MHz2k: MPC from HE over $\mathbb{Z}_{2^k}$ with New Packing, Simpler Reshare, and Better ZKP

Jung Hee Cheon[1,3], Dongwoo Kim[(✉)2⋆], and Keewoo Lee[(✉)1]

[1] Seoul National University, Seoul, Republic of Korea
{jhcheon, activecondor}@snu.ac.kr

[2] Western Digital Research, Milpitas, USA
Dongwoo.Kim@wdc.com

[3] Crypto Lab Inc., Seoul, Republic of Korea

Abstract. We propose a multi-party computation (MPC) protocol over $\mathbb{Z}_{2^k}$ secure against actively corrupted majority from somewhat homomorphic encryption. The main technical contributions are: (i) a new efficient packing method for $\mathbb{Z}_{2^k}$-messages in lattice-based somewhat homomorphic encryption schemes, (ii) a simpler reshare protocol for level-dependent packings, (iii) a more efficient zero-knowledge proof of plaintext knowledge on cyclotomic rings $\mathbb{Z}[X]/\Phi_M(X)$ with M being a prime. Integrating them, our protocol shows from 2.2x upto 4.8x improvements in amortized communication costs compared to the previous best results. Our techniques not only improve the efficiency of MPC over $\mathbb{Z}_{2^k}$ considerably, but also provide a toolkit that can be leveraged when designing other cryptographic primitives over $\mathbb{Z}_{2^k}$.

Keywords: Multi-party computation · Dishonest majority · Homomorphic encryption · Packing method · Zero-knowledge proof · $\mathbb{Z}_{2^k}$

1 Introduction

Secure Multi-Party Computation (MPC) aims to jointly compute a function f on input $(x_1, \cdots, x_n)$ each held by n parties $(P_1, \cdots, P_n)$, without revealing any information other than the desired output to each other. Through steady development from the feasibility results in 1980s (e.g., [GBOW88]), MPC research is now at the stage of improving practicality and developing applications to diverse use-cases: auction [BCD+09], secure statistical analysis [BJSV15], privacy-preserving machine learning [DEF+19], etc.

Among various settings of MPC, the most important setting in practice is the actively corrupted dishonest majority case: corrupted majority is the only meaningful goal in two-party computation (2PC), and modeling the security threat as passive (honest-but-curious) adversaries is often unsatisfactory in real-life applications. At the same time, however, it is notoriously difficult to handle actively corrupted majority efficiently. It is a well-known fact that lightweight

⋆ Work done while at Seoul National University.

information-theoretically secure primitives are not sufficient in this setting and we need rather heavier primitives [CK89].

A seminal work BeDOZa [BDOZ11] observed that one can push the use of heavy public key machinery into a preprocessing phase, without knowing input values and functions to compute. Meanwhile in an online phase, one can securely compute a function using only lightweight primitives. This paradigm, so-called *preprocessing model*, spotlighted the possibility of designing an efficient MPC protocol even in actively corrupted dishonest majority setting. From then, there have been active and steady research on improving efficiency of MPC protocol in this setting: [DPSZ12, DKL$^+$13, KOS16, KPR18, BCS19].

All previously mentioned works consider MPC only over finite fields where arithmetic message authentication code (MAC), the main ingredients of the protocols, is easily defined. Recently, SPD$\mathbb{Z}_{2^k}$ [CDE$^+$18] initiated a study of efficient MPC over $\mathbb{Z}_{2^k}$ in actively corrupted dishonest majority setting by introducing an arithmetic MAC for $\mathbb{Z}_{2^k}$-messages. This is to leverage the fact that integer arithmetic on modern CPUs is done modulo 2^k, e.g. $k = 32, 64, 128$; using MPC over $\mathbb{Z}_{2^k}$, one can naturally deal with such arithmetic. Also, there is no need to emulate modulo prime P operations on CPUs, simplifying the online phase implementation. The authors of SPD$\mathbb{Z}_{2^k}$ claimed that these advantages are much beneficial than the loss from the modified MAC for $\mathbb{Z}_{2^k}$. The claim was convinced by the recent implementation and experimental results [DEF$^+$19].

In regard to the cost of the preprocessing phase, however, there still remains a substantial gap between the finite field case and the $\mathbb{Z}_{2^k}$ case. Particularly, the authors of SPD$\mathbb{Z}_{2^k}$, which is based on oblivious transfer (OT), left an open problem to design an efficient preprocessing phase for MPC over $\mathbb{Z}_{2^k}$ from lattice-based homomorphic encryption (HE). The motivation here is that the HE-based approach has proved the best performance in the finite field case.

The main difficulty is that the conventional message packing method using the isomorphism of cyclotomic ring $\mathbb{Z}_t[X]/\Phi_M(X) \cong \mathbb{Z}_t^{\varphi(M)}$ does not work when t is not prime, especially when $t = 2^k$. In fact, cyclotomic polynomials $\Phi_M(X)$ never fully split in $\mathbb{Z}_{2^k}[X]$ (see e.g. [CL21]). This makes it hard to fully leverage the batching technique of HE and causes inefficiency compared to the finite field case. Followup works, Overdrive2k [OSV20] and Mon$\mathbb{Z}_{2^k}$a [CDRFG20], proposed more efficient preprocessing phases for MPC over $\mathbb{Z}_{2^k}$, yet they do not give a satisfactory solution to this problem.

1.1 Our Contribution

MHz2k — MPC from HE over $\mathbb{Z}_{2^k}$. We propose MHz2k, an MPC over $\mathbb{Z}_{2^k}$ from Somewhat HE (SHE) in actively corrupted dishonest majority setting. It is based on our new solution to the aforementioned problem (of developing high-parallelism in SHE with $\mathbb{Z}_{2^k}$-messages) and non-trivial adaptations of techniques used in the finite field case to the $\mathbb{Z}_{2^k}$ case.

Note that the core of an SHE-based MPC preprocessing phase is the triple (or *authenticated* Beaver's triple [Bea91]) generation protocol which consists of the following building blocks (see Section 2.5):

- a *packing* method for SHE which enables parallelism of the protocol and enhances amortized performance;
- the *reshare* protocol which re-encrypts a *level-0* ciphertext to a *fresh* ciphertext allowing two-level SHE to be sufficient for the generation of authenticated triples;
- and *ZKPoPK* (zero-knowledge proof of plaintext knowledge) which guarantees that ciphertexts are validly generated from a plaintext and restricts adversaries from submitting maliciously generated ciphertexts.

We present improvements on all of these building blocks for $\mathbb{Z}_{2^k}$-messages and integrate them into our new preprocessing phase, which is compatible with the online phase of SPD$\mathbb{Z}_{2^k}$.

New Packing Method for $\mathbb{Z}_{2^k}$-messages. We suggest a new efficient $\mathbb{Z}_{2^k}$-message packing method for SHE which can be applied to a preprocessing phase over $\mathbb{Z}_{2^k}$ (Section 3). Under the plaintext ring of degree N, our packing method achieves near $N/2$-fold parallelism while providing depth-1 homomorphic correspondence which is enough for the preprocessing phase. Previously, the best solution over $\mathbb{Z}_{2^k}$ of Overdrive2k [OSV20] only achieved roughly $N/5$-fold parallelism. Thus, our packing method directly offers 2.5x improvement in the overall (amortized) performance of the preprocessing phase.

When constructing our packing method, to remedy the impossibility[iv] of interpolation on $\mathbb{Z}_{2^k}$, we devise a *tweaked* interpolation, in which we lift the target points of $\mathbb{Z}_{2^k}$ to a larger ring $\mathbb{Z}_{2^{k+\delta}}$ (Lemma 1).

Reshare Protocol for Level-dependent Packings. A seeming problem is that it is difficult to design a *level-consistent* packing method for $\mathbb{Z}_{2^k}$-messages with high parallelism (see [CL21]), while the previous reshare protocol for messages in finite fields (with *level-consistent* packing) should be modified to be utilized in this setting. To this end, in the reshare protocol of Overdrive2k [OSV20], an extra masking ciphertext with ZKPoPK, which is the most costly part, is provided. We propose a new reshare protocol for *level-dependent* packings, which resolves this problem and closes the gap between the field case and the $\mathbb{Z}_{2^k}$ case (Section 4). Concretely, in our triple generation, the total number of ZKPoPK is *five* as using the original reshare, whereas Overdrive2k requires *seven*. From this aspect, we gain an additional 1.4x efficiency improvement in total communication cost.

TopGear2k — Better ZKPoPKs over $\mathbb{Z}[X]/\Phi_p(X)$. When the messages are in $\mathbb{Z}_{2^k}$, using power-of-two cyclotomic rings $\mathbb{Z}[X]/\Phi_{2^m}(X)$ introduces a huge inefficiency in packing, since $\Phi_{2^m}(X)$ has only one irreducible factor in $\mathbb{Z}_{2^k}[X]$. Thus, it is common to use *odd* cyclotomic rings for $\mathbb{Z}_{2^k}$-messages. In this case,

[iv] For example, over $\mathbb{Z}_{2^k}$, a polynomial $f(X)$ of degree 2 such that $f(0) = f(1) = 0$ and $f(2) = 1$ does not exist.

however, we cannot leverage known efficient ZKPoPKs over the ciphertexts regarding $\mathbb{Z}[X]/\Phi_{2^m}(X)$, such as TopGear [BCS19][v].

To this end, we develop an efficient ZKPoPK over $\mathbb{Z}[X]/\Phi_p(X)$ where p is a prime (Section 5). This new protocol named TopGear2k is an adaptation of TopGear to the $\mathbb{Z}_{2^k}$ case. The essence of TopGear2k is that the core properties of power-of-two cyclotomic rings, which was observed in [BCK$^+$14], hold similarly also in prime cyclotomic rings (Lemma 4). This fact not only improves the amortized communication cost, latency, and memory consumption of our ZKPoPK, but can also has ramifications on works derived from [BCK$^+$14].

ZKP of Message Knowledge. For the MPC preprocessing for messages from a finite field $\mathbb{Z}_P$, where SHE has the plaintext space $\mathbb{Z}_P[X]/\Phi_{2^m}(X)$ *isomorphic* to the message space $\mathbb{Z}_P^{\varphi(2^m)}$, ZKPoPK is sufficient. In the $\mathbb{Z}_{2^k}$ case, however, packing methods are not *surjective* (see [CL21]). In other words, there exist invalidly encoded plaintexts which do not correspond to any messages. Thus, we must also make sure that malicious adversaries had not deviated from the packing method when generating the ciphertext. To this end, we propose a Zero-Knowledge Proof of *Message* Knowledge (ZKPoMK) which guarantees that the given ciphertext is generated with a plaintext which is a *valid encoding* with respect to our new packing method (Section 6).

Performance. MHz2k achieves the best efficiency in amortized communication cost among all state-of-the-art MPC protocols over $\mathbb{Z}_{2^k}$ in the actively corrupted dishonest majority setting. Concretely, in our preprocessing phase, the amortized communication costs for triple generation[vi] (in kbit) over $\mathbb{Z}_{2^{32}}$ and $\mathbb{Z}_{2^{64}}$, respectively, are 27.4 and 43.3 which outperforms the current best results, 59.1 of Mon$\mathbb{Z}_{2^k}$a [CDRFG20] and 153.3 of Overdrive2k [OSV20], respectively showing 2.2x and 3.5x improvements. Comparing our protocol with TopGear2k optimization (MHz2k-TG2k) and without it (MHz2k-Plain), our ZKPoPK together with our ZKPoMK improves memory requirement over 5.6x.

1.2 Roadmap

In Section 2, we define notations and recall some known ideas which we frequently refer to in our paper. In Section 3, 4, 5, and 6, we present our results on packing, reshare, ZKPoPK, and ZKPoMK, respectively. In Section 7, we present a performance analysis of our protocols: MHz2k-plain (which exploits our new packing and reshare protocol) and MHz2k-TG2k (which additionally exploits our ZKPoPK and ZKPoMK).

Fig. 1 describes dependencies of this paper. Arrows denote dependencies, and the dashed arrow denote rather weak dependency. Section 4 refers to Section 3

[v] It is the recent refinement with the most efficient ZKPoPK among the line of works [DPSZ12, KPR18, BCS19] exploiting (S)HE to MPC over a finite field.

[vi] We assume a two-party case, and similar improvements occur in multi-party cases.

$$\begin{tikzcd}
 & \text{Section 3 (Packing)} \arrow[r, dashed] \arrow[dr] \arrow[dd] & \text{Section 4 (Reshare)} \arrow[d] \\
 & & \text{MHz2k-Plain} \arrow[d] \\
\text{Section 5 (ZKPoPK)} \arrow[r] & \text{Section 6 (ZKPoMK)} \arrow[r] & \text{MHz2k-TG2k}
\end{tikzcd}$$

Fig. 1. Dependencies of This Paper

only in Section 4.2 to note that our new packing method is compatible with the new reshare process. Section 3, 4, and 5 can be read (except Section 4.2) and employed independently.

1.3 Related Work

We present the previous works achieving the same goal as ours: MPC over the ring $\mathbb{Z}_{2^k}$ secure against actively corrupted dishonest majority. All of the works (including ours) share the same online phase proposed by SPD$\mathbb{Z}_{2^k}$ [CDE+18], whereas the preprocessing phases are all different.

SPD$\mathbb{Z}_{2^k}$ [CDE+18] is the first MPC protocol over $\mathbb{Z}_{2^k}$ secure against actively corrupted dishonest majority. Their main technical contribution is the online phase with an efficient MAC for $\mathbb{Z}_{2^k}$ (see Section 2.5). Their preprocessing phase resembles that of MASCOT [KOS16] which is based on oblivious transfers. The authors of SPD$\mathbb{Z}_{2^k}$ left an open problem to design an efficient HE-based protocol over $\mathbb{Z}_{2^k}$ since, in the finite field setting, it is the approach with the best performance.

Overdrive2k [OSV20] is an HE-based MPC protocol over $\mathbb{Z}_{2^k}$, partially solving the open problem given in SPD$\mathbb{Z}_{2^k}$. The protocol mainly follows the approach of SPDZ [DPSZ12] with the BGV SHE scheme [BGV14]. Their main idea is a new HE-packing method for $\mathbb{Z}_{2^k}$ messages supporting one homomorphic multiplication only (See Section 2.4). Using their method, however, packing density for their parameters stay below 0.25. Moreover, to remedy their *level-dependent* packing, they provide extra masking ciphertexts with ZKPoPKs, substantially increasing the cost of the preprocessing phase.

Mon$\mathbb{Z}_{2^k}$a [CDRFG20] is a 2PC protocol over $\mathbb{Z}_{2^k}$ which mainly follows the linear-HE-based approach of BDOZ [BDOZ11] and Overdrive [KPR18], but with a different HE scheme by Joye-Libert [JL13]. Note that the Joye-Libert scheme does not provide packing for batched computations, whereas major and fastest approaches of MPC over finite fields leverage packing. Also note that Mon$\mathbb{Z}_{2^k}$a provides only 2PC and does not provide general MPC.

2 Preliminaries

2.1 Notations

The ring $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$ is identified with the set of integers in $(-q/2, q/2]$. We denote the set $\{1, 2, \cdots, d\}$ by $[d]$ and the set $\{0, 1, \cdots, d\}$ by $[0, d]$. The additive share of i-th party is denoted as $[\cdot]_i$. For a positive integer a, let $\nu_2(a)$ be the exponent of the largest power of two that divides a. All logarithms $\log(\cdot)$ are of base 2. On homomorphic encryption, ciphertext additions, subtractions, and multiplications are denoted as $\boxplus$, $\boxminus$, and $\boxtimes$, respectively. We denote the M^{th} cyclotomic polynomial as $\Phi_M(X)$ and reserve N for its degree, i.e., $N = \varphi(M)$ where $\varphi(\cdot)$ denotes Euler's totient function. Each elements of $\mathbb{Z}[X]/f(X)$ is identified with its representative of minimal degree. For an element $a \in \mathbb{Z}[X]/f(X)$, we measure the size of a by $||a||_\infty$, the largest absolute value of its coefficients.

2.2 The BGV Homomorphic Encryption Scheme

Following the approach of SPDZ [DPSZ12], our preprocessing only requires secure computations of multiplicative depth one. Hence, it is enough to initiate the BGV [BGV14] homomorphic encryption scheme supporting only two levels. Here, we only give a brief description of the scheme, focusing on the necessary parts for our proposal.

Two-Level BGV Scheme with Power-of-Two Plaintext Modulus. Let $R := \mathbb{Z}[X]/\Phi_M(X)$. The scheme consists of six algorithms ($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ModSwitch}, \mathsf{Dec}, \mathsf{Add}, \mathsf{Mult}$), has a ring $R_{2^t} := R/2^tR = \mathbb{Z}_{2^t}[X]/\Phi_M(X)$ as a plaintext space, and each ciphertext has a level $\ell \in \{0, 1\}$.

For a given security parameter λ, the public parameter pp_λ fixes a cyclotomic polynomial $\Phi_M(X)$ with a sufficiently large degree; ciphertext moduli $q_1 = p_1 \cdot p_0$ and $q_0 = p_0$ for some prime p_0, p_1. Now, the algorithms are as follows: We only give a brief description which is necessary for our proposal.

- $\mathsf{KeyGen}(\mathsf{pp}_\lambda)$: Given a public parameter pp_λ, outputs a secret key $\mathfrak{sk} \in R$, a public key $\mathfrak{pk} = (a, b) \in R_{q_1}^2$, and relinearization data [BGV14] for the ciphertext multiplication.
- $\mathsf{Enc}(m, r; \mathfrak{pk})$: For given plaintext $m \in R_{2^t}$, samples randomnesses $r = (e_0, e_1, v) \in R^3$ as $e_0, e_1 \leftarrow \mathsf{DG}(3.16^2)$ and $v \leftarrow \mathsf{ZO}(0.5)$,[vii] then sets,
$$c_0 = b \cdot v + 2^t \cdot e_0 + m \pmod{q_1}, \quad c_1 = a \cdot v + 2^t \cdot e_1 \pmod{q_1}.$$
Then, outputs a level-one ciphertext $\mathfrak{ct}^{(1)} = (c_0, c_1) \in R_{q_1}^2$.
- $\mathsf{ModSwitch}(\mathfrak{ct}^{(1)} = (c_0, c_1))$: Given a level-one ciphertext $\mathfrak{ct}^{(1)}$, outputs a level-zero ciphertext $\mathfrak{ct}^{(0)} = (c_0', c_1') \in R_{q_0}^2$ having the *same* message as $\mathfrak{ct}^{(1)}$. We call this a modulus-switching operation.

[vii] $\mathsf{DG}(\sigma^2)$ samples each coefficient from discrete Gaussian distribution, $\mathsf{ZO}(\rho)$ samples from $\{-1, 0, 1\}$ with probability $\rho/2$ for each of -1 and 1, probability $1 - \rho$ for 0.

- $\mathsf{Dec}(\mathfrak{ct}^{(\ell)} = (c_0, c_1); \mathfrak{sk})$: If $\ell \neq 0$, it gets a level-zero ciphertext $\mathfrak{ct}^{(0)} = (c_0', c_1')$ via $\mathsf{ModSwitch}$. Then, it decrypts as
$$(c_0' - \mathfrak{sk} \cdot c_1' \pmod{q_0}) \pmod{2^t},$$
and outputs an element of R_{2^t}.
- Homomorphic Operations: Ciphertexts at the same level can be added ($\boxplus$) or multiplied ($\boxtimes$) with each other, resulting in a ciphertext encrypting the sum or the product of the plaintexts in R_{2^t}. Only level-*one* ciphertexts can be multiplied (with each other) to result in a ciphertext of level-*zero*.

2.3 Cyclotomic Rings and CRT Isomorphism in $\mathbb{Z}_{2^T}[X]$

For an odd M, the cyclotomic polynomial $\Phi_M(X)$ of degree N is factorized as $\prod_{i=1}^{r} f_i(X)$ in $\mathbb{Z}_2[X]$ where each irreducible $f_i(X)$ has the same degree $d = \mathrm{ord}_M(2)$, the order of 2 modulo M. Hence, $N = r \cdot d$ holds. The factorization induces the following ring isomorphism by the CRT, for any power of two 2^T:
$$\mathbb{Z}_{2^T}[X]/\Phi_M(X) \cong (\mathbb{Z}_{2^T}[X]/F_1(X)) \times \cdots \times (\mathbb{Z}_{2^T}[X]/F_r(X)), \qquad (1)$$
where each $F_i(X) \in \mathbb{Z}_{2^T}[X]$ is the Hensel lifting of $f_i(X)$ with degree d. Each $\mathbb{Z}_{2^T}[X]/F_i(X)$ is often referred to as a *slot* of $\mathbb{Z}_{2^T}[X]/\Phi_M(X)$. In this paper, we frequently refer to the isomorphism Eq.(1) and the notation $\varphi(M) = N = r \cdot d$.

2.4 Packing Methods for SHE Schemes

Message, Plaintext, and Packing. This paper carefully distinguishes between the use of the terms *message* and *plaintext*. Messages are those we want to compute with using HE. On the other hand, plaintexts are defined by the HE scheme we are using. In this paper, messages are in $\mathbb{Z}_t$ and plaintexts are in $\mathbb{Z}_t[X]/\Phi_M(X)$, for possibly different t's.

Packing is the process of encoding multiple messages into a plaintext while satisfying (somewhat) homomorphic correspondence. Then, when performing homomorphic computations on a ciphertext packed with multiple messages, one can have the effect of *batching*. The idea of packing [SV14] is very useful in most cases, since plaintext space $\mathbb{Z}_t[X]/\Phi_M(X)$ of practical lattice-based HE schemes is usually not the space we want to compute in.

Basic Packing Methods. In lattice-based SHE schemes, including [BGV14], it is common to choose the plaintext modulus as a prime P such that $\Phi_M(X)$ fully splits in $\mathbb{Z}_P[X]$. Then, we can pack N messages of $\mathbb{Z}_P$ into one plaintext in $\mathbb{Z}_P[X]/\Phi_M(X)$ by the CRT ring isomorphism $\mathbb{Z}_P[X]/\Phi_M(X) \cong \mathbb{Z}_P^N$.

Above method, however, does not work for the case of $\mathbb{Z}_{2^k}$-messages, since $\Phi_M(X)$ never fully splits in $\mathbb{Z}_{2^k}[X]$. A common way [GHS12a, HS15] to detour this problem is to identify each $\mathbb{Z}_{2^k}$-message with each constant term of $\mathbb{Z}_{2^k}[X]/F_i(X)$ in Eq.(1). It provides fully homomorphic correspondence between r messages of $\mathbb{Z}_{2^k}$ and one element of $\mathbb{Z}_{2^k}[X]/\Phi_M(X)$, but with extremely low packing density $1/d$, following the notations of Section 2.3.

Overdrive2k Packing. Overdrive2k [OSV20] observed that what we actually need for MPC protocol is a packing method which provides *somewhat* homomorphic correspondence supporting one multiplication (See Section 2.5). For a given degree $d = \deg F_1(X)$, they consider a subset $A = \{a_i\}_{i=1}^{w}$ of $[0, d-1]$ such that $2a_i \neq a_{j_1} + a_{j_2}$ for all $(i,i) \neq (j_1, j_2)$ and $a_i + a_j < d$ for all i, j. They pack w messages in $\mathbb{Z}_{2^k}$ as the a_i-th coefficients ($a_i \in A$) of a polynomial in $\mathbb{Z}_{2^k}[X]/F_1(X)$, putting zeroes in the other coefficients. Repeating this r times for each slot in Eq.(1), we can pack $r \cdot w$ messages into one plaintext achieving the packing density of w/d. Since the set A is carefully chosen, if we multiply two packed plaintexts, the $(2a_i)$-th coefficient of the result equals to the product of a_i-th coefficients of the original plaintexts, providing depth-1 homomorphic correspondence. Note that the Overdrive2k packing is *level-dependent*: messages are at a_i-th coefficients for level one plaintexts, and $(2a_i)$-th coefficients for level zero plaintexts. The authors of Overdrive2k note that the packing density of their method with an optimal subset A seems to follow the trend of $d^{0.6}/d$, approximately.

2.5 Preprocessing Phase — Generation of Authenticated Triples

Since our MPC protocol follows the online phase of SPD$\mathbb{Z}_{2^k}$ [CDE+18], the goal of our preprocessing phases is to generate *authenticated triples* with respect to SPD$\mathbb{Z}_{2^k}$-MAC. That is, n parties together securely generate secret shares $[a]_i, [b]_i, [c]_i$ and $[\alpha a]_i, [\alpha b]_i, [\alpha c]_i$ in $\mathbb{Z}_{2^{\tilde{k}}}$ such that $\sum_i [a]_i = a \pmod{2^k}$, $\sum_i [\alpha a]_i = \alpha a \pmod{2^{\tilde{k}}}$, and similar for the others, satisfying $c = ab \pmod{2^k}$. Here, $\tilde{k} := k + s$ with s as a security parameter[viii], and $\alpha \in \mathbb{Z}_{2^{\tilde{k}}}$ is a single global MAC key of which share $[\alpha]_i \in \mathbb{Z}_{2^s}$ is given to the i-th party. Then, in the online phase, the parties can securely compute any arithmetic circuit via Beaver's trick [Bea91, CDE+18] with these authenticated triples.

Overview of Triple Generation. We give an overview of our preprocessing phase, focusing on the triple generation protocol, which follows the standard methods of SPDZ [DPSZ12] (and Overdrive2k [OSV20]) exploiting *two-level* SHE and zero-knowledge proofs (ZKP) on it. We remark that message packing of SHE enable the parties to generate multiple authenticated triples (represented by vectors) in one execution of the triple generation protocol, significantly reducing the amortized costs.

First, each party P_i generates and broadcasts ciphertexts $\mathfrak{ct}_{\boldsymbol{a}_i}$ and $\mathfrak{ct}_{\boldsymbol{b}_i}$ each encrypting the *vectors* $[\boldsymbol{a}]_i$ and $[\boldsymbol{b}]_i$ of random shares from $\mathbb{Z}_{2^{\tilde{k}}}$; we omit the superscript$^{(1)}$ for level-one ciphertexts. Then, all parties run ZKPs (ZKPoPK and ZKPoMK in Section 5 and 6) on $\mathfrak{ct}_{\boldsymbol{a}} = \sum_i \mathfrak{ct}_{\boldsymbol{a}_i}$ and $\mathfrak{ct}_{\boldsymbol{b}} = \sum_i \mathfrak{ct}_{\boldsymbol{b}_i}$ to guarantee that each ciphertext is generated correctly. Next, all parties compute a ciphertext $\mathfrak{ct}_{\boldsymbol{c}}^{(0)} := \mathfrak{ct}_{\boldsymbol{a}} \boxtimes \mathfrak{ct}_{\boldsymbol{b}}$ whose underlying message is the Hadamard product $\boldsymbol{c} = \boldsymbol{a} \odot$

[viii] SPD$\mathbb{Z}_{2^k}$-MAC provides $\mathsf{sec} = s - \log(s+1)$-bit statistical security ([CDE+18, Theorem 1]).

$\boldsymbol{b}$. Similarly, given ciphertexts $\mathfrak{ct}_{\alpha_i}$, all parties can also compute $\mathfrak{ct}^{(0)}_{\alpha\boldsymbol{a}}$ and $\mathfrak{ct}^{(0)}_{\alpha\boldsymbol{b}}$ with homomorphic operations on the ciphertexts. The parties, however, cannot directly compute $\mathfrak{ct}_{\alpha\boldsymbol{c}}$ from ciphertext multiplication between $\mathfrak{ct}^{(0)}_{\boldsymbol{c}}$ and $\mathfrak{ct}_{\alpha}$ since the former is of level-zero.

Thus, the parties perform so-called *reshare* protocol [DPSZ12] which, given $\mathfrak{ct}^{(0)}_{\boldsymbol{c}}$ as the input, outputs a *level-one* ciphertext $\mathfrak{ct}_{\boldsymbol{c}}$ having the same message as the input and/or the random shares $[\boldsymbol{c}]_i$ of the message to each party. Roughly, it proceeds by decrypting the masked input $\mathsf{ModSwitch}(\mathfrak{ct}_{\boldsymbol{f}}) \boxplus \mathfrak{ct}^{(0)}_{\boldsymbol{c}}$ to get a (masked) message $\boldsymbol{f}+\boldsymbol{c}$, then subtracting the mask $\mathfrak{ct}_{\boldsymbol{f}}$ from the fresh encryption $\mathfrak{ct}_{\boldsymbol{f}+\boldsymbol{c}}$ of the message, resulting in $\mathfrak{ct}_{\boldsymbol{c}} = \mathfrak{ct}_{\boldsymbol{f}+\boldsymbol{c}} \boxminus \mathfrak{ct}_{\boldsymbol{f}}$. Then, parties can compute $\mathfrak{ct}^{(0)}_{\alpha\boldsymbol{c}} := \mathfrak{ct}_{\boldsymbol{c}} \boxtimes \mathfrak{ct}_{\alpha}$. Here, ZKPs for the masking ciphertext $\mathfrak{ct}_{\boldsymbol{f}}$ is also required.

Finally, parties jointly perform *distributed decryption* on the ciphertexts $\mathfrak{ct}_{\alpha\boldsymbol{a}}$, $\mathfrak{ct}_{\alpha\boldsymbol{b}}$, and $\mathfrak{ct}_{\alpha\boldsymbol{c}}$ to get random shares of the underlying messages: $[\alpha\boldsymbol{a}]_i$, $[\alpha\boldsymbol{b}]_i$, and $[\alpha\boldsymbol{c}]_i$. The parties already have the other components of the triple ($[\boldsymbol{a}]_i$, $[\boldsymbol{b}]_i$, and $[\boldsymbol{c}]_i$), so the authenticated triple is generated.

3 New Packing Method for $\mathbb{Z}_{2^k}$-Messages

In this section, we present a new and efficient $\mathbb{Z}_{2^k}$-message packing method for contemporary SHE schemes, e.g. BGV [BGV14]. Since the conventional plaintext packing method of using the isomorphism $\mathbb{Z}_t[X]/\Phi_M(X) \cong \mathbb{Z}_t^{\varphi(M)}$ does not work when $t = 2^k$, an alternative method is required to provide high parallelism.

To tackle this problem, unlike previous approaches which packed messages in coefficients of a polynomial (Section 2.4), we pack messages in evaluation points of a polynomial. Here, we detour the impossibility[ix] of interpolation on $\mathbb{Z}_{2^k}$ by introducing a *tweaked* interpolation on $\mathbb{Z}_{2^k}$.

3.1 Tweaked Interpolation

The crux of our packing method is the following lemma: we can perform interpolation on $\mathbb{Z}_{2^k}$ if we lift the target points of $\mathbb{Z}_{2^k}$ upto a larger ring $\mathbb{Z}_{2^{k+\delta}}$, multiplying an appropriate power of two to eliminate the effect of non-invertible elements.

Lemma 1 (Tweaked Interpolation on $\mathbb{Z}_{2^k}$). *Let $\mu_0, \mu_1, \ldots, \mu_n$ be elements in $\mathbb{Z}_{2^k}$. Assume that an integer δ is not smaller than $\nu_2(n!)$, the multiplicity of 2 in the factorization of $n!$. Then, there exists a polynomial $\Lambda(X) \in \mathbb{Z}_{2^{k+\delta}}[X]$ of degree at most n such that*

$$\Lambda(i) = \mu_i \cdot 2^\delta \qquad \forall i \in [0, n].$$

Proof. Recall that, for $i \in [0, n]$, an i-th Lagrange polynomial on $[0, n]$ is defined as $\lambda_i(X) := \prod_{j \in [0,n]\setminus\{i\}} \frac{X-j}{i-j} \in \mathbb{Q}[X]$. Lagrange polynomial satisfies

$$\lambda_i(X) = \begin{cases} 0 & \text{if } X \in [0, n] \text{ and } X \neq i, \\ 1 & \text{if } X = i. \end{cases}$$

Note that $2^\delta \lambda_i(X)$ has no multiples of 2 in denominators of its coefficients since $\delta \geq \nu_2(n!)$. Then, we can identify $2^\delta \lambda_i(X)$ as a polynomial over $\mathbb{Z}_{2^{k+\delta}}$ of degree at most n, since the denominator of each coefficient is now invertible in $\mathbb{Z}_{2^{k+\delta}}$. Let $\tilde{\lambda}_i(X) \in \mathbb{Z}_{2^{k+\delta}}[X]$ denote the polynomial. Then,

$$\tilde{\lambda}_i(X) = \begin{cases} 0 & \text{if } X \in [0, n] \text{ and } X \neq i, \\ 2^\delta & \text{if } X = i. \end{cases}$$

Now, $\Lambda(X) := \sum_{i=0}^{n} \mu_i \cdot \tilde{\lambda}_i(X) \in \mathbb{Z}_{2^{k+\delta}}[X]$ satisfies the claimed property. □

[ix] For example, over $\mathbb{Z}_{2^k}$, a polynomial $f(X)$ of degree 2 such that $f(0) = f(1) = 0$ and $f(2) = 1$ does not exist.

3.2 New Packing Method from Tweaked Interpolation

Our tweaked interpolation on $\mathbb{Z}_{2^k}$ gives an efficient $\mathbb{Z}_{2^k}$-message packing into $\mathbb{Z}_{2^{k+2\delta}}[X]/\Phi_M(X)$, while providing *depth-1 homomorphic correspondence.* Notice the extra δ added to preserve packed messages: after multiplying two polynomials constructed from tweaked interpolation, the resulting polynomial carries a factor of $2^{2\delta}$. In bird's eye view, our new packing method applies tweaked interpolation on each CRT slots (Eq. (1), Section 2.3), while preventing degree overflow and modulus overflow when multiplying two packed polynomials. Recall the isomorphism Eq. (1) and the notation $\varphi(M) = r \cdot d$ of $\Phi_M(X)$ (Section 2.3).

Theorem 1 (Tweaked Interpolation Packing). *Let $\{\mu_{ij}\}_{i,j}$ be $\mathbb{Z}_{2^k}$-messages for $i \in [r]$ and $j \in [0, \lfloor \frac{d-1}{2} \rfloor]$. For integers δ, t satisfying $\delta \geq \nu_2(\lfloor \frac{d-1}{2} \rfloor !)$ and $t \geq k+\delta$, there exists $L(X) \in \mathbb{Z}_{2^t}[X]/\Phi_M(X)$ satisfying the following properties:*

Let $L_i(X)$ be the projection of $L(X)$ onto the i-th slot $\mathbb{Z}_{2^t}[X]/F_i(X)$. Then, for each i and j,

(i) $\deg(L_i(X)) \leq \lfloor \frac{d-1}{2} \rfloor$,

(ii) $L_i(j) = \mu_{ij} \cdot 2^\delta \mod 2^{k+\delta}$.

We call such $L(X)$ a tweaked interpolation packing *of $\{\mu_{ij}\}$.*

Proof. By Lemma 1, the condition on δ guarantees that there exists $L_i(X) \in \mathbb{Z}_{2^{k+\delta}}[X] \subset \mathbb{Z}_{2^t}[X]$ of degree not greater than $\lfloor \frac{d-1}{2} \rfloor$ such that $L_i(j) = \mu_{ij} \cdot 2^\delta \mod 2^{k+\delta}$ for all $j \in [0, \lfloor \frac{d-1}{2} \rfloor]$. Now, we can define $L(X) \in \mathbb{Z}_{2^t}[X]/\Phi_M(X)$ as the isomorphic image of $(L_1(X), \cdots, L_r(X)) \in \prod_{i=1}^r \mathbb{Z}_{2^t}[X]/F_i(X)$ from the CRT isomorphism; $L(X)$ satisfies the property. □

The next theorem suggests that the tweaked interpolation packing (Theorem 1) homomorphically preserves the messages under (multiplicative) depth-1 arithmetic circuits. This property implies that we can naturally plug our packing method into the two-level BGV scheme (Section 2.2) with a plaintext space $\mathbb{Z}_{2^{k+2\delta}}[X]/\Phi_M(X)$ and exploit it for MPC preprocessing phase.

Theorem 2 (Depth-1 Homomorphic Correspondence[x]). *Let $L(X)$ and $R(X)$ be polynomials in $\mathbb{Z}_{2^{k+2\delta}}[X]/\Phi_M(X)$ which are tweaked interpolation packings (Theorem 1, $t = k+2\delta$) of $\mathbb{Z}_{2^k}$-messages $\{\mu_{ij}^L\}$ and $\{\mu_{ij}^R\}$, respectively. For $\alpha \in \mathbb{Z}_{2^k}$, let $\tilde{\alpha}$ denote an element of $\mathbb{Z}_{2^{k+2\delta}}$ such that $\tilde{\alpha} = \alpha \pmod{2^k}$. Then,*

(a) *$L(X) + R(X)$ is a tweaked interpolation packing of $\{\mu_{ij}^L + \mu_{ij}^R\}$.*

(b) *$\tilde{\alpha} \cdot L(X)$ is a tweaked interpolation packing of $\{\alpha \cdot \mu_{ij}^L\}$.*

[x] Our packing ($\mathbb{Z}_{2^k}^n \hookrightarrow \mathbb{Z}_{2^{k+2\delta}}[X]/F_i(X)$) can be interpreted as an analogue of *reverse multiplication-friendly embeddings* ($\mathbb{F}_q^n \hookrightarrow \mathbb{F}_{q^d}$) [CCXY18]. The *composition* lemma holds similarly in $\mathbb{Z}_{2^k}$ case, since a Galois extension of a Galois ring is again a Galois ring.

(c) *From $LR(X) := L(X) \cdot R(X)$, one can decode homomorphically multiplied $\mathbb{Z}_{2^k}$-messages $\{\mu_{ij}^L \cdot \mu_{ij}^R\}$.*

Proof. Properties (a) and (b) are straightforward from the linearity of projection map and evaluation map, together with the fact that additions and scalar multiplications preserves the degree of polynomial.

To prove (c), let $L_i(X)$, $R_i(X)$, and $LR_i(X)$ respectively be the projection of $L(X)$, $R(X)$, and $LR(X)$ onto the i-th slot $\mathbb{Z}_{2^{k+2\delta}}[X]/F_i(X)$. Then,

$$LR_i(X) = L_i(X) \cdot R_i(X) \qquad \text{in } \mathbb{Z}_{2^{k+2\delta}}[X]/F_i(X).$$

Note that the above equation holds also in $\mathbb{Z}_{2^{k+2\delta}}[X]$: Since the degree of $L_i(X)$ and $R_i(X)$ are at most $\lfloor \frac{d-1}{2} \rfloor$, the sum of their degree is less than the degree d of $F_i(X)$. Therefore,

$$LR_i(j) = L_i(j) \cdot R_i(j) = \mu_{ij}^L \cdot \mu_{ij}^R \cdot 2^{2\delta} \pmod{2^{k+2\delta}},$$

from which one can decode the desired values. □

Remark 1. We call the packing structure of $LR(X)$ in Theorem 2(c) the *level-zero* tweaked interpolation packing, whereas the original packing in Theorem 1 is called *level-one* packing. We omit the level when the packing is of level-one.

3.3 Performance Analysis

Efficiency (Packing Density). As a measure of the efficiency of packing methods, we define *packing density* as the ratio of the total (bit)-size of points packed in a polynomial to the (bit)-size of the polynomial. For example, in the case of finite field $\mathbb{F}$, we can pack N points (of $\mathbb{F}$) to one polynomial (over $\mathbb{F}$) of degree $N-1$ (having N coefficients), which gives the perfect packing density of 1.

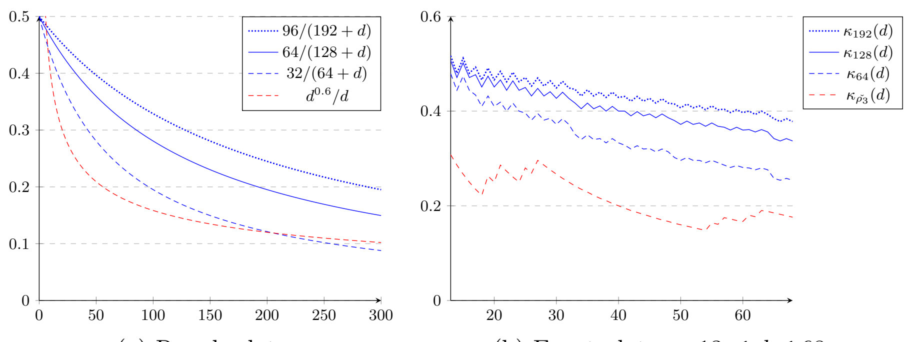
Now, let $\kappa_k(d)$ denote the packing density of tweaked interpolation packing method for $\mathbb{Z}_{2^k}$-messages when the cyclotomic polynomial $\Phi_M(X)$ splits into irreducible factors of degree d. Then,

$$\kappa_k(d) = \frac{k \cdot \lfloor \frac{d+1}{2} \rfloor}{\left(k + 2\nu_2(\lfloor \frac{d-1}{2} \rfloor !)\right) d} \approx \frac{k}{2(k+d)},$$

where the approximation follows from $\nu_2(\lfloor \frac{d-1}{2} \rfloor !) \approx \frac{d}{2}$ and $\lfloor \frac{d+1}{2} \rfloor \approx \frac{d}{2}$.

Remark 2. For a fixed $\mathbb{Z}_{2^k}$, the packing density of our method (Theorem 1) depends only on d: it is better to use $\Phi_M(X)$ with smaller d. When d is sufficiently smaller than k, the packing density of our method approaches $\frac{1}{2}$.

Comparison with Overdrive2k. Let $\kappa_{\tilde{\rho}_3}(d)$ denote the packing density of Overdrive2k packing [OSV20] for given d (Section 2.4). In Fig. 2a, the rough plots of packing densities according to d are presented: the lowest one is the plot of $d^{0.6}/d$ which was mentioned as a rough estimate of $\kappa_{\tilde{\rho}_3}(d)$ in [OSV20]. The graph suggests that our method has higher packing density than theirs when k is not too small compared to d. For practical parameters, this is always the case: in Fig. 2b, the exact plots of packing densities on $13 \le d \le 68$ demonstrates that the density of our method is higher than that of Overdrive2k.



(a) Rough plots (b) Exact plots on $13 \le d \le 68$

Fig. 2. Comparison of packing densities on each method according to d

3.4 Predicates for Valid Packing

In this subsection, we define some predicates $\mathsf{P} : R \to \{\mathsf{true}, \mathsf{false}\}$ over a cyclotomic ring $R = \mathbb{Z}[X]/\Phi_M(X)$, with which we can formally describe the state of a plaintext in regards to our new packing method. We will use these predicates when describing our $\mathsf{Reshare}$ protocol (in Section 4) and our ZKP of Message Knowledge (ZKPoMK) (in Section 6). Readers may skip this subsection and consult it when succeeding sections refer to the definitions.

Definition 1 (Predicates). *The predicates* $\mathsf{Deg}_T^{(D)}, \mathsf{Div}_T^{(D,\Delta)}$, *and* $\mathsf{Pack}_T^{(D,\Delta)}$, *each mapping* R *to* $\{\mathit{true}, \mathit{false}\}$, *are defined as follows:*

For an element $a \in R$, *let* $\tilde{a} \in R_{2^T}$ *be defined by* $\tilde{a} \equiv a \pmod{2^T}$, *and let* $(\tilde{a}_i)_{i=1}^r$ *be the CRT projections (Eq. (1)) of* $\tilde{a}$.

- $\mathsf{Deg}_T^{(D)}(a) = \mathit{true} \iff \deg \tilde{a}_i \le D \quad \forall i \in [r]$
- $\mathsf{Div}_T^{(D,\Delta)}(a) = \mathit{true} \iff 2^\Delta$ *divides* $\tilde{a}_i(j) \quad \forall i \in [r]$ & $j \in [0, D]$
- $\mathsf{Pack}_T^{(D,\Delta)}(a) = \mathit{true} \iff \mathsf{Deg}_T^{(D)}(a) = \mathit{true} \quad \wedge \quad \mathsf{Div}_T^{(D,\Delta)}(a) = \mathit{true}.$

In addition, the predicate $\mathsf{DivCheck}_T^{(D,\Delta)} : R \times \hat{R} \to \{\mathit{true}, \mathit{false}\}$ *is defined as follows, where* $\hat{R} = \mathbb{Z}[X]/\Phi_{\hat{M}}(X)$ *is another cyclotomic ring:*

For $b \in \hat{R}$, *let* $\tilde{b}_{ij} \in \mathbb{Z}_{2^T}$ *be* $\tilde{b}_{ij} \equiv b_{ij} \pmod{2^T}$, *where* b_{ij} *is the* $((i-1)(D+1)+j)$*-th coefficient of* b.[xi]

- $\mathsf{DivCheck}_T^{(D,\Delta)}(a, b) = \mathit{true} \iff \tilde{a}_i(j) = 2^\Delta \cdot \tilde{b}_{ij} \quad \forall i \in [r]$ & $j \in [0, D]$

We omit T *when it is obvious from the context.*

Remark 3. Theorem 1 states that, for $\nu = \left\lfloor \frac{d-1}{2} \right\rfloor$, the predicate $\mathsf{Pack}_t^{(\nu,\delta)}(a) = \mathsf{true}$ if and only if $a \in R$ contains $\mathbb{Z}_{2^k}$-messages with respect to the tweaked

[xi] Such tricky definition is useful when describing our ZKPoMK (Section 6.1).

interpolation packing. We call a polynomial a a (D, Δ, T)-tweaked interpolation packing if it satisfies $\mathsf{Pack}_T^{(D,\Delta)}(a)$.

Remark 4. The essence of Theorem 2(c) is the following fact:
If $\mathsf{Pack}_{k+2\delta}^{(\nu,\delta)}(a) \wedge \mathsf{Pack}_{k+2\delta}^{(\nu,\delta)}(b) = \mathsf{true}$, then $\mathsf{Deg}_{k+2\delta}^{(2\nu)}(a \cdot b) \wedge \mathsf{Div}_{k+2\delta}^{(\nu,2\delta)}(a \cdot b) = \mathsf{true}$.

3.5 Sampling Zero Polynomials in $\mathbb{Z}_{2^k}[X]$

We propose *efficient* random sampling algorithms from the sets of elements satisfying the *predicates* defined in Section 3.4. These play important roles when we construct our Reshare protocol (in Section 4) and our ZKP of Message Knowledge (ZKPoMK) (in Section 6). Readers may skip this subsection and consult it when succeeding sections refer to the definitions.

Due to the unique feature of $\mathbb{Z}_{2^k}$, sampling process is not trivial and has a deep connection with zero polynomials[xii] in $\mathbb{Z}_{2^k}[X]$. Our result possibly has ramifications on cryptographic works regarding polynomial evaluation (or interpolation) over $\mathbb{Z}_{2^k}$, outside of our protocols.

Definition 2 (Distribution with Predicate). *Let $\mathcal{U}(B)$ be the uniform distribution over $\{a \in R : \ ||a||_\infty \leq B\}$. For a predicate $\mathsf{P} \in \{\mathsf{Deg}, \mathsf{Div}\}$ (we omit the superscripts) over $R = \mathbb{Z}[X]/\Phi_M(X)$, the distribution $\mathcal{U}_\mathsf{P}(B)$ is the uniform distribution over the following set:*

$$\{a \in R : \quad ||a||_\infty \leq B \ \wedge \ \mathsf{P}(a) = \textit{true}\}.$$

To show that one can efficiently sample elements from $\mathcal{U}_\mathsf{P}(B)$ with $\mathsf{P} = \mathsf{Div}$, we first identify all zero polynomials in $\mathbb{Z}_{2^k}[X]$ as follows.

Lemma 2. *For $\chi_0(X) := 1$ and $\chi_i(X) := \prod_{\ell=0}^{i-1}(X - \ell) \in \mathbb{Z}_{2^k}[X]$, let $f(X) = \sum_{i=0}^{d} c_i \chi_i(X)$. Then, $f(j) = 0 \pmod{2^k}$ for all $j \in [0, n]$ if and only if $c_i \cdot i! = 0 \pmod{2^k}$ for all $i \in [0, n]$.*

Proof. Assume $f(j) = 0 \pmod{2^k}$ for all $j \in [0, n]$. We proceed by mathematical induction on i. First, since $f(0) = 0 \pmod{2^k}$, $c_0 \cdot 0! = c_0 = 0 \pmod{2^k}$. Assume $c_i \cdot i! = 0 \pmod{2^k}$ holds for all $0 \leq i < s \leq n$. Then, from the fact that $\chi_i(s) = 0$ for $i > s$ and that $i!$ divides $\chi_i(s)$, along with the induction hypothesis, the following equations hold.

$$0 = f(s) = \sum_{i=0}^{n} c_i \chi_i(s) = \sum_{i=0}^{s} c_i \chi_i(s) = c_s \chi_s(s) = c_s \cdot s! \pmod{2^k}$$

For the other direction, assume $c_i \cdot i! = 0 \pmod{2^k}$ holds for all $i \in [0, n]$. Since $i!$ always divides $\chi_i(j)$ for any $j \in \mathbb{Z}$, $c_i \chi_i(j) = 0 \pmod{2^k}$ holds. Then, $f(j) = \sum_{i=0}^{n} c_i \chi_i(j) = 0 \pmod{2^k}$ for all $j \in [0, n]$. □

[xii] A zero polynomial is a polynomial whose evaluations at certain points are all zero.

Corollary 1 (Zero Polynomials over $\mathbb{Z}_{2^k}$). *Let $f(X)$ be a polynomial in $\mathbb{Z}_{2^k}[X]$. Then, for a positive integer n, $f(j) = 0 \pmod{2^k}$ for all $j \in [0, n]$ if and only if $f(X)$ is of the form $\chi_{n+1}(X) \cdot q(X) + \sum_{i=0}^{n} c_i \chi_i(X)$ where c_i's are such that $c_i \cdot i! = 0 \pmod{2^k}$ for all $i \in [0, n]$.*

Proof. Note that $\{\chi_i(X)\}_{i=0}^{n}$ form a basis of the polynomials of degree at most n and $\chi_{n+1}(j) = 0$ for all $j \in [0, n]$. Then, the claim follows from Lemma 2. $\square$

With the identification of zero polynomials from Corollary 1, we can efficiently sample an element from the distribution $\mathcal{U}_{\mathsf{P}}(B)$ as follows.

Corollary 2 (Efficient Sampling from $\mathcal{U}_{\mathsf{P}}(B)$). *Let $\mathsf{P} \in \{\mathsf{Deg}_T^{(D)}, \mathsf{Div}_T^{(D,\Delta)}, \mathsf{Pack}_T^{(D,\Delta)}\}$ be a predicate over $R = \mathbb{Z}[X]/\Phi_M(X)$. Then, one can efficiently sample an element from the distribution $\mathcal{U}_{\mathsf{P}}(B)$, given that $T \geq \Delta \geq \nu_2(D!)$.*

Proof. In both cases, it suffices to sample an element satisfying the predicate from $\mathbb{Z}_{2^T}[X]/\Phi_M(X)$ first with CRT isomorphism (Eq.(1)), then add an element from the distribution $\mathcal{U}(B)$ conditioned on multiples of 2^T.

The case of $\mathsf{P} = \mathsf{Deg}$ is straightforward, since one can sample a polynomial of bounded degree on each CRT slot. For the cases of $\mathsf{P} = \mathsf{Div}$ and $\mathsf{P} = \mathsf{Pack}$, first note that differences of tweaked interpolation packings with same messages are zero polynomials. Fixing the representative (e.g., the one from Lemma 1) for tweaked interpolation packings with same messages, each CRT slot of an element satisfying P can be uniquely represented modulo 2^T by sum of the tweaked interpolation and a zero polynomial. Thus, to uniformly sample from each CRT slot of $\mathbb{Z}_{2^T}[X]/\Phi_M(X)$, first compute a tweaked interpolation (Lemma 1 with $\delta = \Delta, n = D$) with uniform random points from $\mathbb{Z}_{2^{T-\Delta}}$. Then, for $\mathsf{Div}_T^{(D,\Delta)}$ and $\mathsf{Pack}_T^{(D,\Delta)}$, add a random zero polynomial of degree at most d (Eq. (1)) and D, respectively, using Corollary 1 with $n = D$. $\square$

Finally, for the construction of ZKPoMK (Section 6), we present the adaptation of usual statistical masking method to our case with the predicates.

Lemma 3 (Statistical Masking). *For a positive integer $B < B_\infty$ and a predicate $\mathsf{P} \in \{\mathsf{Deg}, \mathsf{Div}, \mathsf{Pack}\}$, let $a \in R = \mathbb{Z}[X]/\Phi_M(X)$ be an element such that $||a||_\infty \leq B$ and $\mathsf{P}(a) = \mathit{true}$. Then, the statistical distance between $a + \mathcal{U}_{\mathsf{P}}(B_\infty)$ and $\mathcal{U}_{\mathsf{P}}(B_\infty)$ is bounded by $\frac{NB}{B_\infty}$ where $N = \varphi(M)$. The similar holds for $\mathcal{U}$.*

Proof. The case of $N = 1$ directly follows from the definition of statistical distance, and the claim is a generalization with $(B_\infty - B)^N > B_\infty^N - NB_\infty^{N-1}B$. $\square$

4 Reshare Protocol for Level-dependent Packings

When designing a packing method for $\mathbb{Z}_{2^k}$-messages with high parallelism, it is inevitable to design a *level-dependent* packing, e.g., the Overdrive2k [OSV20] packing (Section 2.4) and our tweaked interpolation packing (Section 3, Remark 1). However, this leads to a complication in the reshare protocol for $\mathbb{Z}_{2^k}$-messages, which does not occur in the case of a finite field $\mathbb{Z}_P$ with *level-consistent* packing from the isomorphism $\mathbb{Z}_P[X]/\Phi_{2^m}(X) \cong \mathbb{Z}_P^{\varphi(2^m)}$. In particular, the reshare protocol of Overdrive2k [OSV20] exploits an extra masking ciphertext with ZKPoPK on it, which is the most costly part, to remedy the issue.

In this section, we propose a new reshare protocol for *level-dependent* packings, which resolves this complication: our protocol extends the previous reshare protocol of the finite field case to operate also with level-dependent packings *without any extra cost*. Our result closes the gap between the finite field and the $\mathbb{Z}_{2^k}$ cases which originates from the level-dependency.

4.1 Improved Reshare Protocol for Level-dependent Packings

The Problem of Level-dependent Packings. Recall that the goal of the reshare protocols is, for an input level-zero ciphertext, to output shares of the underlying message along with a *level-one* ciphertext having the same message as the input (Section 2.5). The complication, with a level-dependent packing, is that we have to manage not only the *ciphertext level* but also the *packing level*.

Recall that one masking ciphertext $\mathsf{ct}_{\boldsymbol{f}}$ is used twice in the reshare protocol for the finite field case: once to mask the input ciphertext of level-zero and once to reconstruct the fresh ciphertext of level-one by subtracting it (Section 2.5). While the difference of ciphertext levels can be managed easily with modulus-switching, that of the packing levels seems to be problematic.

Solution of Overdrive2k. To resolve this problem, Overdrive2k [OSV20] provides two masking ciphertexts having the *same messages* but in *different packing*: one with level-zero packing and the other with level-one packing. This approach requires an extra ZKPoPK with the additional broadcast of the masking ciphertext, doubling the cost of the reshare protocol. It results in substantial increase of cost in the whole preprocessing protocol. In the triple generation protocol, the number of ZKPoPK with broadcasts of ciphertexts is *five* using the original reshare protocol in the field case, whereas Overdrive2k requires *seven* due to their reshare protocol, resulting roughly a 1.4x reduction in efficiency.[xiii]

Our Solution. The crux of our reshare protocol for level-dependent packings is the idea of generating the ciphertext ct_α of the MAC key $\alpha \in \mathbb{Z}_{2^s}$ by treating α as a constant in the cyclotomic ring $\mathbb{Z}_{2^t}/\Phi_M(X)$, i.e. $\mathsf{ct}_\alpha = \mathsf{Enc}(\alpha)$ for

[xiii] The number of ZKPoPK is counted regarding the *correlated* sacrifice technique [KOS16].

$\alpha \in \mathbb{Z}_{2^t}/\Phi_M(X)$ *without* any packing structure. Then, we actually do *not* need the fresh ciphertext to be of packing level-one: it is okay to be of packing level-zero. This is because, whereas multiplying $\mathfrak{ct}_\alpha$ to a ciphertext consumes a ciphertext level, multiplying α to a plaintext does not consumes a packing level, i.e. multiplying α is a linear operation in the aspect of packing (Theorem 2(b)).

Our reshare protocol itself is more or less verbatim of the previous reshare protocol for the finite field cases [DPSZ12]. Thus, we omit the formal description and proof of our reshare protocol for general level-dependent packings. Instead, we present an instantiation of our reshare protocol with our tweaked interpolation packing in the next subsection.

4.2 Compatibility with Our Packing Method

We present our reshare protocol instantiated with our tweaked interpolation packing (Section 3). While our protocol resembles the Reshare protocol of [DPSZ12] with $\mathbb{Z}_p$ messages, it is slightly more involved due to the nontrivial task of masking the $\mathbb{Z}_{2^k}$ messages encoded with our tweaked interpolation packing (we borrow the results of Section 3.5 for this). We give an overview focusing on our modification and correctness of the protocol, and refer to Appendix B.3 for the formal proof.

Our reshare protocol Π_{Reshare} is presented in Figure 3. The protocol exploits a zero-knowledge proof on a ciphertext, depicted as ZKPoPK and ZKPoMK, which will be described in Section 5,6. For now, we simply assume that they guarantee that the messages are encoded correctly in the ciphertext with respect to our packing method.

A noticeable difference of our protocol from other reshare protocols of [DPSZ12, OSV20] is that each party samples the message f_i of a mask ciphertext from the distribution with predicate, $\mathcal{U}_{\mathsf{P}}(2^T)$ with $\mathsf{P} = \mathsf{Div}_T^{(D,\Delta)}$ (Definition 2, Corollary 2). It not only *preserves* the packing structure, but also *prevent* the information leakage from our packing method in the following distributed decryption (5.-7. in Fig. 3). If f_i was sampled from a random polynomial without any restriction, $\mathsf{Div}_T^{(D,\Delta)}(v) = \mathsf{false}$ (with high probability) and each party cannot retrieve $[\boldsymbol{m}]_i$. On the other hand, if f_i was not added as a mask, each party can get additional information from the plaintext polynomial v which may contain more coefficients than the messages.

Since the mask r_i together with f_i can be seen as a statistical masking from $\mathcal{U}_{\mathsf{P}}(B_{\mathsf{DDec}})$ of Lemma 3, we can utilize the protocol to implement our preprocessing phase (Section 2.5, or formally, Π_{Prep} in Appendix B.4). See Appendix B.3 and B.4 for detailed descriptions.

Protocol Π_{Reshare}

Implicitly call $\mathcal{F}_{\mathsf{Rand}}$ (Appendix B.2) when it is required in ZKPoPK (or ZKPoMK). Each party P_i has $(\mathfrak{pk}, [\mathfrak{sk}]_i)$ given from $\mathcal{F}_{\mathsf{KeyGen}}$ (Appendix B.2).

PARAMETERS:

- B_{DDec}: a bound on the coefficients of the mask values.
- B_{noise}: a bound on the noise of input ciphertext.
- n: the number of participating parties P_i.

COMMON INPUT:

- The parameter $\mathsf{pp} = (D, \Delta, T)$ for the predicate $\mathsf{Div}_T^{(D,\Delta)}$ (Definition 1).
- $\mathfrak{ct}_{\boldsymbol{m}}^{(0)}$: a level-zero ciphertext satisfying that $\mathsf{Div}_T^{(D,\Delta)}(\mathsf{Dec}(\mathfrak{ct}_{\boldsymbol{m}}^{(0)}, \mathfrak{sk})) = \mathsf{true}$, i.e., having a message $\boldsymbol{m} \in \mathbb{Z}_{2^k}^{\nu}$ encoded with our tweaked interpolation packing (Theorem 1) with $\mathsf{pp} = (D, \Delta, T)$ (Section 3.4, Remark 3).

D2: On input ciphertext $\mathfrak{ct}_{\boldsymbol{m}}^{(0)}$ (see COMMON INPUT), parties do as follows.

1. Set $\mathsf{P} = \mathsf{Div}_T^{(D,\Delta)}$. Each P_i samples a polynomial $f_i \leftarrow \mathcal{U}_{\mathsf{P}}(2^{T-1})$ and set $\boldsymbol{f}_i \in \mathbb{Z}_{2^k}^{\nu}$ as the uniform random points used in the sampling process, i.e., $\boldsymbol{f}_i$ are messages of f_i when regarded as a tweaked interpolation packing (see the proof of Corollary 2).
2. Each P_i generates level-one ciphertext $\mathfrak{ct}_{\boldsymbol{f}_i}^{(1)}$ having the polynomial f_i as a plaintext, then broadcasts this ciphertext. All parties compute $\mathfrak{ct}_{\boldsymbol{f}}^{(1)} = \sum_{i\in[n]} \mathfrak{ct}_{\boldsymbol{f}_i}^{(1)}$, then perform ZKPoPK and ZKPoMK (Section 5, 6) on it with parameter $\mathsf{pp} = (D, \Delta, T)$. If the proof of ZKPoPK or ZKPoMK is rejected, then abort.
3. All parties compute $\mathfrak{ct}_{\boldsymbol{f}}^{(0)} = \mathsf{ModSwitch}(\mathfrak{ct}_{\boldsymbol{f}}^{(1)})$, then compute $\mathfrak{ct}_{\boldsymbol{m}+\boldsymbol{f}}^{(0)} = \mathfrak{ct}_{\boldsymbol{m}}^{(0)} \boxplus \mathfrak{ct}_{\boldsymbol{f}}^{(0)}$. Let $\mathfrak{ct}_{\boldsymbol{m}+\boldsymbol{f}}^{(0)}$ be (c_0, c_1).
4. Each P_i computes $w_i = \begin{cases} c_0 - [\mathfrak{sk}]_1 \cdot c_1 & \text{if } i = 1 \\ -[\mathfrak{sk}]_i \cdot c_1 & \text{if } i \neq 1 \end{cases}$.
5. Each P_i samples a mask $r_i \leftarrow \mathcal{U}(B_{\mathsf{DDec}}/2^T)$ (Definition 2), then broadcasts $v_i = w_i + 2^T \cdot r_i \pmod{q_0}$.
6. All parties compute $v = \sum_i v_i \pmod{q_0}$, then check if $||v||_\infty < B_{\mathsf{noise}} + n \cdot B_{\mathsf{DDec}}$ and $\mathsf{Div}_T^{(D,\Delta)}(v) = \mathsf{true}$. If not, abort.
7. All parties retrieve $\boldsymbol{m} + \boldsymbol{f}$ from v by regarding v as a Tweaked Interpolation Packing (Theorem 1) with $\delta = \Delta$, $\lfloor \frac{d-1}{2} \rfloor = D$, and $t = T$.
8. Each P_i sets $[\boldsymbol{m}]_i = \begin{cases} (\boldsymbol{m} + \boldsymbol{f}) - \boldsymbol{f}_1 & \text{if } i = 1 \\ -\boldsymbol{f}_i & \text{if } i \neq 1 \end{cases}$.
9. All parties compute, using default value (e.g., $\mathbf{0}$) for the randomness,
$$\bar{\mathfrak{ct}}_{\boldsymbol{m}}^{(1)} = (\mathsf{Enc}(m + f, \mathbf{0}; \mathfrak{pk})) \boxminus \mathfrak{ct}_{\boldsymbol{f}}^{(1)},$$
where the polynomial $m + f \in \mathbb{Z}_{2^t}[X]/\Phi_M(X)$ is the Tweaked Interpolation Packing (Theorem 1) for the message $\boldsymbol{m} + \boldsymbol{f} \in \mathbb{Z}_{2^k}^{\nu}$ with $\delta = \Delta$, $\lfloor \frac{d-1}{2} \rfloor = D$, and $t = T$.

Fig. 3. Our reshare protocol

5 Better ZKP for Lattice Encryption on $\mathbb{Z}[X]/\Phi_p(X)$

We present an improved ZKP of Plaintext Knowledge (ZKPoPK) for [BGV14] ciphertexts over *prime* cyclotomic rings $\mathbb{Z}[X]/\Phi_p(X)$, which proves that a ciphertext is generated with appropriate *sizes* of noises and a plaintext. ZKPoPK plays an important role in SHE-based MPC preprocessing phases [DPSZ12, KPR18, OSV20] as it restricts adversaries from submitting maliciously generated ciphertexts.

Note that power-of-two cyclotomic polynomials $\Phi_{2^m}(X)$ are detrimental for $\mathbb{Z}_{2^k}$-messages.[xiv] Accordingly, Overdrive2k [OSV20] proposed a ZKPoPK over *prime* cyclotomic rings, adapting the High Gear approach of Overdrive [KPR18] which is over power-of-two cyclotomic rings. Likewise to Overdrive, the challenge space of Overdrive2k is restricted to a rather small set: $\{0, 1\}$.

Taking one step further, we propose a ZKPoPK named TopGear2k for prime cyclotomic rings, adapting the state-of-the-art ZKPoPK over power-of-two cyclotomic rings called TopGear [BCS19]. Our ZKPoPK, similarly as TopGear, allows a *larger challenge space* $\{X^j\}_j \cup \{0\}$, resulting in a better efficiency. The essence is a new observation that the core properties of power-of-two cyclotomic rings (observed in [BCK$^+$14]) also hold similarly in prime cyclotomic rings. Our result possibly has ramifications on works derived from [BCK$^+$14], outside of our specific ZKPoPK.

5.1 A Technical Lemma on Cyclotomic Polynomials of Primes

We present a technical lemma on cyclotomic polynomials of primes, which is the essence of our ZKPoPK protocol. We first recall some facts on $R = \mathbb{Z}[X]/\Phi_M(X)$ when M is a power-of-two, which are the main ingredients of the TopGear protocol [BCS19] and its forebear [BCK$^+$14].

(a) For all $a(X) \in R$ and $i \in \mathbb{Z}$, it holds that $||a(X) \cdot X^i||_\infty = ||a(X)||_\infty$.
(b) ([BCK$^+$14, Lemma 4]) For all $1 \le j < i \le M$, there exists $h(X) \in R$ such that
  - $(X^i - X^j) \cdot h(X) \equiv 2 \pmod{\Phi_M(X)}$
  - and $||h(X)||_\infty = 1$.

Statement (a) indicates that the coefficients do not grow when multiplied by X^i, which is straightforward from the fact that multiplication by X^i acts as *skewed* coefficient shift in $\mathbb{Z}[X]/(X^{M/2}+1)$. On the other hand, (b) says, roughly, that there is a *scaled* inverse of $(X^i - X^j)$ in R with small coefficients.

We now present an analogue of the above facts when M is a prime.

Lemma 4. *For a prime p and $R := \mathbb{Z}[X]/\Phi_p(X)$, the followings hold.*

(a) *For all $a(X) \in R$ and $i \in \mathbb{Z}$, it holds that $||a(X) \cdot X^i||_\infty \le 2||a(X)||_\infty$.*
(b) *For all $1 \le j < i \le p$, there exists $h(X) \in R$ such that*

[xiv] For $k > 1$, the ring $\mathbb{Z}_{2^k}[X]/\Phi_{2^m}(X)$ never split into a product of smaller rings, resulting low packing density (see e.g. [CL21]).

- $(X^i - X^j) \cdot h(X) \equiv p \pmod{\Phi_p(X)}$
- *and* $||h(X)||_\infty \leq p - 1$.

Proof. (a) Let $\tilde{a}(X) \in \mathbb{Z}[X]$ be the representative of $a(X)$ with the minimal degree. When reduced modulo $(X^p - 1)$, every monomials of $\tilde{a}(X) \cdot X^i$ are reduced to distinct-degree monomials preserving the coefficients. Let us denote the ℓ-th coefficient of $(\tilde{a}(X) \cdot X^i \mod (X^p - 1))$ as $\tilde{a}_\ell^{(i)}$. Applying modulo $\Phi_p(X)$ to $(\tilde{a}(X) \cdot X^i \mod (X^p - 1))$, the ℓ-th coefficients of $(\tilde{a}(X) \cdot X^i \mod \Phi_p(X))$ equals $(\tilde{a}_\ell^{(i)} - \tilde{a}_{(p-1)}^{(i)})$, and the inequality $||a \cdot X^i||_\infty \leq 2||a||_\infty$ follows.

(b) Consider the following polynomial in $\mathbb{Z}[X]$.

$$v(X) := \frac{\Phi_p(X) - p}{X - 1} = \sum_{k=0}^{p-1} (p - 1 - k) \cdot X^k$$

We claim that $\tilde{h}(X) := -X^{p-j} \cdot v(X^{i-j}) \in \mathbb{Z}[X]$ satisfies the conditions after being reduced by $\Phi_p(X)$. By definition, the first condition can be easily checked with the fact that $\Phi_p(X)$ divides $\Phi_p(X^{i-j})$ since p does not divide $(i - j)$.

Since p does not divide $(i-j)$, when reduced modulo (X^p-1), every monomials of $\tilde{h}(X)$ are reduced to distinct-degree monomials with coefficients remaining in the interval $[1-p, 0]$. Let us denote the ℓ-th coefficient of $(\tilde{h}(X) \pmod{(X^p-1)})$ as $\tilde{h}_\ell \in [1-p, 0]$. Applying modulo $\Phi_p(X)$ to $(\tilde{h}(X) \pmod{(X^p-1)})$, the ℓ-th coefficients of $(\tilde{h}(X) \pmod{\Phi_p(X)})$ equals $(\tilde{h}_\ell - \tilde{h}_{(p-1)})$. Certainly, $(\tilde{h}_\ell - \tilde{h}_{(p-1)})$ lies in the interval of $[1-p, p-1]$. Thus, the inequality $||\tilde{h}(X) \pmod{\Phi_p(X)}||_\infty \leq p-1$ holds. □

5.2 TopGear2k: Better ZKPoPK over $\mathbb{Z}[X]/\Phi_p(X)$

We describe our ZKPoPK protocol named TopGear2k for BGV ciphertexts with prime cyclotomic rings $\mathbb{Z}[X]/\Phi_p(X)$. In a high level, our ZKPoPK is a batched Schnorr-like protocol as those of SPDZ-family [DPSZ12, KPR18, OSV20].

ZKPoPK Framewok — Schnorr-like Protocol with Predicates. We first introduce the ZKPoPK framework of SPDZ-family which proceeds as the standard *batched* Schnorr-like protocols [CD09] to prove that the underlying plaintext satisfies a certain predicate. While our protocol (Fig. 4) follows the *global* proof style of Overdrive [KPR18] for efficiency, we describe in *per-party* proof style of SPDZ [DPSZ12] for simplicity.

To prove that a plaintext vector $\boldsymbol{a} = (a_i)_{i=1}^u, (a_i \in R := \mathbb{Z}[X]/\Phi_M(X))$ of input ciphertexts $\mathfrak{ct}_{\boldsymbol{a}} = (\mathsf{Enc}(a_i))_{i=1}^u$ satisfy a given predicate $\mathsf{P} : R \to \{\mathsf{true}, \mathsf{false}\}$[xv], the prover publishes a vector of masking ciphertexts $\mathfrak{ct}_{\boldsymbol{y}}$ for a plaintext vector $\boldsymbol{y} \in R^v$ satisfying P. Then, after the verifier queries a challenge matrix $W \in R^{v \times u}$, the prover publishes a plaintext vector $\boldsymbol{z} \in R^v$ with which

[xv] The predicate, for example, can capture the boundedness of the sizes of plaintext and randomnesses, or the correctness of packing (Definition 1).

the verifier checks if $\mathsf{P}(z) = \mathsf{true}$ and $\mathfrak{ct}_{\boldsymbol{y}} + W \cdot \mathfrak{ct}_{\boldsymbol{a}} = \mathfrak{ct}_{\boldsymbol{z}}$. The prover/verifier do similar proofs/checks on the randomnesses required in the encryptions.

Then, the usual rewinding argument guarantees that the elements of $\boldsymbol{a}$ also satisfy P as follows: by inverting the equation on plaintexts $(W - \overline{W}) \cdot \boldsymbol{a} = \boldsymbol{z} - \bar{\boldsymbol{z}}$ derived from the two accepting transcripts with different challenge matrices W and $\overline{W}$, we deduce that $\boldsymbol{a}$ also satisfies the predicate P given that $\mathsf{P}(\boldsymbol{z}) = \mathsf{P}(\bar{\boldsymbol{z}}) = \mathsf{true}$. Note, for this argument to work, two conditions are required: (a) the difference $(W - \overline{W})$ should satisfy some types of *invertibility*, so that one can derive, e.g., $\boldsymbol{a} = (W - \overline{W})^{-1} \cdot (\boldsymbol{z} - \bar{\boldsymbol{z}})$, (b) the predicate should be *homomorphic* under (additions and) multiplications by challenge matrices W (and also by *pseudo-inverses* of their differences), i.e. $\mathsf{P}(\boldsymbol{a}) = \mathsf{true} \implies \mathsf{P}(W \cdot \boldsymbol{a}) = \mathsf{true}$ (and similarly for the pseudo-inverse).

Here, the difficulty is to identify a *nice* challenge space, where the elements of W are sampled from, which meets all of the above conditions. In the previous works [DPSZ12, KPR18, OSV20], the challenge space is restricted to the set $\{0, 1\}$ (and the form of W was also restricted) to satisfy the above conditions. In this case, however, v (the size of masking ciphertext vector) should be as large as the soundness security parameter, leading to substantial inefficiency.

TopGear Review. TopGear [BCS19] offers the most efficient ZKPoPK among the line of works [DPSZ12, KPR18] exploiting (S)HE to MPC over finite fields with power-of-two cyclotomic rings. It is also a batched Schnorr-like protocol (described above) with global proof approach. The essence of their work is to use a *larger challenge space* $\mathsf{Chal} = \{X^j\}_{j=1}^{2^m} \cup \{0\}$ than $\{0, 1\}$ of the other previous works. This is an adaptation of the nice properties (Section 5.1) of power-of-two cyclotomic ring $\mathbb{Z}[X]/\Phi_{2^m}(X)$ from [BCK+14] to the ZKPoPK framework, and is desirable in communication cost, latency, and memory consumption.

Extending the result of TopGear to other cyclotomic polynomials, however, was an open problem, e.g., Overdrive2k [OSV20] exploited a rather small challenge space of $\{0, 1\}$, mentioning that "TopGear improvements cannot be applied directly" to their work.

TopGear2k: Our ZKPoPK over $\mathbb{Z}[X]/\Phi_p(X)$. Following the above framework, we propose ZKPoPK named TopGear2k which is a batched Schnorr-like protocol with global proofs, working over prime cyclotomic rings $\mathbb{Z}[X]/\Phi_M(X)$ ($M = p$ is a prime[xvi]) with larger challenge space $\mathsf{Chal} = \{X^j\}_{j=1}^{M} \cup \{0\}$, adapting Lemma 4. Our ZKPoPK is a prime cyclotomic ring analogue of the ZKPoPK of TopGear [BCS19] over power-of-two cyclotomic rings. The full description of our ZKPoPK protocol TopGear2k ($\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$) is given in Fig. 4.

Our TopGear2k aims to prove that the given ciphertexts are generated with appropriate sizes of a plaintext and randomnesses. If all parties run $\mathsf{Sampling}$

[xvi] We denote p as the smallest prime factor of M. This is to consider the general case of $M = p^s$ and $M = p^s q^t$ in Section 5.4.

Protocol $\Pi^{\mathsf{TG2k}}_{\mathsf{PoPK}}$

PARAMETERS:
- $\mathsf{ZK\_sec}$: the zero-knowledge security parameter.
- 2^t: the plaintext modulus.
- u: the number of ciphertexts to be verified in one protocol execution.
- v: the number of masking ciphertexts (related to soundness probability).
- n: the number of participating parties P_i ($i \in [n]$).

$\mathsf{Sampling}_i$ (Sampling phase for the ith party P_i)

1. For each $k \in [u]$ do
   (a) Choose a plaintext $a_k^i \in \mathbb{Z}_{2^t}[X]/\Phi_M(X)$ and proper randomness $(r_{a_k}^{(i)})$. [xvii]
   (b) Compute a ciphertext $\mathfrak{ct}_{a_k}^i = \mathsf{Enc}(a_k^i, r_{a_k}^i; \mathfrak{pk})$.
2. Let $\mathfrak{ct}_a^i = (\mathfrak{ct}_{a_1}^i, \mathfrak{ct}_{a_2}^i, \ldots, \mathfrak{ct}_{a_u}^i)$, $\boldsymbol{a}^i = (a_1^i, a_2^i, \ldots, a_u^i)$, and $\boldsymbol{r}_a^i = (r_{a_1}^i, r_{a_2}^i, \ldots, r_{a_u}^i)$.
3. Output $(\mathfrak{ct}_a^i, \boldsymbol{a}^i, \boldsymbol{r}_a^i)$.

Commit (Commitment phase)

1. To generate v masking ciphertexts, each party P_i do the followings, for each $l \in [v]$.
   (a) P_i samples $y_l^i \leftarrow \mathcal{U}(2^{\mathsf{ZK\_sec}} \cdot 2^{t-1})$ and $r_{y_l}^i = (r_{y_l}^{i,(\ell)} \leftarrow \mathcal{U}(2^{\mathsf{ZK\_sec}} \cdot \rho_\ell))_{\ell \in [3]}$.
   (b) P_i computes $\mathfrak{ct}_{y_l}^i = \mathsf{Enc}(y_l^i, r_{y_l}^i; \mathfrak{pk})$.
2. Party P_i keeps $\mathsf{state}_i = (\boldsymbol{y}^i, \boldsymbol{r}_y^i)$ where $\boldsymbol{y}^i = (y_l^i)_{l \in [v]}$ and $\boldsymbol{r}_y^i = (r_{y_l}^i)_{l \in [v]}$.
3. Party P_i broadcasts $\mathsf{comm}_i = \mathfrak{ct}_y^i$ where $\mathfrak{ct}_y^i = (\mathfrak{ct}_{y_l}^i)_{l \in [v]}$.

$\mathsf{Challenge}$ (Challenge phase)

1. Parties together uniformly sample challenge matrix W of size $v \times u$, whose entries are sampled from the challenge space $\mathsf{Chal} = \{X^j\}_{j=1}^M \cup \{0\}$.

$\mathsf{Response}$ (Response phase)

1. Each party P_i computes $\boldsymbol{z}^i = \boldsymbol{y}^i + W \cdot \boldsymbol{a}^i$ and $\boldsymbol{r}_z^i = \boldsymbol{r}_y^i + W \cdot \boldsymbol{r}_a^i$.[xviii]
2. Party P_i sets $\mathsf{resp}_i = (\boldsymbol{z}^i, \boldsymbol{r}_z^i)$ and broadcasts resp_i.

Verify (Verification phase)

1. Each party P_i computes,
   (a) $\mathfrak{ct}_z^i = (\mathsf{Enc}(\boldsymbol{z}_l^i, \boldsymbol{r}_{z_l}^i; \mathfrak{pk}))_{l \in [v]}$.
   (b) $\mathfrak{ct}_a = \sum_{i=1}^n \mathfrak{ct}_a^i$, $\mathfrak{ct}_y = \sum_{i=1}^n \mathfrak{ct}_y^i$, $\mathfrak{ct}_z = \sum_{i=1}^n \mathfrak{ct}_z^i$.
   (c) $\boldsymbol{z} = \sum_{i=1}^n \boldsymbol{z}^i$, $\boldsymbol{r}_z = \sum_{i=1}^n \boldsymbol{r}_z^i$.
2. Parties accept if all of the followings hold, otherwise they reject.
   (a) $\mathfrak{ct}_z = \mathfrak{ct}_y + W \cdot \mathfrak{ct}_a$.
   (b) For $l \in [v]$,

$$||z_l||_\infty \leq n \cdot 2^{\mathsf{ZK\_sec}} \cdot 2^t, \quad ||r_{z_l}^{(\ell)}||_\infty \leq n \cdot 2^{\mathsf{ZK\_sec}+1} \cdot \rho_\ell \ \text{ for } \ell \in [3]. \tag{2}$$

xvii Sample $(r^{(1)}, r^{(2)}, r^{(3)})$ where $r^{(1)}, r^{(2)} \leftarrow \mathsf{DG}(\sigma^2)$ and $r^{(3)} \leftarrow \mathsf{ZO}(\rho)$ (Section 2.2).

xviii This means that $\boldsymbol{r}_z^{i,(\ell)} = \boldsymbol{r}_y^{i,(\ell)} + W \cdot \boldsymbol{r}_a^{i,(\ell)}$ for each $\ell \in [3]$.

Fig. 4. Protocol $\Pi^{\mathsf{TG2k}}_{\mathsf{PoPK}}$

honestly, then the outputs satisfy the following relation:

$$\mathcal{R}^{u}_{\mathsf{PoPK}} := \Bigg\{ \text{input } \left(\left\{ \left(\mathfrak{ct}^{i}_{a_k} \right)_{i=1}^{n} \right\}_{k \in [u]}, \mathfrak{pk} \right), \text{witness } \left(\left\{ \left(a^{i}_{k}, r^{i}_{k} \right)_{i=1}^{n} \right\}_{k \in [u]} \right) : \\ \text{For all } k \in [u], \quad \mathfrak{ct}_{a_k} = \sum_{i=1}^{n} \mathfrak{ct}^{i}_{a_k},\ a_k = \sum_{i=1}^{n} a^{i}_{k},\ r_k = \sum_{i=1}^{n} r^{i}_{k}, \\ \mathfrak{ct}_{a_k} = \mathsf{Enc}(a_k, r_k; \mathfrak{pk}),\ \|a_k\|_\infty \le n \cdot 2^{t-1},\ \|r^{(j)}_{k}\|_\infty \le n\rho_j\ (\forall j \in [3]) \Bigg\},$$

where $\rho_1 = \rho_2 = 20$, and $\rho_3 = 1$ are the bound of noises and randomnesses, while 2^t is the plaintext modulus.

However, our protocol only guarantees that the given ciphertexts $\{\mathfrak{ct}_k\}_{k\in[u]}$ satisfies the following relation $\mathcal{R}^{S,u}_{\mathsf{PoPK}}$ which is relaxed from $\mathcal{R}^{u}_{\mathsf{PoPK}}$:

$$\mathcal{R}^{S,u}_{\mathsf{PoPK}} := \Big\{ \text{the same input and witness as } \mathcal{R}^{u}_{\mathsf{PoPK}} : \\ \text{For all } k \in [u], \quad \mathfrak{ct}_{a_k}, a_k, r_k \text{ are defined the same as } \mathcal{R}^{u}_{\mathsf{PoPK}}, \\ \mathfrak{ct}_{a_k} = \mathsf{Enc}(a_k, r_k; \mathfrak{pk}), \\ \|a_k\|_\infty \le nS \cdot 2^{t-1},\ \|r^{(j)}_{k}\|_\infty \le nS\rho_j\ (\forall j \in [3]) \Big\}, \tag{3}$$

where S is called a *soundness slack.* This soundness slack S comes from the rewinding process and appears also in the previous ZKPoPKs [DPSZ12, KPR18, BCS19, OSV20] for MPC and ZKPs for lattice encryptions [BCK⁺14]. Meanwhile, it is standard to design the (S)HE-based MPC preprocessing phase so that it runs correctly even with the soundness slack, e.g., by enlarging the ciphertext modulus.

5.3 Correctness, Zero-Knowledge, and Soundness

We show that $\Pi^{\mathsf{TG2k}}_{\mathsf{PoPK}}$ satisfies the correctness, soundness, and zero-knowledge properties. For correctness, it suffices to show that honest inputs pass the checks in line 2 of Verify algorithm, which can be done by setting the parameters considering Lemma 4(a).

Theorem 3 (Correctness). *The n-party ZKPoPK protocol $\Pi^{\mathsf{TG2k}}_{\mathsf{PoPK}}$ (Fig. 4) with $u \le 2^{\mathsf{ZK\_sec}-1}$ satisfies the following* ***Correctness:***

– *If all parties P_i, with inputs sampled using* Sampling *algorithm, follow the protocol honestly, then* Verify *algorithm outputs accept with probability one.*

Proof. The correctness of the equality check (a) in line 2 of Verify is trivial. For the bound checks (b), let $(W)_l \cdot \boldsymbol{a}^i$ denotes the innerproduct between the l-th row of W and the vector $\boldsymbol{a}^i$. Then, by the equality $\boldsymbol{z}^i = \boldsymbol{y}^i + W \cdot \boldsymbol{a}^i$ and Lemma 4(a),

$$||z_l||_\infty = ||\sum_{i=1}^{n} z_l^i||_\infty \le \sum_{i=1}^{n} ||y_l^i + (W)_l \cdot \boldsymbol{a}^i||_\infty$$
$$\le n \cdot (2^{\mathsf{ZK\_sec}} \cdot \frac{2^t}{2} + u \cdot 2 \cdot \frac{2^t}{2}) \le n \cdot 2^{\mathsf{ZK\_sec}} \cdot 2^t,$$

where the final inequality follows from our assumption $u \le 2^{\mathsf{ZK\_sec}-1}$. The bound on $r_{z_l}^{(\ell)}$ can be proved similarly. □

Zero-knowledgeness essentially follows from the fact that the $\boldsymbol{y}^i$'s in protocol $\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$ can statistically mask the responses with Lemma 3.

Theorem 4 (Zero-Knowledge). *The n-party ZKPoPK protocol $\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$ (Fig. 4) satisfies the following **Honest-verifier Zero-knowledge**:*

– *There exists a PPT algorithm $S_{I'}$ indexed by a (honest) set $I' \subset [n]$, which takes as input an element in $\mathcal{R}_{\mathsf{PoPK}}^{u}$ and a challenge W, and outputs tuples $\{\mathsf{comm}_i, \mathsf{resp}_i\}_{i \in I'}$ such that this output is statistically indistinguishable from a valid execution of the protocol (with statistical distance $\le 8Muv/2^{\mathsf{ZK\_sec}}$).*

Proof. Let the simulator $S_{I'}$ output resp_i by sampling each component from the uniform distribution with sufficiently large bound, e.g., sample $\boldsymbol{z}^i = (z_l^i)_{l \in [v]}$ where $z_l^i \leftarrow \mathcal{U}(2^{\mathsf{ZK\_sec}} \cdot 2^{t-1})$. Then it outputs comm_i by computing each component from the challenge W and corresponding input ciphertexts, e.g., $\mathfrak{ct}_y^i = \mathsf{Enc}(\boldsymbol{z}^i, \boldsymbol{r}_z^i; \mathfrak{pk}) - W \cdot \mathfrak{ct}_a^i$.

Note that the statistical distance between the simulated and the real execution is determined by that between the distribution of resp_i in both executions (since each comm_i is computed in the same way from resp_i). In the real execution, $\boldsymbol{z}^i$ is computed by sampling $\boldsymbol{y}^i$ and adding $W \cdot \boldsymbol{a}^i$. Thus, Lemma 3 (without P) gives that the distance between $\boldsymbol{z}^i$ from both executions are bounded by $\varphi(M)\frac{||(W)_l \cdot \boldsymbol{a}^i||_\infty}{2^{\mathsf{ZK\_sec}} \cdot 2^{t-1}} \cdot \le \frac{2Mu}{2^{\mathsf{ZK\_sec}}}$, and similar results hold for $\boldsymbol{r}_z^i$. □

Finally, the soundness of $\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$ follows from the usual rewinding argument leveraging Lemma 4(b) on invertibility.

Theorem 5 (Soundness). *Assume that the n-party ZKPoPK protocol $\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$ (Fig. 4) is parameterized with $v \ge (\mathsf{Snd\_sec} + 2)/\log(|\mathsf{Chal}|)$ where $\mathsf{Snd\_sec}$ is the soundness security parameter and $|\mathsf{Chal}|$ is the size of the challenge space. Then, it satisfies the **Soundness** (see [BCS19, Definition 1] or Appendix B.6) with soundness probability $2^{-\mathsf{Snd\_sec}}$ and slack $S = 8\varphi(M) \cdot 2^{\mathsf{ZK\_sec}}$.*

Proof. The proof mostly resembles that of [BCS19, Theorem 1], and we give detailed description focusing on the unique aspects of our protocol. With a usual rewinding argument (we refer to [BCS19, Theorem 1] for formal description of an extractor), an extractor can output $(W, \{\boldsymbol{z}^i, \boldsymbol{r}_z^i\}_{i=1}^n)$ and $(\overline{W}, \{\bar{\boldsymbol{z}}^i, \bar{\boldsymbol{r}}_z^i\}_{i=1}^n)$, which are two accepting transcripts corresponding to $\mathfrak{ct}_a$ and $\mathfrak{ct}_y$ such that W and $\overline{W}$

are identical except k-th column. Let $\boldsymbol{z} := \sum_{i=1}^{n} \boldsymbol{z}^i$ and similarly for $\boldsymbol{r}_z$, $\bar{\boldsymbol{z}}$, $\bar{\boldsymbol{r}}_z$. Then, since these values satisfy the equation at line 2(a) of Verify algorithm (Fig. 4) and ciphertexts have homomorphic property, we get $\boldsymbol{z} = \boldsymbol{y} + W \cdot \boldsymbol{a}$ and $\bar{\boldsymbol{z}} = \boldsymbol{y} + \overline{W} \cdot \boldsymbol{a}$. With subtraction, since W and $\overline{W}$ are identical except k-th column, we get,

$$z_l - \bar{z}_l = (w_{l,k} - \bar{w}_{l,k}) \cdot a_k \text{ for some } l \in [v],$$

where $w_{l,k}$ and $\bar{w}_{l,k}$ are entries of W and $\overline{W}$ and are from $\{X^j\}_{j=1}^{M}$. Thus, multiplying $h(X)$ (of Lemma 4 (b)) according to $(w_{l,k} - \bar{w}_{l,k})$ on both sides, we get

$$\begin{aligned} ||p \cdot a_k||_\infty = ||h(X) \cdot (z_{1,l} - \bar{z}_{1,l})||_\infty &\leq 2 \cdot \varphi(M) \cdot ||h(X)||_\infty \cdot ||z_{1,l} - \bar{z}_{1,l}||_\infty \\ &\leq 2 \cdot \varphi(M) \cdot (p-1) \cdot ||z_{1,l} - \bar{z}_{1,l}||_\infty \\ &\leq 2 \cdot \varphi(M) \cdot (p-1) \cdot 2 \left(n \cdot 2^{\mathsf{ZK\_sec}} \cdot 2^t\right). \end{aligned}$$

The first inequality follows by regarding $h(X)$ as sum of monomials then applying Lemma 4 (a). The second inequality is obtained by the definition of $h(X)$ (Lemma 4 (b)). The last inequality follows from Eq. (2) (Fig. 4). Hence, $||a_k||_\infty \leq nS \cdot 2^{t-1}$ with the desired soundness slack $S = 8\varphi(M) \cdot 2^{\mathsf{ZK\_sec}}$. Similarly, one can derive the bound and slackness on the r_{a_k} from $\boldsymbol{r}_z, \bar{\boldsymbol{r}}_z$ in the transcripts. □

5.4 Extension to $\Phi_{p^s}(X)$ and $\Phi_{p^s q^t}(X)$

In fact, we can extend our ZKPoPK to work over cyclotomic polynomials $\Phi_M(X)$ with $M = p^s$ or $M = p^s q^t$ where p, q are primes satisfying $p < q$ and s, t are positive integers. Then, we can increase the packing density of our packing by taking cyclotomic polynomials of composites into consideration, which allow parameters with smaller $d = \mathrm{ord}_M(2)$ (see Section 3.3).

These follow from the results of [CKKL21] which are generalization of Lemma 4 to the cases with $M = p^s$ or $M = p^s q^t$. Then, in both cases of $\Phi_{p^s}(X)$ and $\Phi_{p^s q^t}(X)$, the protocol $\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$ is exactly the same with the prime case. In the case of p^s, the statements and the proofs of Theorem 3, 4, 5 also stay exactly the same. (We carefully distinguished the role of M and p for this.) In the case of $p^s q^t$, the major changes are the followings: the condition on u in Theorem 3 is $u \leq 2^{\mathsf{ZK\_sec}-1}/p$, the statistical distance in Theorem 4 is bounded by $8pMuv/2^{\mathsf{ZK\_sec}}$, and the soundness slack in Theorem 5 is $S = 8p^2 M \cdot 2^{\mathsf{ZK\_sec}}$.

6 Zero-Knowledge Proof of Message Knowledge

In SHE with messages from a finite field $\mathbb{Z}_P$, the plaintext space $\mathbb{Z}_P[X]/\Phi_{2^m}(X)$ can be taken to be *isomorphic* to $\mathbb{Z}_P^{\varphi(2^m)}$, a product of message spaces. When we deal with messages from $\mathbb{Z}_{2^k}$, however, the plaintext space $\mathbb{Z}_{2^t}[X]/\Phi_M(X)$ is never isomorphic to a product of $\mathbb{Z}_{2^k}$'s. It is inevitable that some plaintexts do not correspond to any packing of messages (see [CL21]). Thus, we must guarantee, in MPC preprocessings for $\mathbb{Z}_{2^k}$-messages, that each party encrypted a *valid* plaintext according to a specific packing method, in addition to the guarantee of valid encryption. This is an intricacy of the $\mathbb{Z}_{2^k}$-case that differs from the $\mathbb{Z}_P$-case where ZKPoPK (for the guarantee of valid encryption) is sufficient [DPSZ12, KPR18, BCS19].

Therefore, we propose, in addition to ZKPoPK, a Zero-Knowledge Proof of Message Knowledge (ZKPoMK) which guarantees that the given ciphertext is generated with a plaintext which is a *valid encoding* with respect to our tweaked interpolation packing (Section 3).[xix]

6.1 ZKPoMK for Tweaked Interpolation Packing

As our ZKPoPK, our ZKPoMK is a batched Schnorr-like protocol with predicates, and it proceeds similarly but with appropriate challenge spaces for the predicates which capture the valid plaintexts of our packing method. Since most parts of our ZKPoMK are similar to the ZKPoPK, here we only give an overview and refer to Appendix B.5 (Fig. 12,13, Π_{PoMK}) for the full description.

Overview of Our ZKPoMK. Recall the predicates (Definition 1) presented in Section 3.4 and that $a \in R$ is a valid plaintext, i.e. a tweaked interpolation packing of Theorem 1, if and only if, for $D = \left\lfloor \frac{d-1}{2} \right\rfloor$, $\Delta = \delta$, and $T = t$,

$$\mathsf{Pack}_T^{(D,\Delta)}(a) \iff \mathsf{Deg}_T^{(D)}(a) \wedge \mathsf{Div}_T^{(D,\Delta)}(a).$$

Our ZKPoMK separately proves those two statements (i) $\mathsf{Deg}_T^{(D)}(a) = \mathsf{true}$ and (ii) $\mathsf{Div}_T^{(D,\Delta)}(a) = \mathsf{true}$ as follows.

For the statement (i), we run the same as our $\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$ (Fig. 4) but with two modifications: (1) set the predicate $\mathsf{P} = \mathsf{Pack}_T^{(D,\Delta)}$ then sample the masks y_l^i from $\mathcal{U}_{\mathsf{P}}(2^{\mathsf{ZK\_sec}} \cdot 2^{t-1})$ using Corollary 2 and check if $\mathsf{P}(z_l) = \mathsf{true}$, instead of the bound check on it; (2) set the challenge space $\mathsf{Chal} = [-2^E + 1, 2^E] \cap \mathbb{Z}$ for a positive integer E. Note that these *constants* from the challenge space *preserve* the degree of given element a when multiplied, giving the key equation for the rewinding argument (and the soundness), while *enlarging* the challenge space. We remark that this approach introduces a new type of *slackness* which will be described later in this section.

[xix] Overdrive2k [OSV20] performs ZKPoMK implicitly in their ZKPoPK. If we set $\mathsf{Chal} = \{0, 1\}$ as their ZKPoPK, our ZKPoMK can also be integrated into ZKPoPK (by additionally checking if z is a valid encoding), resulting in our MHz2k-Plain protocol.

For the statement (ii), a prover provides a' such that $\mathsf{DivCheck}_T^{(D,\Delta)}(a,a') = \mathsf{true}$ (see Definition 1), or very roughly, $a' = a/2^{\Delta}$. For zero-knowledgeness, a' must be provided as a ciphertext $\hat{\mathfrak{ct}}_{a'}$ with the proof that $\hat{\mathfrak{ct}}_{a'}$ is generated correctly as well. Then, the parties (simultaneously) execute Schnorr-like protocol on $\mathfrak{ct}_{a'}$ with the same challenge matrix W from the above proof on $\mathfrak{ct}_a$ for the statement (i) and the masks $y_l'^i$ such that $\mathsf{DivCheck}_T^{(D,\Delta)}(y_l^i, y_l'^i) = \mathsf{true}$. Then verifiers check if $\mathsf{DivCheck}_T^{(D,\Delta)}(z,z') = \mathsf{true}$ from which one can derive $\mathsf{DivCheck}_T^{(D,\Delta)}(a,a') = \mathsf{true}$ with a rewinding argument (see Appendix B.5).

A caveat here is that we *cannot* use tweaked interpolation packing for $\hat{\mathfrak{ct}}_{a'}$: a factor of 2^T will also arise in the tweaked interpolation packing for $\hat{\mathfrak{ct}}_{a'}$; and we again need ZKPoMK on $\hat{\mathfrak{ct}}_{a'}$ to check that it is encoded correctly.

The key observation for our solution is that $\hat{\mathfrak{ct}}_{a'}$ (in contrasts to $\mathfrak{ct}_a$) does not need to satisfy multiplicative homomorphism (on message space) since it is only used in ZKPoMK for $\mathfrak{ct}_a$, which requires linear homomorphism only. Therefore, we exploit coefficient packing (i.e., each message is encoded as each coefficient of a') for $\hat{\mathfrak{ct}}_{a'}$,[xx] which makes ZKPoPK $\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$ (without any ZKPoMK) suffices to guarantee that $\hat{\mathfrak{ct}}_{a'}$ is correctly encoded. As a bonus, we can use considerably smaller parameters for $\hat{\mathfrak{ct}}_{a'}$, providing almost perfect packing density and resulting better efficiency.

A New Type of Slackness. We now describe the new type of slackness arises from our ZKPoMK Π_{PoMK}. If all parties run $\mathsf{Sampling}$ honestly, then the outputs satisfy the following relation:

$$\begin{aligned}\mathcal{R}_{\mathsf{PoMK}}^{u,\mathsf{Pack}} := \big\{&\text{the same input and witness as } \mathcal{R}_{\mathsf{PoPK}}^{u} :\\ &\text{For all } k \in [u], \quad \mathsf{Pack}_T^{(D,\Delta)}(a_k) = \mathsf{true}\big\}\end{aligned}$$

Note that, however, a verifier *cannot* be guaranteed that $\mathsf{Deg}_T^{(D)}(a_k) = \mathsf{true}$ with our ZKPoMK (for the statement (i) in above). This is because, in the rewinding argument, $\mathsf{Deg}_T^{(D)}((w_{l,k} - \bar{w}_{l,k}) \cdot a_k) = \mathsf{true}$ can occur even with $\mathsf{Deg}_T^{(D)}(a_k) = \mathsf{false}$, since there is a possibility of some non-zero coefficients of a_k becoming zero when multiplied by $(w_{l,k} - \bar{w}_{l,k})$. However, since the difference $w_{l,k} - \bar{w}_{l,k}$ of elements from the challenge space $\mathsf{Chal} = [-2^E + 1, 2^E] \cap \mathbb{Z}$ is at most divisible by 2^E, our ZKPoMK protocol can only guarantee that the given ciphertexts $\{\mathfrak{ct}_k\}_{k\in[u]}$ satisfies the following relation $\mathcal{R}_{\mathsf{PoMK}}^{u,\mathsf{Pack\_sl}}$ which is relaxed from $\mathcal{R}_{\mathsf{PoMK}}^{u,\mathsf{Pack}}$:

$$\begin{aligned}\mathcal{R}_{\mathsf{PoMK}}^{u,\mathsf{Pack\_sl}} := \{&\text{the same input and witness as } \mathcal{R}_{\mathsf{PoPK}}^{u} :\\ &\text{For all } k \in [u], \quad \mathsf{Pack\_sl}_T(a_k) = \mathsf{true}\},\end{aligned}$$

where the predicate $\mathsf{Pack\_sl} : R \to \{\mathsf{true}, \mathsf{false}\}$ is defined as follows (see Section 3.4 for comparison with the original predicates). For $a \in R$, let $(\tilde{a}_i)_{i=1}^{r}$

[xx] This is why we denoted it as $\hat{\mathfrak{ct}}_{a'}$ (not $\mathfrak{ct}_{a'}$) and $\mathsf{DivCheck}$ is defined in such a way.

denote the CRT projections (Eq. (1)) of $\tilde{a} = a \pmod{2^T}$.

- $\mathsf{Pack\_sl}_T^{(D,\Delta,E)}(a) = \mathsf{true} \iff \mathsf{Deg\_sl}_T^{(D,E)}(a) = \mathsf{true} \quad \wedge \quad \mathsf{Div}_T^{(D,\Delta)}(a) = \mathsf{true}.$
- $\mathsf{Deg\_sl}_T^{(D,E)}(a) = \mathsf{true} \iff$ All CRT projections $\tilde{a}_i$ of a satisfy that coefficients at deg $> D$ are divisible by 2^{T-E}.

While the soundness slack S of ZKPoPK appeared also in the previous literature, above *slackness* represented by the predicate $\mathsf{Pack\_sl}$ is a unique feature of our ZKPoMK protocol.

6.2 Managing the Slackness in MPC Preprocessing

In this subsection, we clarify that the new type of slackness which arises in our ZKPoMK (Π_{PoMK}, Appendix B.5) can be managed, i.e., that the guarantee of ZKPoMK is sufficient for the MPC preprocessing phase (Section 2.5).

The idea is to reserve an extra space in the plaintext modulus for the slackness E: for $\mathbb{Z}_{2^k}$-messages, we apply the tweaked interpolation packing (Theorem 1) with $t = E + k + 2\delta$ instead of $t = k + 2\delta$ (Theorem 2)[xxi].

Let $\mathfrak{ct}_a$ be a ciphertext encrypting $a(X)$, which passed the verification of our ZKPoMK parameterized by $D = \left\lfloor \frac{d-1}{2} \right\rfloor$, $\Delta = \delta$, $T = t$, and E. For simplicity, we assume that the plaintext space $\mathbb{Z}_{2^T}[X]/\Phi_M(X)$ does not split. Since $\mathsf{Pack\_sl}_T^{(D,\Delta,E)}(a) = \mathsf{true}$ and $T - E = k + 2\delta$, we can regard $a(X) \pmod{2^{T-E}}$ as a tweaked interpolation packing of $\mathbb{Z}_{2^k}$-messages in $\mathbb{Z}_{2^{k+2\delta}}[X]/\Phi_M(X)$ as before. We have to make sure that, in the Reshare protocol Π_{Reshare}(Section 4.2), $\mathfrak{ct}_{ab} := \mathfrak{ct}_a \boxtimes \mathfrak{ct}_b$ is a valid ciphertext that can be securely decrypted via distributed decryption. Due to the slackness, it is only guaranteed that $a(X) = \tilde{a}(X) + 2^{T-E} \cdot s_a(X)$ where $\deg(\tilde{a}) \leq D$, and similarly for $b(X)$. Note, however, that the terms $s_a(X)$ and $s_b(X)$ arising from the slackness do not affect the messages, since they are multiplied by 2^{T-E}. Therefore, it holds that $ab(j) = a(j) \cdot b(j) \pmod{2^{T-E}}\ \forall j \in [0; D]$, and the most significant 2^E bits (which contains unnecessary information) can be masked. One can also see that similar argument holds for $\mathfrak{ct}_{\alpha a} := \mathfrak{ct}_\alpha \boxtimes \mathfrak{ct}_a$.

[xxi] Our ZKPoMK does not produce the slackness when $E = 0$. Nevertheless, an appropriate $E > 0$ enlarges the challenge space in a cost of only a slight reduction in the packing density.

7 Performance Analysis

In this section, we analyze the performance of our MHz2k with comparison to other works in the literature. We can summarize the improvements by our packing (Section 3) and reshare protocol (Section 4) as follows: (i) Our tweaked interpolation packing achieves near 1/2 packing density, 2.5x compared to 1/5 of Overdrive2k [OSV20], (ii) Our reshare protocol requires only 5 ZKPoPKs which is 1.4x less than 7 ZKPoPKs of Overdrive2k. In total, we can expect that the amortized communication costs of MHz2k-Plain (without the Topgear2k optimization) will show 3.5x improvements from Overdrive2k.

On the other hand, how our ZKPoPK and ZKPoMK (Section 5,6) affect the performance in MHz2k is a bit more involved. In the following subsection we provide a brief cost analysis on our ZKPs.

7.1 Cost Analysis on ZKPoPK and ZKPoMK

The communication cost of ZKPoPK and ZKPoMK *per party* can be estimated by the size of ciphertexts arise in protocols, which dominates the others. Excluding the u input ciphertexts $\mathfrak{ct}_a^i$, using our ZKPoPK and ZKPoMK, there arise additional u ciphertexts $\hat{\mathfrak{ct}}_{a'}^i$, $2v$ masking ciphertexts $\mathfrak{ct}_y^i$, and $2v$ masking ciphertexts $\hat{\mathfrak{ct}}_{y'}^i$. Assuming that $u = 2v$ (as in Topgear [BCS19]) and that the size of $\hat{\mathfrak{ct}}$ is a half of that of $\mathfrak{ct}$, we can conclude that the total cost is roughly $2u \cdot |\mathfrak{ct}|$ in ZKPoPK and ZKPoMK on u input ciphertexts $\mathfrak{ct}_a^i$.

On the other hand, following the approach of Overdrive2k [OSV20], MHz2k can also be initiated with the challenge space of $\{0, 1\}$ without TopGear2k optimization, which we call MHz2k-Plain. In this case, while the challenge space is restricted to $\{0, 1\}$, it requires only *one* Schnorr-like protocol (contrary to *four* in our case) but with $v = 2u - 1$. Hence, the size of masking ciphertexts $\mathfrak{ct}_y^i$ will be roughly $2u \cdot |\mathfrak{ct}|$, and in *amortized* sense, the communication cost does not differ seriously between the case with TopGear2k and without it. The main advantage of our TopGear2k with ZKPoMK (similarly as TopGear [BCS19] to [DPSZ12, KPR18, OSV20]) is that u can be chosen much smaller than that of ZKPoPK of [DPSZ12, KPR18, OSV20] where u is forced to be as large as statistical security parameter at least. This contributes to the substantial reduction of latency and memory requirement (Table 2). Moreover, since there is a trade-off between amortized communication cost versus latency and memory requirement along the choice of u, we can shift the improvements to the amortized communication cost.

7.2 Comparison

For comparison, we present the communication costs of our schemes and previous works. Though we restrict our discussion to secure two-party computation (2PC), similar efficiency improvements occur in any multi-party case. We refer to Table 3

Table 1. Amortized communication (in kbit) of producing triples (2PC)

(k,s)	SPD$\mathbb{Z}_{2^k}$	Mon$\mathbb{Z}_{2^k}$a	Overdrive2k	MHz2k Plain	MHz2k TG2k ($u=2v$)	MHz2k TG2k ($u=4v$)
(32,32)	79.9	59.1	101.8 (72.8)	27.2	26.4	**20.1**
(64,64)	319.5	175.5	171.4 (153.3)	46.2	43.3	**31.9**
(128,64)	557.1	176.6	190.4 (212.2)	56.6	55.0	**40.9**

Table 2. Memory usage (in MB) of producing triples (2PC)

(k,s)	Overdrive2k	MHz2k-Plain	MHz2k-TG2k ($u=2v$)	MHz2k-TG2k ($u=4v$)
(32,32)	272	503	**44**	74
(64,64)	1273	1392	**137**	229
(128,64)	2555	2237	**241**	402

in Appendix A for the detailed description on the parameters for our schemes and others.

In Table 1, we compare the previous works [CDE$^+$18, CDRFG20, OSV20] and ours with respect to (amortized) communication costs for triple generation. For lattice-based HE approaches (Overdrive2k, MHz2k-Plain, and MHz2k-TG2k), the results are computed from the parameters given in Table 3 (Appendix A). For reader's convenience, we also present communication costs of Overdrive2k which are listed in the paper [OSV20] in parentheses.[xxii] Note that Mon$\mathbb{Z}_{2^k}$a only provides secure two-party computation, whereas other protocols can be used for general multi-party computation. MHz2k-Plain shows substantial improvements in communication costs from previous works. In particular, we can check that MHz2k-Plain shows roughly 3.5x improvement from Overdrive2k as we predicted in Section 7.1. As mentioned, applying TopGear2k technique to MHz2k-Plain does not significantly effect the communication costs, if we choose parameters as $u = 2v$. However, increasing the ratio between u and v, we can further reduce the communication costs utilizing more memory (still, less memory than Overdrive2k).

In Table 2, we compare the memory consumption of SHE-based approaches, which are computed as $(u+v)\cdot 2\varphi(M)\log q$ with parameters in Table 3. Applying TopGear2k optimization, we can significantly reduce the memory consumption. With Table 1, we can also check the trade-off between the amortized communication costs and the memory utilization along the choice of u.

[xxii] Due to the lack of information, it was hard to reproduce the communication costs of Overdrive2k. In particular, their parameters does not seem to achieve 128 bits security if we consider key-switching modulus which is not noted.

Acknowledgement

In addition to the appreciation on constructive comments of the reviewers of Crypto 2021, the authors express gratitude to the reviewers of Eurocrypt 2021 who provided invaluable comments to improve the earlier version of this paper. The authors also thank Duhyeong Kim, Jiseung Kim, and Yongsoo Song for helpful discussions. This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2020-0-00840, Development and Library Implementation of Fully Homomorphic Machine Learning Algorithms supporting Neural Network Learning over Encrypted Data)

References

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

BCD+09. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

BCK+14. Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 551–572. Springer, 2014.

BCS19. Carsten Baum, Daniele Cozzo, and Nigel P Smart. Using topgear in overdrive: A more efficient zkpok for spdz. In *International Conference on Selected Areas in Cryptography*, pages 274–302. Springer, 2019.

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–188. Springer, 2011.

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

BGV14. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

BJSV15. Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In *International conference on financial cryptography and data security*, pages 227–234. Springer, 2015.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

CCXY18. Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure mpc revisited. In *Annual International Cryptology Conference*, pages 395–426. Springer, 2018.

CD09. Ronald Cramer and Ivan Damgård. On the amortized complexity of zero-knowledge protocols. In *Annual International Cryptology Conference*, pages 177–191. Springer, 2009.

CDE$^+$18. Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spd$\mathbb{Z}_{2^k}$: Efficient mpc mod 2^k for dishonest majority. In *Annual International Cryptology Conference*, pages 769–798. Springer, 2018.

CDRFG20. Dario Catalano, Mario Di Raimondo, Dario Fiore, and Irene Giacomelli. Mon$\mathbb{Z}_{2^k}$a: Fast maliciously secure two party computation on $\mathbb{Z}_{2^k}$. In *IACR International Conference on Public-Key Cryptography*, pages 357–386. Springer, 2020.

CK89. B Chor and E Kushilevitz. A zero-one law for boolean privacy. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 62–72, 1989.

CKKL21. Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, and Keewoo Lee. On the scaled inverse of $(x^i - x^j)$ modulo cyclotomic polynomial of the form $\phi_{p^s}(x)$ or $\phi_{p^s q^t}(x)$. *arXiv preprint arXiv:2106.01742*, 2021.

CL21. Jung Hee Cheon and Keewoo Lee. Limits of polynomial packings for $\mathbb{Z}_{p^k}$ and $\mathbb{F}_{p^k}$. Cryptology ePrint Archive, Report 2021/1033, 2021.

DEF$^+$19. Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure mpc over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1102–1120. IEEE, 2019.

DKL$^+$13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

GBOW88. S Goldwasser, M Ben-Or, and A Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proc. of the 20th STOC*, pages 1–10, 1988.

GHS12a. Craig Gentry, Shai Halevi, and Nigel P Smart. Better bootstrapping in fully homomorphic encryption. In *International Workshop on Public Key Cryptography*, pages 1–16. Springer, 2012.

GHS12b. Craig Gentry, Shai Halevi, and Nigel P Smart. Homomorphic evaluation of the aes circuit. In *Annual Cryptology Conference*, pages 850–867. Springer, 2012.

HS15. Shai Halevi and Victor Shoup. Bootstrapping for helib. In *Annual International conference on the theory and applications of cryptographic techniques*, pages 641–670. Springer, 2015.

JL13. Marc Joye and Benoît Libert. Efficient cryptosystems from 2 k-th power residue symbols. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 76–92. Springer, 2013.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.

OSV20. Emmanuela Orsini, Nigel P Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure mpc over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In *Cryptographers' Track at the RSA Conference*, pages 254–283. Springer, 2020.

SV14. Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71(1):57–81, 2014.

Table 3. Parameter sets for 2PC

Protocol	M	$\log q$	$\log q_0$	t	k	s	sec	κ	u	v	E
Overdrive2k	23311	290	190	64	32	32	26	.177	54	107	-
MHz2k-Plain	42799	320	220	80				.419	55	109	-
MHz2k-TG2k	42799	330	220	80				.419	8	4	7
$(u = 2v)$	20161	200	200	64				.999			
MHz2k-TG2k	42799	330	220	80				.419	16	4	7
$(u = 4v)$	20161	210	210	64				.999			
Overdrive2k	38737	500	350	128	64	64	57	.180	88	175	-
MHz2k-Plain	42799	530	380	144				.466	88	175	-
MHz2k-TG2k	42799	520	380	144				.466	16	8	7
$(u = 2v)$	20161	330	330	128				.999			
MHz2k-TG2k	42799	530	380	144				.466	32	8	7
$(u = 4v)$	20161	330	330	128				.999			
Overdrive2k	61681	630	480	196	128	64	57	.200	88	175	-
MHz2k-Plain	60787	660	510	208				.462	88	175	-
MHz2k-TG2k	60787	650	510	208				.462	16	8	7
$(u = 2v)$	27281	400	400	192				.999			
MHz2k-TG2k	60787	660	510	208				.462	32	8	7
$(u = 4v)$	27281	400	400	192				.999			

A Parameters

In Table 3, concrete parameter sets for our two schemes MHz2k-Plain and MHz2k-TG2k are presented. For MHz2k-TG2k, two parameter sets are presented with different ratio between u and v. For each parameter set of MHz2k-TG2k, two sets of SHE parameters are given, and the below one is for $\hat{\mathfrak{ct}}_{a'}$. In the column κ, packing densities are provided.

For parameter choice, we basically follow the analysis of [GHS12b] and [DKL$^+$13]. We chose the ciphertext modulus q to be a product of moduli q_0 and q_1. We use key-switching modulus Q, which is the largest modulus used in our SHE scheme, as $Q = q \cdot q_0$. We chose parameters with more than 128-bit computational security according to LWE Estimator [APS15]. The parameter M's are chosen from integers with two or less prime factors with $d < 40$ to optimize packing densities (Section 5.4). We can also choose smaller M's, if we slightly sacrifice packing densities. The column sec is the statistical security of MAC protocols we are using in our online phase (from SPD$\mathbb{Z}_{2^k}$ [CDE$^+$18, Theorem 1]). We carefully chose parameters so that all parts of protocols satisfy sec-bit statistical security.

For a fair comparison, in Table 3, we also present parameter sets of Overdrive2k which are computed by our analysis.[xxiii] For Overdrive2k, the parameter M's are selected as the smallest prime providing 128-bit computational security among recommended parameters listed in the paper [OSV20].

[xxiii] Due to the lack of information, it was hard to reproduce the communication costs of Overdrive2k. In particular, their parameters does not seem to achieve 128 bits security if we consider key-switching modulus which is not noted.

B Formal Descriptions

In this section, we give deferred more formal descriptions of MHz2k preprocessing phase. While the functionality of our preprocessing phase (Π_{wPrep} in Fig. 8) is the same as that of previous works, the protocol itself and constituting functionalities are slightly different due to the particularity of our packing method. We present our preprocessing phase focusing on these differences.

B.1 Security Model and Notations

We prove that our protocols are secure in the universal composable (UC) framework [Can01]. Specifically, we consider static (i.e., corruption takes place before the protocol starts) malicious adversaries corrupting upto $n-1$ parties among n participants $(P_1, P_2, \ldots, P_n)$. We say that a protocol Π *implements* a functionality $\mathcal{F}$ with computational (resp. statistical) security parameter λ (resp. sec) if the advantage of any environment $\mathcal{Z}$ in distinguishing the ideal and real executions is $O(2^{-\lambda})$ (resp. $O(2^{-\mathsf{sec}})$).

As other line of works, we rely on the registered public-key model. In particular, we assume that there is a functionality $\mathcal{F}_{\mathsf{KeyGen}}$ (Appendix B.2) that generates and distributes secret key shares for given encryption scheme. We also assume a coin-tossing functionality $\mathcal{F}_{\mathsf{Rand}}$ (Appendix B.2) which outputs, to all parties, a common uniform random element.

For consistency, we use boldface to denote vectors, e.g. $\boldsymbol{m} \in \mathbb{Z}_{2^k}^{\nu}$, and regular typeface to denote polynomials, e.g. $m \in \mathbb{Z}_{2^T}[X]/\Phi_M(X)$. We remark that the functionalities ($\mathcal{F}_{\mathsf{KeyGenDec}}$, $\mathcal{F}_{\mathsf{Prep}}$) will be described according to vectors of messages, while the protocols will be more relevant with polynomials. The simulators will manage these discrepancies based on the properties of our packing method.

B.2 Basic Functionalities - $\mathcal{F}_{\mathsf{Rand}}$, $\mathcal{F}_{\mathsf{KeyGen}}$

As other line of works in MPC [DPSZ12, KPR18, OSV20], we assume that the following basic functionalities $\mathcal{F}_{\mathsf{KeyGen}}$ and $\mathcal{F}_{\mathsf{Rand}}$ can be efficiently implemented.

– The functionality $\mathcal{F}_{\mathsf{Rand}}$:
  On input $(\text{Rand}, \mathcal{D})$ from all parties, randomly sample $r \leftarrow \mathcal{D}$, then output r to all parties.

– The functionality $\mathcal{F}_{\mathsf{KeyGen}}$:

  1. On receiving (Init) form all honest parties, run $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{BGV.KeyGen}(\mathsf{pp}_\lambda)$. Send $\mathfrak{pk}$ to the adversary $\mathcal{A}$.
  2. Receive shares $\{[\mathfrak{sk}]_i\}_{i \in \mathcal{A}}$ from $\mathcal{A}$.
  3. Construct other shares $\{[\mathfrak{sk}]_i\}_{i \notin \mathcal{A}}$ for the honest parties such that $\mathfrak{sk} = \sum_{i \in [n]} \mathfrak{sk}_i$, where n is the number of all participating parties.
  4. Send $\mathfrak{pk}$ to all parties, and $[\mathfrak{sk}]_i$ to each party P_i.

Protocol Π_{DistrDec}

PARAMETERS:
- B_{DDec}: a bound on the coefficients of the mask values.
- B_{noise}: a bound on the noise of input ciphertext.
- n: the number of participating parties P_i.

Init: Each party P_i calls $\mathcal{F}_{\mathsf{KeyGen}}$ (Appendix B.2) receiving $(\mathfrak{pk}, [\mathfrak{sk}]_i)$.

D1: On input parameter $\mathsf{pp} = (D, \Delta, T)$ and a ciphertext $\mathfrak{ct}_{\boldsymbol{m}}^{(0)} = (c_0, c_1)$ satisfying that $\mathsf{Div}_T^{(D,\Delta)}(\mathsf{Dec}(\mathfrak{ct}_{\boldsymbol{m}}^{(0)}, \mathfrak{sk})) = \mathsf{true}$, parties do as follows.

1. Each P_i computes $w_i = \begin{cases} c_0 - [\mathfrak{sk}]_1 \cdot c_1 & \text{if } i = 1 \\ -[\mathfrak{sk}]_i \cdot c_1 & \text{if } i \neq 1 \end{cases}$.
2. Set $\mathsf{P} \leftarrow \mathsf{Div}_T^{(D,\Delta)}$. Each P_i samples a mask $r_i \leftarrow \mathcal{U}_{\mathsf{P}}(B_{\mathsf{DDec}})$ (Definition 2), then broadcasts $v_i = w_i + r_i \pmod{q_0}$.
3. For $v = \sum_i v_i \pmod{q_0}$, parties check if $||v||_\infty < B_{\mathsf{noise}} + n \cdot B_{\mathsf{DDec}}$ and $\mathsf{P}(v) = \mathsf{true}$. If not, abort.
4. Each P_i computes $u_i = \begin{cases} (\sum_i v_i \pmod{q_0}) - r_1 \pmod{2^T} & \text{if } i = 1 \\ -r_i \pmod{2^T} & \text{if } i \neq 1 \end{cases}$.
5. Each P_i retrieves $[\boldsymbol{m}]_i$ from u_i by regarding u_i as a Tweaked Interpolation Packing without the condition (i) (Theorem 1), with $\delta = \Delta$, $\lfloor \frac{d-1}{2} \rfloor = D$, and $t = T$.

Fig. 5. Protocol for distributed decryption

B.3 Distributed Decryption Protocol

We first describe a distributed decryption protocol Π_{DistrDec} in Fig. 5. Given that each party is distributed with a share $\mathfrak{sk}_i$ of the secret key $\mathfrak{sk}$ for a ciphertext having a message $\boldsymbol{m} \in \mathbb{Z}_{2^k}^{\nu}$ encoded with our packing method, the protocol distributes each share $[\boldsymbol{m}]_i$ of the underlying message to each party.

Similarly as other usual distributed decryption protocols in the literature (e.g., [DPSZ12, OSV20]), each party adds a mask r_i to their partial decryption w_i to prevent leakage of information. The crucial difference is that each mask r_i is sampled from the distribution $\mathcal{U}_{\mathsf{P}}(B_{\mathsf{DDec}})$ with $\mathsf{P} = \mathsf{Div}_T^{(D,\Delta)}$ (Definition 2). Not only the mask statistically prevents the information leakage (by Lemma 3), since $\mathsf{Div}_T^{(D,T)}(r_i) = \mathsf{true}$, it also *preserves* the packing structure[xxiv] so that each party can retrieve the share of message.

Theorem 6 (Distributed Decryption). *On a cyclotomic ring $\mathbb{Z}[X]/\Phi_M(X)$, the protocol Π_{DistrDec} in Fig. 5 implements the functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$ in Fig. 6 against any static, active adversary corrupting up to $n-1$ parties in the $\mathcal{F}_{\mathsf{KeyGen}}$-hybrid model with statistical security $\varphi(M) \cdot 2^{-sec}$ if $B_{\mathsf{DDec}} > 2^{sec+1} \cdot \max((B_{\mathsf{noise}} + 2^T), n \cdot 2^T)$, and $B_{\mathsf{noise}} + n \cdot B_{\mathsf{DDec}} < q_0/2$.*

[xxiv] If r_i is sampled from a random polynomial without any restriction, $\mathsf{Div}_T^{(D,\Delta)}(u_i) = \mathsf{false}$ (with high probability) and u_i can not be seen as a tweaked interpolation packing.

Functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$

PARAMETERS:
- $\mathcal{A}$: the set of corrupted parties.
- B_{DDec}, B_{noise}, n: the same as those of Π_{DistrDec} (Fig. 5).

Init: Run $(\mathfrak{pk}, \mathfrak{sk}) \leftarrow \mathsf{BGV.KeyGen}(\mathsf{pp}_\lambda)$ then store $\mathfrak{sk}$.
1. Receive shares $\{\mathfrak{sk}_i\}_{i\in\mathcal{A}}$ of the secret key from the adversary.
2. Randomly sample shares $\{\mathfrak{sk}_j\}_{j\notin\mathcal{A}}$ of the secret key for the honest parties such that $\mathfrak{sk} = \sum_{i\in[n]} \mathfrak{sk}_i$.
3. Send $\mathfrak{pk}$ to all parties and $\mathfrak{sk}_j$ to each honest party P_j.

D1: On receiving input (same as those of Π_{DistrDec}.**D1** (Fig. 5)) from all parties, performs the following steps.
1. Execute $\mathsf{Dec}(\mathfrak{ct}^{(0)}_{\boldsymbol{m}}, \mathfrak{sk})$ then retrieve $\boldsymbol{m}$ regarding it as a tweaked interpolation packing (possible since $\mathsf{Div}^{(D,\Delta)}_T(\mathsf{Dec}(\mathfrak{ct}^{(0)}_{\boldsymbol{m}}, \mathfrak{sk})) = \mathsf{true}$).
2. Wait for the adversary to input either abort or $\boldsymbol{\eta}$. If abort, then forward abort to the honest parties and halt.
3. Otherwise, uniformly sample the honest shares $[\boldsymbol{m}]_i, i \notin \mathcal{A}$ such that $\sum_{i\notin\mathcal{A}}[\boldsymbol{m}]_i = \boldsymbol{m} + \boldsymbol{\eta}$. Send each share to corresponding $P_i, i \notin \mathcal{A}$.

Fig. 6. Functionality for distributed decryption

Proof. The proof resembles those of [DPSZ12, Theorem 3] and takes into account the specificity of our packing method.

Correctness. In command **D1** of Π_{DistrDec}, if all parties are honest, each P_i can retrieve $[\boldsymbol{m}]_i$ since $\mathsf{Div}^{(D,\Delta)}_T(r_i) = \mathsf{true}$. Note that the decryption can be done correctly since $\sum_i ||v_i||_\infty \leq (B_{\mathsf{noise}} + n \cdot B_{\mathsf{DDec}}) \leq q_0/2$ where B_{noise} is the noise bound of the input $\mathfrak{ct}_{\boldsymbol{m}}$.

Security. Under the UC framework, we need to show the indistinguishability between the ideal execution and real execution to an environment $\mathcal{Z}$, which can be proved similarly as in [DPSZ12]. In ideal execution, the corrupted parties interact with, instead of honest parties, the simulator $\mathcal{S}_{\mathsf{DistrDec}}$ (Fig. 7) having access to the functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$ (Fig. 6).

The essence is that the mask $r_i \leftarrow \mathcal{U}_{\mathsf{P}}(B_{\mathsf{DDec}})$ with $\mathsf{P} = \mathsf{Div}^{(D,\Delta)}_T$ in the command **D1** statistically hides the output of each party by Lemma 3 (Section 3.5). More precisely, it suffices to prove the following:

– The output $\{\tilde{v}_i\}_{i\notin\mathcal{A}}$ from $\mathcal{S}_{\mathsf{DistrDec}}$ to the corrupted parties $\mathcal{A}$ and the output for honest parties from functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$, are indistinguishable from those outputs from the honest parties to $\mathcal{A}$ in the real execution.

To this end, we introduce following hybrid machine then show that its output is statistically indistinguishable from the output of honest parties (in the real

Simulator $\mathcal{S}_{\mathsf{DistrDec}}$

Init: From corrupted parties $\mathcal{A}$ obtain $\{\mathfrak{sk}_i\}_{i\in\mathcal{A}}$. Then, passing these to the functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$, obtain $\mathfrak{pk}$. Set random $\{\mathfrak{sk}_i\}_{i\notin\mathcal{A}}$ such that $\sum_{i\in[n]}\mathfrak{sk}_i = \mathbf{0}$, a zero vector. Send $\mathfrak{pk}$ to $\mathcal{A}$.

D1: On input $\mathsf{pp} = (D, \Delta, T)$ and $\mathfrak{ct}^{(0)}_{\boldsymbol{m}}$,

1. Compute all $\tilde{w}_i$ and $\tilde{v}_i$ values the same as the protocol Π_{DistrDec}.**D1** (Fig. 5), but with $\mathfrak{sk}_i$'s at **Init**, except for one $j \notin \mathcal{A}$.
2. For above j, set $\tilde{v}_j = -\sum_{i\neq j}\tilde{w}_i + \tilde{r}_j \pmod{q_0}$ where $\tilde{r}_j \leftarrow \mathcal{U}_{\mathsf{P}}(B_{\mathsf{DDec}})$ with $\mathsf{P} = \mathsf{Div}_T^{(D,\Delta)}$.
3. Send $\{\tilde{v}_i\}_{i\notin\mathcal{A}}$ to $\mathcal{A}$ and receive $\{v_i^*\}_{i\in\mathcal{A}}$ from $\mathcal{A}$.
4. On $\sum_{i\in\mathcal{A}} v_i^* + \sum_{i\notin\mathcal{A}}\tilde{v}_i$, if the check (3. in Π_{DistrDec}.**D1**) does not pass, send abort to the functionality.
5. Otherwise, if P_1 is honest: retrieve $\boldsymbol{\eta}$ from $\sum_{i\in\mathcal{A}}(v_i^* - \tilde{w}_i)$ regarding it as a tweaked interpolation packing.
   If P_1 is corrupt, retrieve $\boldsymbol{\eta}$ from $\sum_{i\notin\mathcal{A}}(-\tilde{r}_i)$ regarding it as a tweaked interpolation packing.
6. Send $\boldsymbol{\eta}$ to the functionality.

Fig. 7. Simulator for $\mathcal{F}_{\mathsf{KeyGenDec}}$

protocol) and from outputs of $\mathcal{S}_{\mathsf{DistrDec}}$ and $\mathcal{F}_{\mathsf{KeyGenDec}}$ (in the ideal execution).

Hybrid: outputs $\{\tilde{v}_i\}_{i\notin\mathcal{A}}$ as $\mathcal{S}_{\mathsf{DistrDec}}$ then outputs $\{\widetilde{[\boldsymbol{m}]_i}\}_{i\notin\mathcal{A}}$ as Π_{DistrDec} with $\tilde{r}_i$'s of $\mathcal{S}_{\mathsf{DistrDec}}$, except that for the j chosen in $\mathcal{S}_{\mathsf{DistrDec}}$, outputs $[\boldsymbol{m}] + \widetilde{[\boldsymbol{m}]_j}$.
(Instead of $\{\widetilde{[\boldsymbol{m}]_i}\}_{i\notin\mathcal{A}}$, output abort if the check 3. does not pass.)

For indistinguishability from real execution, note that $\mathcal{S}_{\mathsf{DistrDec}}$ computes each $\tilde{v}_i$ exactly the same as honest parties from the real execution, except for the party P_j. Let $\tilde{v}_j$ and v_j be the output of simulator and P_j (from real execution), respectively. Then,

$$\tilde{v}_j = -\sum_{i\neq j}\tilde{w}_i + \tilde{r}_j,$$
$$v_j = w_j + r_j = c_0 - \mathfrak{sk}\cdot c_1 - \sum_{i\neq j} w_i + r_j,$$

and since the size of decryption $c_0 - \mathfrak{sk}\cdot c_1$ is bounded by $B_{\mathsf{noise}} + 2^T$, Lemma 3 (Section 3.5) implies that the statistical distance between v_j and $\tilde{v}_j$ is less than $\phi(M)\cdot 2^{-sec-1}$ (the distribution of w_i and $\tilde{w}_i$ is identical for $i \neq j$). We also mention that the distribution of output $\widetilde{[\boldsymbol{m}]_i}$'s (and $[\boldsymbol{m}] + \widetilde{[\boldsymbol{m}]_j}$) is identical to that of $[\boldsymbol{m}]_i$'s from real execution: it is obvious for $i \neq j$ while for $i = j$, both outputs are added by $[\boldsymbol{m}]$ after being retrieved from $-\tilde{v}_j - \sum_{i\neq j}\tilde{w}_i$ and $-v_j - \sum_{i\neq j} w_i$, respectively.

For indistinguishability from ideal execution, we first assume that P_1 is honest and that $j = 1$ in $\mathcal{S}_{\mathsf{DistrDec}}$ without loss of generality. The only difference between hybrid machine and ideal execution is that the output $\{\widetilde{[\boldsymbol{m}]_i}\}_{i \notin \mathcal{A}}$ is sampled randomly subject to the condition $\sum_{i \notin \mathcal{A}} [\boldsymbol{m}]_i = \boldsymbol{m} + \boldsymbol{\eta}$ in the latter one. We can see that both output distribution is statistically indistinguishable as follows. For $i \neq 1$, rewrite $\mathcal{S}_{\mathsf{DistrDec}}$'s output with $\tilde{v}_i = \tilde{w}'_i + \tilde{r}'_i$ where $\tilde{w}'_i = \tilde{w}_i + \tilde{r}_i - \tilde{r}'_i$ and $\tilde{r}'_i$ is sampled from the same distribution as $\tilde{r}_i$ and one can retrieve $\widetilde{[\boldsymbol{m}]_i}$ from $-\tilde{r}'_i$. Then, for $i = 1$, $\mathcal{S}_{\mathsf{DistrDec}}$'s output $\tilde{v}_1 = -\sum_{i \neq 1, i \notin \mathcal{A}} \tilde{w}'_i - \sum_{i \in \mathcal{A}} \tilde{w}_i + \sum_{i \neq 1, i \notin \mathcal{A}} (\tilde{r}_i - \tilde{r}'_i) + \tilde{r}_1$ is statistically close to that of hybrid machine with the distance bounded by $\phi(M) \cdot 2^{-sec-1}$ since $|| \sum_{i \neq 1, i \notin \mathcal{A}} (\tilde{r}_i - \tilde{r}'_i) ||_\infty$ is bounded by $n \cdot 2^T$ (again from Lemma 3 with $\tilde{r}_1 \leftarrow \mathcal{U}_{\mathsf{P}}(B_{\mathsf{DDec}})$). Finally, let $\widetilde{[\boldsymbol{m}]_1}$ and $[\boldsymbol{m}]_1$ be the output of P_1 in the ideal execution and hybrid machine, respectively, and $\tilde{v}_i$ and v_i^* respectively are generated from the $\mathcal{S}_{\mathsf{DistrDec}}$ and corrupted P_i's. Then,

$$\widetilde{[\boldsymbol{m}]_1} = \boldsymbol{m} + \boldsymbol{\eta} - \sum_{i \neq 1, i \notin \mathcal{A}} \widetilde{[\boldsymbol{m}]_i} \text{ where } \boldsymbol{\eta} \text{ is retrieved from } \sum_{i \in \mathcal{A}} (v_i^* - \tilde{w}_i),$$

$$[\boldsymbol{m}]_1 \text{ is retrieved from } \sum_{i \notin \mathcal{A}} \tilde{v}_i + \sum_{i \in \mathcal{A}} v_i^* - \tilde{r}_1.$$

Substituting $\sum_{i \in \mathcal{A}} v_i^*$ by $\sum_{i \in \mathcal{A}} \tilde{w}_i + \sum_{i \in \mathcal{A}} (v_i^* - \tilde{w}_i)$ in the second equation, and recalling that $\sum_{i \notin \mathcal{A}} \tilde{v}_i + \sum_{i \in \mathcal{A}} \tilde{w}_i = \sum_{i \notin \mathcal{A}} r_i + \mathsf{Dec}(\mathfrak{ct}_{\boldsymbol{m}}^{(0)}, \mathfrak{sk})$, we can see that $\widetilde{[\boldsymbol{m}]_1}$ and $[\boldsymbol{m}]_1$ is indisginguishable (note that both $[\boldsymbol{m}]_i$ and $-r_i$ are sampled uniformly for $i \neq 1$). Here, with overwhelming probability, the check (3. in $\Pi_{\mathsf{DistrDec}}.\mathbf{D1}$) on $\sum_{i \in \mathcal{A}} v_i^* + \sum_{i \notin \mathcal{A}} \tilde{v}_i$ by the simulator (and hybrid machine) passes if and only if the check passes on $v = \sum_{i \in \mathcal{A}} v_i^* + \sum_{i \notin \mathcal{A}} v_i$ in the real execution. If the former check passes, the simulator can retrieve $\boldsymbol{\eta}$ from $\sum_{i \in \mathcal{A}} (v_i^* - \tilde{w}_i)$ which satisfies $\mathsf{P}(\cdot) = \mathsf{true}$, since the difference $\sum_{i \notin \mathcal{A}} \tilde{v}_i + \sum_{i \in \mathcal{A}} \tilde{w}_i$ (from the former one) passes the check with overwhelming probability.

The case with corrupted P_1 can be proved similarly as above with the same hybrid machine, and the claim follows.

□

B.4 Preprocessing Protocol

Now we construct our preprocessing phase using the functionality $\mathcal{F}_{\mathsf{KeyGenDec}}$ (Fig. 6) from the previous section and the protocol Π_{Reshare} (Fig. 3, Section 4). Our preprocessing protocol Π_{wPrep} (Fig. 8) implements the same (weak) preprocessing functionality $\mathcal{F}_{\mathsf{wPrep}}$ given in [OSV20] and resembles the corresponding protocol of [OSV20] in high level. A slight but noteworthy difference is that we use a constant encoding for the ciphertext $\mathfrak{ct}_\alpha$ for a global mac key α (Section 4). Though our Π_{wPrep} describes triple generation only, we can also implement **Input**, **Square**, and **Bit** commands as those of [OSV20]. The whole preprocessing phase can be obtained by combining this $\mathcal{F}_{\mathsf{wPrep}}$ with the standard *sacrifice* step (See e.g. [OSV20]).

For the proof, we follow the high level idea of one given in [DPSZ12].

Theorem 7 (Preprocessing). *On a cyclotomic ring $\mathbb{Z}[X]/\Phi_M(X)$, the protocol Π_{wPrep} in Fig. 8 implements the functionality $\mathcal{F}_{\mathsf{wPrep}}$ [OSV20] with computational security against any static, active adversary corrupting up to $n-1$ parties in the ($\mathcal{F}_{\mathsf{KeyGen}}$, $\mathcal{F}_{\mathsf{Rand}}$)-hybrid model.*

Proof. Simiarly to [DPSZ12], we will construct an algorithm $\mathcal{B}$ that can distinguish a valid public key $\mathfrak{pk}$ and a meanigless one $\mathfrak{pk}^*$ with non-negligible advantage $\epsilon/2$ using an environment that can distinguish real and ideal execution with non-negligible advantage ϵ. Then, given that $\mathfrak{pk}$ and $\mathfrak{pk}^*$ is computationally indistinguishable, there is no coputationally bounded environment that can distinguish those two executions.

The algorithm $\mathcal{B}$ works as follows, given a valid $\mathfrak{pk}$ or a meaningless $\mathfrak{pk}^*$ from a challenger. It first randomly decides to simulate idea or real execution. For ideal execution, it imitates the simulator $\mathcal{S}_{\mathsf{wPrep}}$ (Fig. 11) and works almost the same as it except the following differences: (i) It uses $\mathfrak{pk}$ or $\mathfrak{pk}^*$ given from the challenger and does not have access to the $\mathfrak{sk}$. Therefore, when decryption is required, it extracts the underlying message using the extractor of ZKPoPK (and ZKPoMK) and rewinding the environment (who controlls the adversary). Note that every ciphertext that requires decryption for $\mathcal{S}_{\mathsf{wPrep}}$ is performed by ZKPoPK (and ZKPoMK) and $\mathcal{B}$ can imitate $\mathcal{S}_{\mathsf{wPrep}}$ with this strategy. (ii) When generating proofs for ZKPoPK (and ZKPoMK), $\mathcal{B}$ uses the ZK simulator instead of generating it with its underlying messages of ciphertexts it has generated.[xxv] (iii) When emulating $\mathcal{F}_{\mathsf{KeyGenDec}}$.**D1**, algorithm $\mathcal{B}$ uses extracted messages and $\mathfrak{sk}_j$'s it has generated at the $\mathcal{S}_{\mathsf{DistrDec}}$.**Init** phase, which is sufficient (see $\mathcal{S}_{\mathsf{DistrDec}}$(Fig. 7) and $\mathcal{F}_{\mathsf{KeyGenDec}}$(Fig. 6)).

For real execution, $\mathcal{B}$ also works the same as abvoe $\mathcal{B}$, except that it outputs the final shares based on what it has chosen to generate the simulated output (instead of the one from functionality $\mathcal{F}_{\mathsf{wPrep}}$).

Now, we can see that when a valid $\mathfrak{pk}$ is given to $\mathcal{B}$, its output is statistically indistinguishable to that of the ideal or real execution, respectively, according to

[xxv] This is required to show the indistinguishability of $\mathcal{B}$'s output in ideal and real executions when the given $\mathfrak{pk}^*$ is a meaningless one.

the decision $\mathcal{B}$ had made: the statisticall Zero-Knowledgeness and the soundness of our ZKPoPK (and ZKPoMK) make the differences between $\mathcal{B}$ and $\mathcal{S}_{\mathsf{wPrep}}$ statistically indistinguishable; similar reasoning (and Theorem 6) gives that the difference between $\mathcal{B}$ and real execution (Π_{Prep}) is statistically indistinguishable also.

On the other hand, if meaningless $\mathfrak{pk}^*$ is given to $\mathcal{B}$, its outputs for ideal and real execution is statistically indistinguishable. It follows from the fact[xxvi] that ciphertexts encrypted with a meaningless key are statistically indistinguishable from encryptions of zero. Note that the only difference between $\mathcal{B}$ for real and ideal execution is that $\mathcal{B}$ reuses the internal input (e.g., $[\alpha]_i, [\boldsymbol{m}]_i, [\boldsymbol{a}]_i, ...$) to output the final share in the former case. Still, since other outputs of $\mathcal{B}$ such as zk proofs and ciphertexts (which may leak some information on those internal input) do not leak any information to the adversary (the proofs are simulated and ciphertexts are statistically indistinguishable from those of zero), both executions are statistically indistinguishable.

Now if there is an environment that can distinguish real and ideal execution, $\mathcal{B}$ can distinguish the valid $\mathfrak{pk}$ and meaningless $\mathfrak{pk}^*$ as follows: with given public key, $\mathcal{B}$ randomly decides to output real or ideal execution to the environment then see the decision of the environement. If the environment gives a correct guess with non-negligible advantage, the given public key is $\mathfrak{pk}$. If it is not, the given public keye is $\mathfrak{pk}^*$.

We finally remark that our ZKPoPK and ZKPoMK ensures that the adversaries can only modify its messages (possibly, a polynomials in general) of the ciphertexts in a way that the simulator $\mathcal{S}_{\mathsf{wPrep}}$ can extract the appropriate vectors (e.g., δ_m, δ_γ) from it. The point is that, the other part of the underlying messages of adversaries' input ciphertexts — that is not captured by $\mathcal{S}_{\mathsf{wPrep}}$'s extracted vectors — can not affect the final output (vectors of shares) of honest parties, e.g., see Section 6.2. $\square$

[xxvi] We need to assume that our encryption scheme satisfies this propoerty as [DPSZ12] does.

Protocol $\mathit{\Pi}_{\mathsf{wPrep}}$

PARAMETERS:

- A cyclotomic ring $\mathbb{Z}[X]/\Phi_M(X)$.
- n: the number of participating parties P_i.
- $\rho = r \times (\bar{d}+1)$: the number of authenticated triples we produce for each call of the following commands. Here, r and $\bar{d}$ are from the CRT isomorphism $\mathbb{Z}_{2^t}[X]/\Phi_M(X) \cong \prod_{i=1}^{r} \mathbb{Z}_{2^t}[X]/(F_i(X))$ with $\bar{d} = \lfloor \frac{d-1}{2} \rfloor$ where $d = \deg F_i(X)$ (Section 2.3).
- Set $t = k+\delta+\delta$, where $k = \tilde{k}+s$ and $\tilde{k} = 32, 64,$ or 128 (the bit-size of messages), s is the statistical security parameter of MAC (in the online-phase), and $\delta \geq \nu_2(\bar{d}!)$ (the condition for tweaked interpolation packing, Theorem 1).
- We work over the plaintext modulus $\mathbb{Z}_{2^t}$ to get input masks and triples over $\mathbb{Z}_{2^k}$.

Init: distributes shares of secret key and MAC key.

1. Parites call $\mathcal{F}_{\mathsf{KeyGenDec}}$.**Init** to obtain $\mathfrak{pk}$ and $\mathfrak{sk}_i$ for each P_i.
2. Each P_i samples a random $[\alpha]_i \leftarrow \mathbb{Z}_{2^k}$ then generates and broadcasts a level-one ciphertext $\mathfrak{ct}^{(1)}_{[\alpha]_i}$ having the constant $[\alpha]_i$ as a message polynomial.
3. All parties compute $\mathfrak{ct}^{(1)}_{\alpha} = \sum_{i\in[n]} \mathfrak{ct}^{(1)}_{[\alpha]_i}$, then perform ZKPoPK and ZKPoMK (Section 6 or Fig. 12) on it with parameter $\mathsf{pp} = (D = 0, \Delta = 0, T = t)$.

Input, P_I: produces ρ random masks for inputs from P_I.

1. P_I samples a random $\boldsymbol{m} \in \mathbb{Z}_{2^k}$, generates shares $\{[\boldsymbol{m}]_i\}_{i\in[n]}$ of $\boldsymbol{m}$, then sends each to designated party P_i. Hence each P_i obtains $[\boldsymbol{m}]_i$ (and P_I obatins $\boldsymbol{m}$ additionally).
2. Each P_i generates and broadcasts $\mathfrak{ct}^{(1)}_{[\boldsymbol{m}]_i}$ by encoding $[\boldsymbol{m}]_i$ as a tweaked interpolation packing with $D = \bar{d}, \Delta = \delta, T = t$ (Section 3.4, Remark 3).
3. All parties compute $\mathfrak{ct}^{(1)}_{\boldsymbol{m}} = \sum_{i\in[n]} \mathfrak{ct}^{(1)}_{[\boldsymbol{m}]_i}$, then performs ZKPoPK and ZKPoMK on it with parameter $(D = \bar{d}, \Delta = \delta, T = t)$.
4. Parties run $\mathit{\Pi}_{\mathsf{Auth}}$ (Fig. 9) on input $\mathfrak{ct}^{(1)}_{\boldsymbol{m}}$ with $\mathsf{pp} = (\bar{d}, \delta, t)$ to obtain $[\boldsymbol{\gamma_m}]_i$.

Triple: produces ρ authenticated triples.

1. Each P_i samples a random $[\boldsymbol{a}]_i \in \mathbb{Z}^{\rho}_{2^k}$, generates and broadcasts $\mathfrak{ct}^{(1)}_{[\boldsymbol{a}]_i}$ by encoding $[\boldsymbol{a}]_i$ as a tweaked interpolation packing with $D = \bar{d}, \Delta = \delta, T = t$ (Section 3.4, Remark 3).
2. All parties compute $\mathfrak{ct}^{(1)}_{\boldsymbol{a}} = \sum_{i\in[n]} \mathfrak{ct}_{[\boldsymbol{a}]_i}$, then performs ZKPoPK and ZKPoMK on it with parameter $(D = \bar{d}, \Delta = \delta, T = t)$.
3. All parties repeat 1. and 2. for $[\boldsymbol{b}]_i \in \mathbb{Z}^{\rho}_{2^k}$ to get $\mathfrak{ct}^{(1)}_{\mathbf{b}}$.
4. Parties locally compute $\mathfrak{ct}^{(0)}_{\mathbf{c}} \leftarrow \mathsf{ModSwitch}(\mathfrak{ct}^{(1)}_{\mathbf{a}} \boxtimes \mathfrak{ct}^{(1)}_{\mathbf{b}})$.
5. The parties run $\mathit{\Pi}_{\mathsf{Reshare}}$.**D2** (Section 3.4, Fig. 3) on input $\mathfrak{ct}^{(0)}_{\mathbf{c}}$ and $\mathsf{pp} = (2\bar{d}, 2\delta, t)$, but each party decodes $\bar{d}+1$ values on each CRT ring only (instead of $2\bar{d}$) so that each P_i receives $[\mathbf{c}]_i$ and a fresh ciphertext $\bar{\mathfrak{ct}}^{(1)}_{\mathbf{c}}$.
6. Parties run $\mathit{\Pi}_{\mathsf{Auth}}$ (Fig. 9) on inputs $\mathfrak{ct}^{(1)}_{\mathbf{a}}$ and $\mathfrak{ct}^{(1)}_{\mathbf{b}}$ each with $\mathsf{pp} = (\bar{d}, \delta, t)$, then run $\mathit{\Pi}_{\mathsf{Auth}}$ on input $\bar{\mathfrak{ct}}^{(1)}_{\mathbf{c}}$ with $\mathsf{pp} = (2\bar{d}, 2\delta, t)$ to obtain $(\langle \boldsymbol{a_j} \rangle, \langle \boldsymbol{b_j} \rangle, \langle \boldsymbol{c_j} \rangle)_{j\in[\rho]}$.[*]

[*] Notation $\langle \boldsymbol{a} \rangle$ denotes $([\boldsymbol{a}]_i, [\boldsymbol{\gamma_a}] := [\alpha \cdot \boldsymbol{a}]_i)_{i\in[n]}$.

Fig. 8. Offline (weak) preprocessing protocol

Subprotocol $\mathit{\Pi}_{\mathsf{Auth}}$

On input $(\mathfrak{ct}_{\boldsymbol{m}}, \mathsf{pp})$,

1. Parties locally compute $\mathfrak{ct}_{\alpha\cdot\boldsymbol{m}} \leftarrow \mathfrak{ct}_{\alpha} \boxtimes \mathfrak{ct}_{\boldsymbol{m}}$.
2. Parties call $\mathcal{F}_{\mathsf{KeyGenDec}}.\mathbf{D1}$ on input $\mathfrak{ct}_{\alpha\cdot\boldsymbol{m}}$ and pp, each P_i receives $[\boldsymbol{\gamma_m}]_i := [\alpha\cdot\boldsymbol{m}]_i$.

Fig. 9. Subprotocol $\mathit{\Pi}_{\mathsf{Auth}}$ for $\mathit{\Pi}_{\mathsf{wPrep}}$

Functionality $\mathcal{F}_{\mathsf{wPrep}}$

Init:
1. Wait for an input from the adversary, if it is abort, then abort.
2. Otherwise, receive $[\alpha]_{\mathcal{A}} \in \mathbb{Z}_{2^k}$ from the adversary, then uniformly sample $\{[\alpha]_j \in \mathbb{Z}_{2^k}\}_{j\notin\mathcal{A}}$. Store $\alpha := [\alpha]_{\mathcal{A}} + \sum_{j\notin\mathcal{A}}[\alpha]_j$.

Input, P_I:
1. If $I \in \mathcal{A}$, set $\boldsymbol{m} = \boldsymbol{0}$ and receive shares $\{[\boldsymbol{m}]_j \in \mathbb{Z}_{2^k}^{\rho}\}_{j\notin\mathcal{A}}$ for honest parties from the adversary.
   If $I \notin \mathcal{A}$, uniformly sample $\boldsymbol{m}$ and $\{[\boldsymbol{m}]_i \in \mathbb{Z}_{2^k}^{\rho}\}_{i\in[n]}$.
2. Send each shares to corresponding parties (and $\boldsymbol{m}$ to P_I). Wait for an input from the adversary, if it is abort, then abort.
3. Otherwise, receive $\boldsymbol{\delta_m}$ and $\boldsymbol{\delta_\gamma}$ from the adversary, then run the macro $\mathbf{Auth}(\boldsymbol{m}+\boldsymbol{\delta_m}, \boldsymbol{\delta_\gamma})$ to get $[\boldsymbol{\gamma_m}]_j$'s. Output $[\boldsymbol{\gamma_m}]_j$'s to each honest P_j's.

Triple:
1. If adversary sends abort, then abort. Otherwise, receive $\{[\boldsymbol{a}]_{\mathcal{A}}, [\boldsymbol{b}]_{\mathcal{A}} \in \mathbb{Z}_{2^k}^{\rho}\}$ and $\{\boldsymbol{\delta_{\gamma_a}}, \boldsymbol{\delta_{\gamma_b}}, \boldsymbol{\delta_{\gamma_c}}, \boldsymbol{\delta_c} \in \mathbb{Z}_{2^k}^{\rho}\}$ from the adversary.
2. Uniformly sample $\{[\boldsymbol{a}]_j, [\boldsymbol{b}]_j \in \mathbb{Z}_{2^k}^{\rho}\}_{j\notin\mathcal{A}}$ and set $\boldsymbol{a} = [\boldsymbol{a}]_{\mathcal{A}} + \sum_{j\notin\mathcal{A}}[\boldsymbol{a}]_j$, $\boldsymbol{b} = [\boldsymbol{b}]_{\mathcal{A}} + \sum_{j\notin\mathcal{A}}[\boldsymbol{b}]_j$.
3. Set $\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b} + \boldsymbol{\delta_c}$.* Uniformly sample honest shares $\{[\boldsymbol{c}]_j \in \mathbb{Z}_{2^k}^{\rho}\}_{j\notin\mathcal{A}}$ such that $\sum_{j\notin\mathcal{A}}[\boldsymbol{c}]_j = \boldsymbol{c}$.
4. Run the macros to get followings:**
   $\langle\boldsymbol{a}\rangle \leftarrow \mathbf{Auth}(\boldsymbol{a}, \boldsymbol{\delta_{\gamma_a}})$, $\langle\boldsymbol{b}\rangle \leftarrow \mathbf{Auth}(\boldsymbol{b}, \boldsymbol{\delta_{\gamma_b}})$, $\langle\boldsymbol{c}\rangle \leftarrow \mathbf{Auth}(\boldsymbol{c}, \boldsymbol{\delta_{\gamma_c}})$.
5. Output $(\langle\boldsymbol{a}\rangle_j, \langle\boldsymbol{b}\rangle_j, \langle\boldsymbol{c}\rangle_j)$ to each honest P_j.

$\mathbf{Auth}(\boldsymbol{x}, \boldsymbol{\delta_\gamma} \in \mathbb{Z}_{2^k}^{\rho})$:
1. Set $\boldsymbol{\gamma_x} = \alpha\cdot\boldsymbol{x}$ where $\alpha \in \mathbb{Z}_{2^k}$ is stored at **Init**.
2. Uniformly sample honest shares $\{[\boldsymbol{\gamma_x}]_j \in \mathbb{Z}_{2^k}^{\rho}\}_{j\notin\mathcal{A}}$ such that $\sum_{j\notin\mathcal{A}}[\boldsymbol{\gamma_x}]_j = \boldsymbol{\gamma_x} + \boldsymbol{\delta_\gamma}$.
3. If adversary sends abort, then abort. Otherwise, return $([\boldsymbol{\gamma_x}]_j)_{j\notin\mathcal{A}}$

$*$ $\odot$ denotes Hadarmad product, i.e., componenet-wise product.
** Notation $\langle\boldsymbol{a}\rangle$ denotes $([\boldsymbol{a}]_j, [\boldsymbol{\gamma_a}]_j := [\alpha\cdot\boldsymbol{a}]_j)_{j\notin\mathcal{A}}$.

Fig. 10. Functionality for preprocessing phase

Simulator $\mathcal{S}_{\mathsf{wPrep}}$

Let $\mathcal{A}$ be the set of corrupted parties.

Init:

1. With $\mathfrak{sk}_i$ from $\mathcal{A}$, emulate (internally run) $\mathcal{F}_{\mathsf{KeyGenDec}}$.**Init** (Fig. 6) to obtain $\mathfrak{pk}, \mathfrak{sk}'$, and $(\mathfrak{sk}'_j)_{j \notin \mathcal{A}}$.
2. Run the step 2. and 3. of the protocol Π_{wPrep}.**Init** (Fig. 8) $\mathcal{A}$, participating as the honest parties P_j with $\mathfrak{sk}'_j$'s.
3. If the ciphertext $\mathfrak{ct}_{\alpha}^{(1)}$ does not pass the ZKPoPK or ZKPoMK, send abort to the functionality. Otherwise, decrypt $\mathfrak{ct}_{\alpha}^{(1)}$ to get α, then send $[\alpha]_{\mathcal{A}} := \alpha - \sum_{j \notin \mathcal{A}} [\alpha]_j$ to the functionality.
4. Store α.

Input, P_I:

1. Run the step 1., 2., and 3. of the protocol Π_{wPrep}.**Input** (Fig. 8) with $\mathcal{A}$, participating as the honest parties. Meanwhile, if $I \in \mathcal{A}$, transmit to the functionality $\mathcal{F}_{\mathsf{wPrep}}$, $\{[\boldsymbol{m}]_j\}_{j \notin \mathcal{A}}$ given from $\mathcal{A}$; if $I \notin \mathcal{A}$, transmit to $\mathcal{A}$, $\{[\boldsymbol{m}]_i\}_{i \in \mathcal{A}}$ given from the functionality $\mathcal{F}_{\mathsf{wPrep}}$.
2. If the ciphertext $\mathfrak{ct}_{\boldsymbol{m}}^{(1)}$ does not pass the ZKPoPK or ZKPoMK, send abort to the functionality. Otherwise, decrypt $\mathfrak{ct}_{\boldsymbol{m}}^{(1)}$ to get $\boldsymbol{m}$. Then, compute
$$\boldsymbol{\delta_m} := \begin{cases} \boldsymbol{m} - \sum_{j \notin \mathcal{A}} [\boldsymbol{m}]_j & \text{if } I \in \mathcal{A} \\ \boldsymbol{m} - \sum_{i \in [n]} [\boldsymbol{m}]_i & \text{if } I \notin \mathcal{A} \end{cases}.$$
3. Run the step 4. of the protocol Π_{wPrep}.**Input** (Fig. 8) with $\mathcal{A}$, but emulate $\mathcal{F}_{\mathsf{KeyGenDec}}$.**D1** (Fig. 6) to get abort or $\boldsymbol{\eta}$ from the adversary. Send abort or $\boldsymbol{\delta_m}$ (computed as above) and $\boldsymbol{\delta_\gamma} = \boldsymbol{\eta}$ accordingly to the functionality $\mathcal{F}_{\mathsf{wPrep}}$.

Triple:

1. Run the step 1., 2., 3., and 4. of the protocol Π_{wPrep}.**Triple** (Fig. 8) with $\mathcal{A}$, participating as the honest parties. Meanwhile, if the ciphertext $\mathfrak{ct}_{\boldsymbol{a}}^{(1)}$ or $\mathfrak{ct}_{\boldsymbol{b}}^{(1)}$ does not pass the ZKPoPK or ZKPoMK, send abort to the functionality.
2. Otherwise, decrypt $\mathfrak{ct}_{\boldsymbol{a}}^{(1)}$ to get $\boldsymbol{a}$, compute $[\boldsymbol{a}]_{\mathcal{A}} = \boldsymbol{a} - \sum_{j \notin \mathcal{A}} [\boldsymbol{a}]_j$. Repeat this process for $\mathfrak{ct}_{\boldsymbol{b}}^{(1)}$ to get $[\boldsymbol{b}]_{\mathcal{A}}$.
3. Run the step 5. of the protocol Π_{wPrep}.**Triple** (Fig. 8) with $\mathcal{A}$, participating as the honest parties: send abort to the functionality if it occurs during execution. When running Π_{Reshare}.**D2**, if P_1 is honest, retrieve $\boldsymbol{\delta_c}$ from $\sum_{i \in \mathcal{A}} (v_i^* - w_i)$ (where v_i^*'s denote the v_i's given from $\mathcal{A}$) regarding it as a tweaked interpolation packing. If P_1 is corrupt, set $\boldsymbol{\delta_c} = \sum_{j \notin \mathcal{A}} (-\boldsymbol{f}_j)$.
4. Run the step 6. of the protocol Π_{wPrep}.**Triple** (Fig. 8) with $\mathcal{A}$, but emulate $\mathcal{F}_{\mathsf{KeyGenDec}}$.**D1** (Fig. 6) on $\mathfrak{ct}_{\mathbf{a}}^{(1)}, \mathfrak{ct}_{\mathbf{b}}^{(1)}, \bar{\mathfrak{ct}}_{\mathbf{c}}^{(1)}$ to get abort or $\boldsymbol{\eta_a}, \boldsymbol{\eta_b}, \boldsymbol{\eta_c}$ from the adversary. Send abort or $\boldsymbol{\delta_{\gamma_a}} = \boldsymbol{\eta_a}$, and similarly for $\boldsymbol{\delta_{\gamma_b}}, \boldsymbol{\delta_{\gamma_c}}$ accordingly to the functionality $\mathcal{F}_{\mathsf{wPrep}}$.

Fig. 11. Simulator for $\mathcal{F}_{\mathsf{wPrep}}$

B.5 ZKPoMK

We give a full description of Π_{PoMK} (Fig. 12, Fig. 13) and show that it satisfies correctness, soundness, and zero-knowledge, assuming that subroutine ZKPoPK[xxvii] for $\hat{\mathfrak{ct}}_{a'}$ has perfect correctness, zero-knowledge, and soundness. For readability, we use the same notation as the overview in Section 6.1: hat $\hat{}$ and apostrophe $'$ denote that they are related to proving the statement (ii) $\mathsf{Div}_T^{(D,\Delta)}(a) = \mathsf{true}$. That is, roughly speaking, a prover provides a' such that $a = 2^{\Delta} a'$ as a ciphertext $\hat{\mathfrak{ct}}_{a'}$.

Correctness. The same correctness statement as Theorem 3 holds for our ZKPoMK, except that here we do not need a bound on u. Honest inputs pass the predicate checks ((b) in line 2 of Verify algorithm) since our predicates satisfy certain homomorphic properties:

Note that $\mathsf{Deg}^{(D)}(a_k^i) = \mathsf{true}$ and $\mathsf{Deg}^{(D)}(y_l^i) = \mathsf{true}$ by the honest sampling and commitment phases (Fig. 12). Then, $\mathsf{Deg}^{(D)}(z_l) = \mathsf{true}$ follows from the linearly homomorphic property of Deg and the equality $z_l = \sum_{i=1}^n y_l^i + (W)_l \cdot \boldsymbol{a}^i$ with the entries of W being *constants*, i.e., degree-0 polynomials.

The case of $\mathsf{DivCheck}$ can be shown similarly, with a note that $\mathsf{DivCheck}$ is homomorphic under multiplications of same *constants* on both inputs (i.e., $\mathsf{DivCheck}((W)_l \cdot \boldsymbol{a}^i, (W)_l \cdot \boldsymbol{a}'^i) = \mathsf{true}$ given that $\mathsf{DivCheck}(a_k^i, a_k'^i) = \mathsf{true}$).

Zero-Knowledge. The same zero-knowledge statement as Theorem 4 holds for our ZKPoMK, except that the statistical distance is bounded by $16Muv2^E/2^{\mathsf{ZK\_sec}}$. The proof also stays essentially the same, applying Lemma 3 with appropriate predicates.

Soundness. The same soundness statement as Theorem 5 holds for our ZKPoMK, except that we additionally require an assumption $T \geq E + \Delta$ (this assumption always holds in our application, since we will set $E \leq \Delta$ to manage slackness and set $T = t = k + 2\Delta$ for tweaked interpolation packing). The proof also stays essentially the same:

With a rewinding argument, from the two accepting transcripts regarding $\boldsymbol{z}^i$ (we omit $\boldsymbol{r}_z^i$ for simplicity), we get

$$z_l - \bar{z}_l = (\omega_{l,k} - \bar{\omega}_{l,k}) \cdot a_k \text{ for some } l \in [v]. \tag{4}$$

Note that the largest possible power-of-two divisor of $(\omega_{l,k} - \bar{\omega}_{l,k})$ is 2^E, since each ω is from the challenge space $\mathsf{Chal} = [-2^E, 2^E - 1] \cap \mathbb{Z}$. Hence, $\mathsf{Deg}_T^{(D)}(z_l) = \mathsf{Deg}_T^{(D)}(\bar{z}_l) = \mathsf{true}$ implies that $\mathsf{Deg\_sl}_T^{(D,E)}(a_k) = \mathsf{true}$.

[xxvii] We can use our $\Pi_{\mathsf{PoPK}}^{\mathsf{TG2k}}$ or the original TopGear [BCS19] depending on the cyclotomic ring $\Phi_{\hat{M}}(X)$ that $\hat{\mathfrak{ct}}_{a'}$ use. Note that, since we use coefficient embedding for $\hat{\mathfrak{ct}}_{a'}$, we can also exploit power-of-two cyclotomic ring regardless of its splitting property over $\mathbb{Z}_{2^k}$.

Protocol Π_{PoMK} - Part I

Implicitly call $\mathcal{F}_{\mathsf{Rand}}$ (Appendix B.2) when sampling a challenge.

PARAMETERS:

- We use $\mathsf{ZK\_sec}, 2^t, u, v, n$ same as in Fig. 4.
- D, Δ, T: the parameters for the predicate $\mathsf{Pack}_T^{(D,\Delta)} : R \to \{\mathsf{true}, \mathsf{false}\}$.
- E: a positive integer parameter for the challenge space Chal.
- We use an additional set of SHE parameters whose plaintext space is $\mathbb{Z}_{2^{t-E}}[X]/\Phi_{\hat{M}}(X)$. The corresponding public key is denoted as $\hat{\mathfrak{pk}}$. When encrypting by this set of parameters, denoted as $\widehat{\mathsf{Enc}}$, we use coefficient packing (instead of our tweaked interpolation packing).

$\mathsf{Sampling}$ (Sampling phase)

1. Set the predicate $\mathsf{P} \leftarrow \mathsf{Pack}_T^{(D,\Delta)}$.
2. For each $k \in [u]$ do
   (a) Sample $a_k^i \leftarrow \mathcal{U}_{\mathsf{P}}(2^{t-1})$ and $r_{a_k}^i \leftarrow \mathsf{RC}(\sigma^2, \rho)$.
   (b) Compute a ciphertext $\mathfrak{ct}_{a_k}^i = \mathsf{Enc}(a_k^i, r_{a_k}^i; \mathfrak{pk})$.
3. For each $k \in [u]$ do
   (a) Set the predicate $\mathsf{P}_k \leftarrow \mathsf{DivCheck}_T^{(D,\Delta)}(a_k^i, \cdot)$ with respect to $\mathbb{Z}[X]/\Phi_{\hat{M}}(X)$.
   (b) Sample $a_k'^i \leftarrow \mathcal{U}_{\mathsf{P}_k}(2^{t-E-1})$[xxviii]
   (c) Compute ciphertexts $\hat{\mathfrak{ct}}_{a_k'}^i = \widehat{\mathsf{Enc}}(a'^i_k, r_{a_k'}^i; \hat{\mathfrak{pk}})$.
4. Output $(\mathfrak{ct}_a^i, \hat{\mathfrak{ct}}_{a'}^i, \boldsymbol{a}^i, \boldsymbol{r}_a^i, \boldsymbol{a}'^i, \boldsymbol{r}_{a'}^i)$, which are defined similarly as in Fig. 4.
5. Perform ZKPoPK on $(\hat{\mathfrak{ct}}_{a'}^i, \boldsymbol{a}'^i, \boldsymbol{r}_{a'}^i)$.

Commit (Commitment phase)

0. In the following, the party P_i implicitly samples, for each y, corresponding noises and randomnesses $r_y^i \leftarrow (r_y^{i,(\ell)} \leftarrow \mathcal{U}(2^{\mathsf{ZK\_sec}} \cdot \rho_\ell))_{\ell \in [3]}$.
1. To generate $2v$ masking ciphertexts, each P_i do the followings, for each $l \in [v]$.
   (a) Set the predicate $\mathsf{P} \leftarrow \mathsf{Deg}^{(D)}$.
   (b) Sample $y_l^i \leftarrow \mathcal{U}_{\mathsf{P}}(2^{\mathsf{ZK\_sec}} \cdot 2^{t-1})$ and compute $\mathfrak{ct}_{y_l}^i = \mathsf{Enc}(y_l^i, r_{y_l}^i; \mathfrak{pk})$.
   (c) Set the predicate $\mathsf{P}_l \leftarrow \mathsf{DivCheck}_T^{(D,\Delta)}(y_l^i, \cdot)$ with respect to $\mathbb{Z}[X]/\Phi_{\hat{M}}(X)$.
   (d) Sample $y_l'^i \leftarrow \mathcal{U}_{\mathsf{P}_l}(2^{\mathsf{ZK\_sec}} \cdot 2^{t-E-1})$ and compute $\hat{\mathfrak{ct}}_{y_l'}^i = \widehat{\mathsf{Enc}}(y_l'^i, r_{y_l'}^i; \hat{\mathfrak{pk}})$.
2. Party P_i keeps $\mathsf{state}_i \leftarrow (\boldsymbol{y}^i, \boldsymbol{y}'^i, \boldsymbol{r}_y^i, \boldsymbol{r}_{y'}^i)$ and broadcasts $\mathsf{comm}_i \leftarrow (\mathfrak{ct}_y^i, \hat{\mathfrak{ct}}_{y'}^i)$.
   $(\boldsymbol{y}^i, \boldsymbol{y}'^i, \boldsymbol{r}_y^i, \boldsymbol{r}_{y'}^i, \mathfrak{ct}_y^i, \hat{\mathfrak{ct}}_{y'}^i$ are defined similarly as in Fig. 4.)

$\mathsf{Challenge}$ (Challenge phase)

1. Parties together uniformly sample challenge matrices W of size $v \times u$, whose entries are sampled from the challenge space $\mathsf{Chal} = [-2^E + 1, 2^E] \cap \mathbb{Z}$.

[xxviii] Uniformly sample corresponding element at each coefficient.

Fig. 12. Protocol Π_{PoMK} - Sampling, Commitment, and Challenge phases

Protocol Π_{PoMK} - Part II

$\mathsf{Response}$ (Response phase)

1. Each party P_i computes,
   (a) $\boldsymbol{z}^i \leftarrow \boldsymbol{y}^i + W \cdot \boldsymbol{a}^i$ and $\boldsymbol{r}_z^i \leftarrow \boldsymbol{r}_y^i + W \cdot \boldsymbol{r}_a^i$.[xxviii]
   (b) $\boldsymbol{z}'^i \leftarrow \boldsymbol{y}'^i + W \cdot \boldsymbol{a}'^i$ and $\boldsymbol{r}_{z'}^i \leftarrow \boldsymbol{r}_{y'}^i + W \cdot \boldsymbol{r}_{a'}^i$.[xxviii]
2. Party P_i sets $\mathsf{resp}_i \leftarrow (\boldsymbol{z}^i, \boldsymbol{r}_z^i, \boldsymbol{z}'^i, \boldsymbol{r}_{z'}^i)$ and broadcasts resp_i.

Verify (Verification phase)

1. Each party P_i computes,
   (a) $\mathfrak{ct}_z^i \leftarrow (\mathsf{Enc}(\boldsymbol{z}_l^i, \boldsymbol{r}_{z_l}^i; \mathfrak{pk}))_{l\in[v]}$, $\hat{\mathfrak{ct}}_{z'}^i \leftarrow (\widehat{\mathsf{Enc}}(\boldsymbol{z}_l'^i, \boldsymbol{r}_{z_l'}^i; \hat{\mathfrak{pk}}))_{l\in[v]}$.
   (b) $\mathfrak{ct}_a$, $\mathfrak{ct}_y$, $\mathfrak{ct}_z$, $\hat{\mathfrak{ct}}_{a'}$, $\hat{\mathfrak{ct}}_{y'}$, $\hat{\mathfrak{ct}}_{z'}$, $\boldsymbol{z}$, $\boldsymbol{z}'$ are defined similarly as in Fig. 4.
2. Parties accept if all of the followings hold, otherwise they reject.
   (a) $\mathfrak{ct}_z = \mathfrak{ct}_y + W \cdot \mathfrak{ct}_a$, $\hat{\mathfrak{ct}}_{z'} = \hat{\mathfrak{ct}}_{y'} + W \cdot \hat{\mathfrak{ct}}_{a'}$.
   (b) For $l \in [v]$, $\mathsf{Deg}^{(D)}(z_l) = \mathsf{true}$, $\mathsf{DivCheck}_T^{(D,\Delta)}(z_l, z_l') = \mathsf{true}$

Fig. 13. Protocol Π_{PoMK} - Response and Verification phases

Similarly, rewinding argument on the transcripts regarding $\boldsymbol{z}'^i$, we get,

$$z_l' - \bar{z}_l' = (\omega_{l,k} - \bar{\omega}_{l,k}) \cdot a_k' \text{ for the same } l \in [v] \text{ as Eq. (4).}$$

Then, $\mathsf{DivCheck}_T^{(D,\Delta)}(z_l, z_l') = \mathsf{true}$ and $\mathsf{DivCheck}_T^{(D,\Delta)}(\bar{z}_l, \bar{z}_l') = \mathsf{true}$ together implies that, after CRT projection (and evaluation at $j \in [0; D]$),

$$a_k = 2^{\Delta} \cdot a_k' + b \cdot 2^{T-E} \text{ for some } b,$$

which is equivalent to $\mathsf{Div}_T^{(D,\Delta)}(a_k) = \mathsf{true}$ from the assumption $T \geq E + \Delta$.

B.6 Definition of Soundness for Theorem 5

– Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be a tuple of PPT algorithms. Consider the following game:
  1. $\mathcal{A}_1$ outputs $I \subset [n], \{\mathfrak{ct}_a^i\}_{i \in I}$ and $\mathsf{state}_{\mathcal{A}_1}$.
  2. Choose $(\mathfrak{ct}_a^j, \boldsymbol{a}^j, \boldsymbol{r}_a^j) \leftarrow \mathsf{Sampling}(j)$ honestly for $P_j, j \notin I$.
  3. Compute $(\mathsf{comm}_j, \mathsf{state}_j) \leftarrow \mathsf{Commit}(\mathfrak{ct}_a^j, \boldsymbol{a}^j, \boldsymbol{r}_a^j)$ for $j \notin I$.
  4. $\mathcal{A}_2$ on input $\mathsf{state}_{\mathcal{A}_1}, \{\mathfrak{ct}_a^j, \mathsf{comm}_j\}_{j \notin I}$ outputs $\mathsf{state}_{\mathcal{A}_2}, \{\mathsf{comm}_i\}_{i \in I}$.
  5. Choose a uniformly random W and compute $\mathsf{resp}_j \leftarrow \mathsf{Response}(\mathsf{state}_j, W)$ for $j \notin I$.
  6. $\mathcal{A}_3$ on input $\mathsf{state}_{\mathcal{A}_2}, W, \{\mathsf{resp}_j\}_{j \notin I}$ outputs $\{\mathsf{resp}_i\}_{i \in I}$.
  7. $\mathcal{A}$ wins the game if $\mathsf{Verify}(\{\mathsf{comm}_i, \mathsf{resp}_i\}_{i \in [n]}, W) = \mathsf{true}$.

  Suppose $\mathcal{A}$ wins the game with probability $\epsilon > 2^{-\mathsf{Snd\_sec}}$. Then there exists a PPT algorithm $\mathsf{Extract}$ which, for any fixed output of $\mathcal{A}_1$, with honestly generated $\{\mathfrak{ct}_a^j, \boldsymbol{a}^j, \boldsymbol{r}_a^j, \mathsf{comm}_j, \mathsf{state}_j\}_{j \notin I}$ as inputs, and black-box access to $(\mathcal{A}_2, \mathcal{A}_3)$, outputs $\{witness^i\}_{i \in I}$ such that $\mathcal{R}_{\mathsf{PoPK}}^{S,u}$ (Eq. (3), with the soundness slack $S = 8\varphi(M) \cdot 2^{\mathsf{ZK\_sec}}$) holds in expected $f_u(\mathsf{Snd\_sec})/(\epsilon - 2^{-\mathsf{Snd\_sec}})$ steps where $f_u(\cdot)$ is a positive polynomial (depending on u).