (\psi^{-1}(\mathbf{0})) \subset R^K$, $[\boldsymbol{\lambda_\alpha}]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing given the shares of corrupted parties. In the ideal world, $\mathcal{F}_{Prep}$ generates a random degree-$(n-1)$ packed Shamir sharing of $\boldsymbol{\lambda_\alpha}$ given the shares of corrupted parties. Therefore, the sharing $[\boldsymbol{\lambda_\alpha}]_{n-1}$ has the same distribution in both worlds.

□

4.4 Online Protocol

Our online protocol consists of four phases: Preprocessing Phase, Input Phase, Computation Phase, and Output Phase. The preprocessing phase is handled by $\mathcal{F}_{Prep}$. In the online phase, for each wire α, P_1 learns $\mu_\alpha = v_\alpha - \lambda_\alpha$, where v_α is the real value.

In Figure 9, we show the protocol Π_{Input} for clients to input securely.

Π_{Input}

1. For each group of input gates that belong to *Client*, let $\boldsymbol{\alpha}$ denote the batch of output wires of these input gates. All parties receive the sharing $[\boldsymbol{\lambda_\alpha}]_{n-1}$ from $\mathcal{F}_{Prep}$ and *Client* holds input $\mathbf{v}_{\boldsymbol{\alpha}}$.
2. *Client* collects the shares of $[\boldsymbol{\lambda_\alpha}]_{n-1}$ and reconstructs $\boldsymbol{\lambda_\alpha}$.
3. *Client* computes $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ and sends $\boldsymbol{\mu_\alpha}$ to P_1.

Figure 9: Input Protocol

In the input phase, *Client* reconstructs $\boldsymbol{\lambda_\alpha}$, computes $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$ and sends $\boldsymbol{\mu_\alpha}$ to P_1.

In the computation phase, the circuit is evaluated layer by layer. For an addition gate, with input wires α, β and output wire γ, $v_\gamma = v_\alpha + v_\beta$ and $\lambda_\gamma = \lambda_\alpha + \lambda_\beta$. So P_1 can locally compute

$$\mu_\gamma = \mu_\alpha + \mu_\beta.$$

Π_{Mult}

(1) For each big group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties receive from $\mathcal{F}_{Prep}$ a packed Beaver triple $([\mathbf{a}]_{n-K}, [\mathbf{b}]_{n-K}, [\mathbf{c}]_{n-K})$ and $[\boldsymbol{\lambda}_{\boldsymbol{\gamma}}]_{n-1}$. P_1 receives from $\mathcal{F}_{Prep}$ two vectors $\mathbf{d}_1 = \boldsymbol{\lambda}_{\boldsymbol{\alpha}} + \mathbf{a}$, $\mathbf{d}_2 = \boldsymbol{\lambda}_{\boldsymbol{\beta}} + \mathbf{b}$. P_1 also learns $\boldsymbol{\mu}_{\boldsymbol{\alpha}}, \boldsymbol{\mu}_{\boldsymbol{\beta}}$ during the online phase;

(2) P_1 locally compute $\boldsymbol{v}_{\boldsymbol{\alpha}} + \mathbf{a} = \boldsymbol{\mu}_{\boldsymbol{\alpha}} + \mathbf{d}_1$ and $\boldsymbol{v}_{\boldsymbol{\beta}} + \mathbf{b} = \boldsymbol{\mu}_{\boldsymbol{\beta}} + \mathbf{d}_2$. Then P_1 computes $[\boldsymbol{v}_{\boldsymbol{\alpha}} + \mathbf{a}]_{K-1}$ and $[\boldsymbol{v}_{\boldsymbol{\beta}} + \mathbf{b}]_{K-1}$ and distributes the shares to all parties;

(3) All parties locally compute

$$\begin{aligned}[\mathbf{d}]_{n-1} =& [\boldsymbol{v}_{\boldsymbol{\alpha}} + \mathbf{a}]_{K-1} * [\boldsymbol{v}_{\boldsymbol{\beta}} + \mathbf{b}]_{K-1} - \\ & [\boldsymbol{v}_{\boldsymbol{\alpha}} + \mathbf{a}]_{K-1} * [\mathbf{b}]_{n-K} - \\ & [\boldsymbol{v}_{\boldsymbol{\beta}} + \mathbf{b}]_{K-1} * [\mathbf{a}]_{n-K} + [\mathbf{c}]_{n-K} - [\boldsymbol{\lambda}_{\boldsymbol{\gamma}}]_{n-1};\end{aligned}$$

(4) P_1 collects the whole sharing $[\mathbf{d}]_{n-1}$ and reconstructs $\mathbf{d} \in R^K$. Apply $\phi \circ \psi$ to every entry of $\mathbf{d}$ and the result is $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$.

Figure 10: Multiplication Protocol

For multiplication gates, we follow the technique of packed Beaver triples in [18] and present Π_{Mult} in Figure 10.

For correctness, we assume all parties follow Π_{Mult}. Then P_1 obtains $\mathbf{d} = \mathbf{v}_{\boldsymbol{\alpha}} * \mathbf{v}_{\boldsymbol{\beta}} - \boldsymbol{\lambda}_{\boldsymbol{\gamma}} \in R^K$. Applying $\phi \circ \psi$ to all entries of $\boldsymbol{\lambda}_{\boldsymbol{\gamma}}$ gives $(\phi(\lambda_{\gamma_1}, \dots, \lambda_{\gamma_l}), \dots, \phi(\lambda_{\gamma_{Kl-l+1}}, \dots, \lambda_{\gamma_{Kl}}))$. Write

$$\mathbf{v}_{\boldsymbol{\alpha}} = (\phi(v_{\alpha_1}, \dots, v_{\alpha_l}), \dots)$$

and

$$\mathbf{v}_{\boldsymbol{\beta}} = (\phi(v_{\beta_1}, \dots, v_{\beta_l}), \dots).$$

Applying $\phi \circ \psi$ to all entries of $\mathbf{v}_{\boldsymbol{\alpha}} * \mathbf{v}_{\boldsymbol{\beta}}$ gives

$$(\phi(v_{\alpha_1} v_{\beta_1}, \dots, v_{\alpha_l} v_{\beta_l}), \dots) = (\phi(v_{\gamma_1}, \dots, v_{\gamma_l}), \dots)$$

So applying $\phi \circ \psi$ to all entries of $\mathbf{d}$ gives $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$. This is called ReEncode in [8]. But we merge ReEncode into multiplication in order to avoid extra communication cost.

Now we analyse the communication complexity of Π_{Mult}. For each big group, it requires to communicate $3(n-1)$ elements in R, which are $3d(n-1)$ elements in $\mathbb{Z}_{p^k}$. The amortized communication complexity per gate is $\frac{3d(n-1)}{Kl} \approx 12d/l$.

In the output phase, for each big group of output gates that belong to some *Client*, let $\boldsymbol{\alpha}$ denote the input wires of these output gates. P_1 learns $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$ and sends it to *Client*. *Client* reconstructs $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ and computes $\boldsymbol{v}_{\boldsymbol{\alpha}} = \boldsymbol{\mu}_{\boldsymbol{\alpha}} + \boldsymbol{\lambda}_{\boldsymbol{\alpha}}$.

We are now ready to give the main functionality $\mathcal{F}_{Main}$ in Figure 11 and the online protocol Π_{Online} in Figure 12.

$\mathcal{F}_{Main}$

(1) $\mathcal{F}_{Main}$ receives the input from all clients. Let x denote the input and C denote the circuit.

(2) $\mathcal{F}_{Main}$ computes $C(x)$ and distributes the output to all clients.

Figure 11: Main Functionality

Π_{Online}

(1) **Preprocessing Phase**: All parties invoke $\mathcal{F}_{Prep}$ to receive correlated randomness that will be used in the online phase.

(2) **Input Phase**: In the input layer, for each big group of Kl input gates that belong to some *Client*, let $\boldsymbol{\alpha}$ denote the output wires of these input gates. All parties and *Client* invoke Π_{Input}. At the end of the protocol, P_1 learns $\boldsymbol{\mu}_{\boldsymbol{\alpha}} = \boldsymbol{v}_{\boldsymbol{\alpha}} - \boldsymbol{\lambda}_{\boldsymbol{\alpha}} \in R^K$.

(3) **Computation Phase**: All parties maintain the invariant that for each wire α, P_1 learns $\mu_\alpha = v_\alpha - \lambda_\alpha \in \mathbb{Z}_{p^k}$, where v_α is the real value and λ_α is a random mask. The circuit is evaluated layer by layer. Assume that the invariant holds for wires in previous layers. Consider gates in the current layer.
For each addition gate with input wires α, β and output wire γ, P_1 locally compute $\mu_\gamma = \mu_\alpha + \mu_\beta$. For each big group of Kl multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, all parties invoke Π_{Mult}. At the end of the protocol, P_1 learns $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$.

(4) **Output Phase**: For each big group of output gates that belong to some *Client*, let $\boldsymbol{\alpha}$ denote the input wires of these output gates. Recall that all parties receive $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ from $\mathcal{F}_{Prep}$ and P_1 learns $\boldsymbol{\mu}_{\boldsymbol{\alpha}} = \boldsymbol{v}_{\boldsymbol{\alpha}} - \boldsymbol{\lambda}_{\boldsymbol{\alpha}}$. All parties send their shares of $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ to *Client*. P_1 sends $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$ to *Client*. Then *Client* reconstructs $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ and computes $\boldsymbol{v}_{\boldsymbol{\alpha}} = \boldsymbol{\mu}_{\boldsymbol{\alpha}} + \boldsymbol{\lambda}_{\boldsymbol{\alpha}}$. Applying ψ to all entries of $\boldsymbol{v}_{\boldsymbol{\alpha}}$ gives $(v_{\alpha_1}, \dots, v_{\alpha_{Kl}})$.

Figure 12: Main Protocol

THEOREM 4.3. *Protocol Π_{Online} computes $\mathcal{F}_{Main}$ with perfect security in the $\mathcal{F}_{Prep}$-hybrid model against a semi-honest adversary who controls t parties and corrupts up to c of the clients. The amortized online communication complexity per multiplication gate is approximately $12d/l$ elements in $\mathbb{Z}_{p^k}$.*

PROOF. Let $\mathcal{A}$ denote the adversary. Let *Corr* denote the set of corrupted parties and $\mathcal{H}$ denote the set of honest parties.

The correctness of addition gates follows from the construction of random masks assigned to wires. The correctness of multiplication gates follows from the discussion about Π_{Mult}. Then the correctness of Π_{Online} follows.

We now describe the construction of the simulator $\mathcal{S}$.

(1) In the preprocessing phase, $\mathcal{S}$ emulates the ideal functionality $\mathcal{F}_{Prep}$ and receives the shares of corrupted parties for each packed Shamir sharing. If P_1 is corrupted, then $\mathcal{S}$ receives $\mathbf{d}_1 = \boldsymbol{\lambda}_{\boldsymbol{\alpha}} + \mathbf{a}$ and $\mathbf{d}_2 = \boldsymbol{\lambda}_{\boldsymbol{\beta}} + \mathbf{a}$ in Step (3) of $\mathcal{F}_{Prep}$.

(2) In the input phase, for each big group of input gates that belong to some *Client*, let $\boldsymbol{\alpha}$ denote the batch of output wires of these input gates.
If *Client* is honest, then $\mathcal{S}$ samples random values as $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$ and sends them to P_1 if P_1 is corrupted.
If *Client* is corrupted, then $\mathcal{S}$ holds the corrupted parties's shares of $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ from $\mathcal{F}_{Prep}$. $\mathcal{S}$ samples random values as $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$. Then based on the secrets $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ and the shares of corrupted parties, $\mathcal{S}$ randomly samples the whole sharing

$[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ and sends the shares of $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ held by honest parties to *Client*. From the inputs $\mathbf{v}_{\boldsymbol{\alpha}}$ of *Client*, $\mathcal{S}$ computes $\boldsymbol{\mu}_{\boldsymbol{\alpha}} = \boldsymbol{v}_{\boldsymbol{\alpha}} - \boldsymbol{\lambda}_{\boldsymbol{\alpha}}$.

(3) In the computation phase, we will maintain the invariant that μ_α for each wire α is known to $\mathcal{S}$.
For each addition gate with input wires α, β and output wire, $\mathcal{S}$ honestly computes $\mu_\gamma = \mu_\alpha + \mu_\beta$.
For each big group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, $\mathcal{S}$ simulates Π_{Mult} as follows.
If P_1 is honest, then $\mathcal{S}$ picks random $\mathbf{d}_1, \mathbf{d}_2 \in R^K$ and computes $\boldsymbol{v}_{\boldsymbol{\alpha}} + \mathbf{a} = \boldsymbol{\mu}_{\boldsymbol{\alpha}} + \mathbf{d}_1$, $\boldsymbol{v}_{\boldsymbol{\beta}} + \mathbf{b} = \boldsymbol{\mu}_{\boldsymbol{\beta}} + \mathbf{d}_2$. $\mathcal{S}$ computes degree-$(K-1)$ packed Shamir sharings $[\boldsymbol{v}_{\boldsymbol{\alpha}} + \mathbf{a}]_{K-1}$ and $[\boldsymbol{v}_{\boldsymbol{\beta}} + \mathbf{b}]_{K-1}$ and distributes them to corrupted parties. $\mathcal{S}$ distributes the shares of corrupted parties. $\mathcal{S}$ computes the shares of $[\mathbf{d}]_{n-1}$ of corrupted parties and samples $\mathbf{d} \leftarrow R^K$. $\mathcal{S}$ generates a random $[\mathbf{d}]_{n-1}$ given the shares of corrupted parties and $\mathbf{d}$.
If P_1 is corrupted, then P_1 receives $\mathbf{d}_1 = \boldsymbol{\lambda}_{\boldsymbol{\alpha}} + \mathbf{a}$ and $\mathbf{d}_2 = \boldsymbol{\lambda}_{\boldsymbol{\beta}} + \mathbf{b}$ from $\mathcal{F}_{Prep}$. $\mathcal{S}$ receives the shares of $[\boldsymbol{v}_{\boldsymbol{\alpha}} + \mathbf{a}]_{K-1}$ and $[\boldsymbol{v}_{\boldsymbol{\beta}} + \mathbf{b}]_{K-1}$ of honest parties from P_1. $\mathcal{S}$ follows Step 3 in Π_{Mult} to compute shares of $[\mathbf{d}]_{n-1}$ of corrupted parties. $\mathcal{S}$ samples $\mathbf{d} \leftarrow R^K$. $\mathcal{S}$ generates a random $[\mathbf{d}]_{n-1}$ given the shares of corrupted parties and $\mathbf{d}$. $\mathcal{S}$ sends the shares of honest parties to P_1. $\mathcal{S}$ reconstructs $\mathbf{d}$.
$\mathcal{S}$ computes $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$ from $\mathbf{d}$ as in Π_{Mult}.

(4) In the output phase, for each big group of output gates that belong to some *Client*, let $\boldsymbol{\alpha}$ denote the batch of input wires of these output gates.
If *Client* is honest, then $\mathcal{S}$ does nothing.
If *Client* is corrupted, $\mathcal{S}$ sends the inputs of corrupted clients to $\mathcal{F}_{Main}$. Then $\mathcal{S}$ receives the outputs $\boldsymbol{v}_{\boldsymbol{\alpha}}$ from $\mathcal{F}_{Main}$. $\mathcal{S}$ computes $\boldsymbol{\lambda}_{\boldsymbol{\alpha}} = \boldsymbol{v}_{\boldsymbol{\alpha}} - \boldsymbol{\mu}_{\boldsymbol{\alpha}}$. Note that this $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ is not the one from $\mathcal{F}_{Prep}$ but $\mathcal{S}$ stores the shares of corrupted parties. Based on the secrets $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ and the shares of corrupted parties, $\mathcal{S}$ randomly samples the whole sharing $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$. $\mathcal{S}$ sends to *Client* the shares of $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ of honest parties. If P_1 is honest, $\mathcal{S}$ also sends to *Client* $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$.

This completes the description of $\mathcal{S}$.

We now show that the distributions in two worlds are the same. It is sufficient to focus on the places where honest parties and clients need to communicate with corrupted parties and clients.

In the input phase, for each big group of input gates that belong to some *Client*, let $\boldsymbol{\alpha}$ denote the batch of output wires of these input gates.

If *Client* is honest, then $\mathcal{S}$ needs to simulate the values $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$ sent from *Client* to P_1. In the ideal world, $\mathcal{S}$ simply samples random elements as $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$. Since the masks $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ are uniformly random in $(im\phi)^K$, so are $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$.

If *Client* is corrupted, then $\mathcal{S}$ needs to simulate the shares of $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ of honest parties. Since $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing, the secrets $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ are uniformly random given the shares of corrupted parties. In the ideal world, $\mathcal{S}$ randomly samples $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ and then randomly samples the shares of honest parties based on the secrets $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ and the shares of corrupted parties. Therefore, the distribution of the shares of $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ of honest parties is identical to that in the real world. Note that from the inputs of *Client*, $\mathcal{S}$ can also compute $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$.

In the computation phase, we will show that $\mathcal{S}$ can always learn μ_α for each wire α in the circuit. Furthermore, $\{\mu_\alpha\}_\alpha$ has the same distribution as that in the real world.

For an addition gate with input wires α, β and output wire γ, $\mathcal{S}$ can compute μ_γ from μ_α and μ_β. The above statement holds.

For each group of multiplication gates with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, given the shares of $[\boldsymbol{\lambda}_{\boldsymbol{\gamma}}]_{n-1}$ of corrupted parties, the shares of honest parties are uniformly random in R. Therefore, given the shares of $[\mathbf{d}]_{n-1}$ of corrupted parties, d is random in R^K. Since $\mathcal{S}$ samples $\mathbf{d} \leftarrow R$ and then follows Π_{Mult}, the distributions in two worlds are the same.

In the output phase, for each big group of K output gates that belong to some *Client*, let $\boldsymbol{\alpha}$ denote the batch of input wires of these output gates. If *Client* is honest, honest parties and *Client* do not need to send any messages to corrupted parties. If Client is corrupted, $\mathcal{S}$ can learn the outputs of *Client* from $\mathcal{F}_{Main}$. Since $\mathcal{S}$ learns $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$, $\mathcal{S}$ can compute $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$. In both worlds, $[\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]_{n-1}$ is a random degree-$(n-1)$ packed Shamir sharing given the secrets $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ and the shares of corrupted parties. Thus, the shares of honest parties generated by $\mathcal{S}$ have the same distribution as that in the real world. □

The protocols in this section only have semi-honest security and we discuss how to achieve malicious security in Appendix A.

5 CONCLUSION

In this paper, we present a concrete construction of RMFEs that work well for $n < 2^{128}$, i.e. (ϕ, ψ) is an (l, d)-RMFE and $d/l \leq 8$. For $n < 2^{15}$, $d/l \leq 4$. The asymptotically good RMFEs [13] ensure that d/l is bounded by a constant.

The packing method based on tweaked interpolation in MHz2k [10] does not achieve constant online communication. If the degree of the cyclotomic polynomial of the BGV homomorphic encryption increases monotonically with respect to n, then the packing density decreases. On the way to constant online communication per gate, RMFEs seem to be a better packing method than the one based on tweaked interpolation.

In Section 4, we build our semi-honest MPC protocols over $\mathbb{Z}_{p^k}$ via packed Shamir secret-sharing over R. Our protocol consists of three phases:

(1) A circuit and input-independent phase, requiring communication of $7nd/l$ elements in $\mathbb{Z}_{p^k}$ per multiplication gate.
(2) A circuit-dependent but input-independent phase, requiring communication of $\frac{8}{3}nd^2/l + \frac{13}{3}nd/l$ elements in $\mathbb{Z}_{p^k}$ per multiplication gate. Since d/l is bounded by a constant, the communication complexity is $\Theta(n \log n)$.
(3) A circuit and input-dependent phase, requiring communication of $12d/l$ elements in $\mathbb{Z}_{p^k}$ per multiplication gate.

This is the first protocol over $\mathbb{Z}_{p^k}$ in the semi-honest setting such that an arithmetic circuit can be evaluated with overall communication complexity $O(1)$ elements per multiplication gate. Turbopack [18], the state-of-the-art, requires communication of 12 field elements per multiplication gate in the online phase. Our protocol need more communication but is comparable to Turbopack. Turbopack requires the circuit width to be $O(n)$, while our work requires the circuit width to be $O(n \log n)$. Turbopack assumes the field size is larger than $n + K$, while our work can fix the base ring $\mathbb{Z}_{p^k}$ even if n grows. Since modern CPUs prefer computation over $\mathbb{Z}_{p^k}$, the computation over $\mathbb{Z}_{p^k}$ is faster than that over $\mathbb{F}_{p^k}$. MPC over $\mathbb{Z}_{2^k}$ is supposed to match closely the data format used by CPUs.

As mentioned in [17], the fact that 32 and 64-bit computation has been the norm for many years means that there are many algorithms optimized for this domain. The comparison primitives constructed in [17] based on SPD$\mathbb{Z}_{2^k}$ shows a speedup of 4x over SPDZ. Note that SPD$\mathbb{Z}_{2^k}$ works over $\mathbb{Z}_{2^k}$ and SPDZ works over fields. So it is hopeful that the primitives like truncation and comparison based on rings have better performance than those based on fields. But we do not know how to construct the primitives based on our MPC protocol. We believe the core challenge is constructing the conversion primitive between arithmetic sharings and boolean sharings. The construction of the primitives above is left as challenging future work.

The typical online communication of semi-honest MPC over rings per gate is approximately $2n$ ring elements [20]. For $n = 30$, the RMFEs satisfy $d/l \leq 4$. The online communication of our protocol is $12d/l$ ring elements per gate, which is less than $2n = 60$. The communication advantage will be greater when the number of parties grows.

The concurrent work, "HYY24" [24], gives an honest majority MPC protocol for boolean circuits, while our work applies to circuits over $\mathbb{Z}_{p^k}$. HYY24 leverages RMFEs over fields to achieve constant online communication, while ours leverages RMFEs over rings with many zero divisors. HYY24 fixes the base field $\mathbb{Z}_2$ and our work fixes the base ring $\mathbb{Z}_{p^k}$. The core techniques of HYY24 and our work in the case $p^k = 2$ are the same. One noticable difference is that HYY24 does not divide the offline phase but we divide the offline phase into two phases: the circuit independent phase and the circuit dependent phase. HYY24 generates masks like $[\phi(\boldsymbol{\lambda_\alpha})|i]_{n-K}$ in the offline phase. But we need to assemble the masks λ_α via $\Pi_{ConvertToRing}$ in the circuit dependent phase.

Although the online communication complexity per gate is a constant, it is challenging to bring down the communication complexity of the preprocessing phase to $O(n)$, which is left as future work. We believe that the protocol $\Pi_{ConvertToRing}$ can be computed locally in general, but we do not know how to achieve this.

To conclude, our MPC protocol is suitable for computing arithmetic circuits over $\mathbb{Z}_{p^k}$, whose width is $O(n \log n)$. The base ring $\mathbb{Z}_{p^k}$ can be fixed when the number of parties grows. When $n > 30$, the online communication of our protocol outperforms those that have linear online communication.

ACKNOWLEDGMENTS

Thank the workmates in Shanghai Fudan Microelectronics Group Co., Ltd for inspiring discussions about multiparty computation. Also thank the anonymous reviewers for helpful comments.

REFERENCES

[1] Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. 2019. Efficient Information-Theoretic Secure Multiparty Computation over $\mathbb{Z}/p^k\mathbb{Z}$ via Galois Rings. In *Theory of Cryptography*, Dennis Hofheinz and Alon Rosen (Eds.), Vol. 11891. Springer International Publishing, Cham, 471–501. https://doi.org/10.1007/978-3-030-36030-6_19

[2] Mark Abspoel, Anders P. K. Dalskov, Daniel Escudero, and Ariel Nof. 2021. An Efficient Passive-to-Active Compiler for Honest-Majority MPC over Rings. In *Applied Cryptography and Network Security*, Kazue Sako and Nils Ole Tippenhauer (Eds.), Vol. 12727. Springer International Publishing, Cham, 122–152. https://doi.org/10.1007/978-3-030-78375-4_6

[3] Donald Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology — CRYPTO '91*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 420–432. https://doi.org/10.1007/3-540-46766-1_34

[4] Zuzana Beerliová-Trubíniová and Martin Hirt. 2006. Efficient Multi-party Computation with Dispute Control. In *Theory of Cryptography*, Shai Halevi and Tal Rabin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 305–328. https://doi.org/10.1007/11681878_16

[5] Zuzana Beerliová-Trubíniová and Martin Hirt. 2008. Perfectly-Secure MPC with Linear Communication Complexity. In *Theory of Cryptography*, Ran Canetti (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 213–230. https://doi.org/10.1007/978-3-540-78524-8_13

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the twentieth annual ACM symposium on Theory of computing* (Chicago, Illinois, USA) *(STOC'88)*, J'anos Simon (Ed.). Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/62212.62213

[7] Fabrice Benhamouda, Elette Boyle, Niv Gilboa, Shai Halevi, Yuval Ishai, and Ariel Nof. 2021. Generalized Pseudorandom Secret Sharing and Efficient Straggler-Resilient Secure Computation. In *Theory of Cryptography*, Kobbi Nissim and Brent Waters (Eds.). Springer International Publishing, Cham, 129–161. https://doi.org/10.1007/978-3-030-90453-1_5

[8] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. 2018. Amortized Complexity of Information-Theoretically Secure MPC Revisited. In *Advances in Cryptology – CRYPTO 2018*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer International Publishing, Cham, 395–426. https://doi.org/10.1007/978-3-319-96878-0_14

[9] Dario Catalano, Mario Di Raimondo, Dario Fiore, and Irene Giacomelli. 2020. Mon$\mathbb{Z}_{2^k}$a: Fast Maliciously Secure Two Party Computation on $\mathbb{Z}_{2^k}$. In *Public-Key Cryptography – PKC 2020*, Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas (Eds.). Springer International Publishing, Cham, 357–386. https://doi.org/10.1007/978-3-030-45388-6_13

[10] Jung Hee Cheon, Dongwoo Kim, and Keewoo Lee. 2021. MHz2k: MPC from HE over $\mathbb{Z}_{2^k}$ with New Packing, Simpler Reshare, and Better ZKP. In *Advances in Cryptology – CRYPTO 2021*, Tal Malkin and Chris Peikert (Eds.). Springer International Publishing, Cham, 426–456. https://doi.org/10.1007/978-3-030-84245-1_15

[11] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. 2018. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority. In *Advances in Cryptology – CRYPTO 2018*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer International Publishing, Cham, 769–798. https://doi.org/10.1007/978-3-319-96881-0_26

[12] Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. 2003. Efficient Multi-party Computation over Rings. In *Advances in Cryptology — EUROCRYPT 2003*, Eli Biham (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 596–613. https://doi.org/10.1007/3-540-39200-9_37

[13] Ronald Cramer, Matthieu Rambaud, and Chaoping Xing. 2021. Asymptotically-Good Arithmetic Secret Sharing over $\mathbb{Z}/p^\ell\mathbb{Z}$ with Strong Multiplication and Its Applications to Efficient MPC. In *Advances in Cryptology – CRYPTO 2021*, Tal Malkin and Chris Peikert (Eds.). Springer International Publishing, Cham, 656–686. https://doi.org/10.1007/978-3-030-84252-9_22

[14] Anders P. K. Dalskov, Eysa Lee, and Eduardo Soria-Vazquez. 2020. Circuit Amortization Friendly Encodingsand Their Application to Statistically Secure Multiparty Computation. In *Advances in Cryptology – ASIACRYPT 2020*, Shiho Moriai and Huaxiong Wang (Eds.). Springer International Publishing, Cham, 213–243. https://doi.org/10.1007/978-3-030-64840-4_8

[15] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and Unconditionally Secure Multiparty Computation. In *Advances in Cryptology - CRYPTO 2007*, Alfred Menezes (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 572–590. https://doi.org/10.1007/978-3-540-74143-5_32

[16] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology – CRYPTO 2012*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 643–662. https://doi.org/10.1007/978-3-642-32009-5_38

[17] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. 2019. New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, CA, USA, 1102–1120. https://doi.org/10.1109/

SP.2019.00078

[18] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. 2022. TurboPack: Honest Majority MPC with Constant Online Communication. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 951–964. https://doi.org/10.1145/3548606.3560633

[19] Daniel Escudero, Cheng Hong, Hongqing Liu, Chaoping Xing, and Chen Yuan. 2023. Degree-D Reverse Multiplication-Friendly Embeddings: Constructions and Applications. In *Advances in Cryptology – ASIACRYPT 2023: 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4–8, 2023, Proceedings, Part I*. Springer-Verlag, Berlin, Heidelberg, 106–138. https://doi.org/10.1007/978-981-99-8721-4_4

[20] Daniel Escudero and Eduardo Soria-Vazquez. 2021. Efficient Information-Theoretic Multi-party Computation over Non-commutative Rings. In *Advances in Cryptology – CRYPTO 2021*, Tal Malkin and Chris Peikert (Eds.). Springer International Publishing, Cham, 335–364. https://doi.org/10.1007/978-3-030-84245-1_12

[21] Matthew K. Franklin and Moti Yung. 1992. Communication Complexity of Secure Computation (Extended Abstract). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of Computing (STOC'92)*, Rao Kosaraju, Mike Fellows, and Avi Wigderson (Eds.). Association for Computing Machinery, New York, NY, United States, 699–710. https://doi.org/10.1145/129712.129780

[22] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. 2014. Circuits resilient to additive attacks with applications to secure computation. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing (STOC'14)*, David Shmoys (Ed.). Association for Computing Machinery, New York, NY, United States, 495–504. https://doi.org/10.1145/2591796.2591861

[23] Vipul Goyal and Yifan Song. 2020. Malicious security comes free in honest-majority mpc. Cryptology ePrint Archive, Report 2020/134. https://eprint.iacr.org/2020/134

[24] Zhenkai Hu, Kang Yang, and Yu Yu. 2024. Unconditionally secure MPC for Boolean circuits with constant online communication. Cryptology ePrint Archive, Paper 2024/015. https://eprint.iacr.org/2024/015

[25] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. 2020. Overdrive2k: Efficient Secure MPC over $\mathbb{Z}_{2^k}$ from Somewhat Homomorphic Encryption. In *Topics in Cryptology – CT-RSA 2020*, Stanislaw Jarecki (Ed.). Springer International Publishing, Cham, 254–283. https://doi.org/10.1007/978-3-030-40186-3_12

[26] Adi Shamir. 1979. How to Share a Secret. *Association for Computing Machinery* 22, 11 (Nov. 1979), 612–613. https://doi.org/10.1145/359168.359176

[27] Zhexian Wan. 2003. *Lectures on finite fields and Galois rings.* World Scientific Publishing Company, 5 Toh Tuck Link, Singapore, 596224. https://doi.org/10.1142/8250

A TOWARDS MALICIOUS SECURITY

In this section, we discuss the obstructions on the way to malicious security and how to remove them. We assume $p^d > 2^{64}$. The asymptotically good RMFEs ensure that this assumption does not harm the efficiency.

First of all, $R = GR(p^k, d)$ is local ring with a unique maximal ideal (p). The ratio of zero divisors is $\frac{1}{p^d}$. This ratio implies that there is a difference between R and fields. But if we assume p^k is large, then R will behave like a field.

The main trick in Turbopack [18] is maintaining degree-t Shamir sharings of all true values, so that modifying the packed sharing will be detected.

The protocol $\Pi_{Consistency}$ from Turbopack should be analyzed over R in our setting. For a polynomail $f \in R[X]_{\leq m}$, f might have more than m roots. However, $f(X)$ has $\leq m$ roots in an exceptional set of R. Denote

$$E = \{a_{d-1}\overline{Y}^{d-1} + \cdots + a_0 \in R | 0 \leq a_i < p, 0 \leq i < d\},$$

which is a maximal exceptional set of R. Then we build $\Pi_{Consistency}$ to check the validity of degree-$(K-1)$ packed Shamir sharings as shown in Figure 13. The functionality $\mathcal{F}_{Coin}$ generates a random element in a designated ring. It can be instantiated as in [23]. Note that the set E can be encoded the same as $\mathbb{F}_{p^d}$. So $\mathcal{F}_{Coin}$ can generate a random element in E. Step (2) of $\Pi_{Consistency}$ can be replaced by generating a random element $r \in R$. The restriction to E makes it easier to analyze the security. The assumption $p^d > 2^{64}$ guarantees security if m is not too large.

$\Pi_{Consistency}$

(1) Let $\{[\boldsymbol{w}_i]_{K-1}\}_{i=1}^m$ denote all degree-$(K-1)$ packed Shamir sharings distributed by P_1.
(2) All parties invoke $\mathcal{F}_{Coin}$ to generate a random element $r \in R$.
(3) All parties locally compute $[\boldsymbol{w}]_{K-1} = \sum_{i=1}^m r^{i-1}[\boldsymbol{w}_i]_{K-1}$.
(4) Each party P_i broadcasts its share of $[\boldsymbol{w}]_{K-1}$. Then each party P_j checks whether the shares of $[\boldsymbol{w}]_{K-1}$ lie on a degree-$(K-1)$ polynomial. If true, P_j accepts the check. Otherwise, P_j aborts.

Figure 13: Consistency Protocol

The protocol *Compress* from [23] implements $\mathcal{F}_{InnerVerify}$ in [18]. Similar analysis shows that *Compress* also works in our setting.

Then we can achieve malicious security using the techniques from Turbopack.

Received 12 March 2024; revised 8 April 2024; accepted 9 April 2024