TopGear 2.0: Accelerated Authenticated Matrix Triple Generation with Scalable Prime Fields via Optimized HE Packing

HyunHo Cha
Seoul National University

Intak Hwang
Seoul National University

Seonhong Min
Seoul National University

Jinyeong Seo
Seoul National University

Yongsoo Song
Seoul National University

***Abstract*—The SPDZ protocol family is a popular choice for secure multi-party computation (MPC) in a dishonest majority setting with active adversaries. Over the past decade, a series of studies have focused on improving its offline phase, where special additive shares, called authenticated triples, are generated. However, to accommodate recent demands for matrix operations in secure machine learning and big integer arithmetic in distributed RSA key generation, updates to the offline phase are required.**

In this work, we propose a new protocol for the SPDZ offline phase, TopGear 2.0, which improves upon the previous state-of-the-art construction, TopGear (Baum et al., SAC'19), and its variant for matrix triples (Chen et al., Asiacrypt'20). Our protocol aims to achieve a speedup in matrix triple generation and support for larger prime fields, up to 4096 bits in size. To achieve this, we employ a variant of the BFV scheme and a homomorphic matrix multiplication algorithm optimized for our purpose.

As a result, our protocol achieves about 3.6x speedup for generating scalar triples in a 1024-bit prime field and about 34x speedup for generating 128x128 matrix triples. In addition, we reduce the size of evaluation keys from 27.4 GB to 0.22 GB and the communication cost for MAC key generation from 816 MB to 16.6 MB.

1. Introduction

Secure Multi-Party Computation (MPC) enables a set of parties to jointly compute a function over their private inputs without revealing any information other than the result. Constructing MPC protocols has been a major research topic in the field of cryptography, and there has been a number of solutions optimized for various settings, such as assuming an honest or dishonest majority among the parties, and whether the corrupted parties behave passively or actively.

In a dishonest majority setting with an active adversary, the SPDZ protocol family [DPSZ12, KPR18, CKR$^+$20, BCS19] has been a popular choice, as it provides active security against up to $n-1$ corrupted parties. The SPDZ protocol is divided into two phases: an online phase and an offline phase. In the online phase, parties jointly compute a target function based on additive shares of their private inputs, which are elements of some prime field $\mathbb{Z}_p$. Since the protocol assumes the active security, the private inputs are distributed in a special form called authenticated shares to prevent deviation from the protocol. At the end of the online phase, the parties jointly verify the output by checking the validity of the authenticated shares to ensure the protocol was correctly executed.

In the offline phase, authenticated shares are generated in addition to authenticated triples, which are required for performing multiplication in the online phase. Multiplications in the online phase are processed using Beaver's trick [Bea92], which requires additive shares of a, b, and c such that $ab = c \pmod p$. The authenticated shares of such triples (a, b, c) are called authenticated triples. The core idea of the initial SPDZ protocol [DPSZ12] is to generate authenticated triples using homomorphic encryption (HE). The parties first jointly generate HE ciphertexts ct_a and ct_b for a and b, respectively. They then compute the encryption ct_c of c through the homomorphic multiplication of ct_a and ct_b. Finally, they collectively decrypt ct_c to obtain the additive share of c.

To achieve active security in the offline phase, it is necessary to validate the HE ciphertexts ct_a and ct_b to prevent the use of ill-formed ciphertexts that could be generated by malicious adversaries. This is done using a proof of plaintext knowledge (PoPK) protocol, which is a Schnorr-like zero-knowledge protocol that verifies the well-formedness of the HE ciphertexts. Since the PoPK protocol has been a main performance bottleneck in the offline phase, subsequent studies [KPR18, BCS19] on the SPDZ protocol have focused on improving the efficiency of the PoPK protocol.

Recently, MPC protocols have been used in a wide range of applications. One important application is privacy-preserving machine learning [MZ17, MR18, DEK20, WGC19], which enables distrustful parties to perform machine learning tasks on their private data. Since machine learning tasks heavily utilize matrix operations, recent improvements [CKR$^+$20, MG23, RRKK23] to the SPDZ protocol have introduced the concept of authenticated matrix triples, which significantly reduce the overhead of matrix

operations in both the online and offline phases. On the other hand, MPC protocols can also be utilized in distributed key generation for threshold cryptography, which has recently gained attention due to NIST's preparation for the standardization of threshold cryptography [BP23]. For example, there have been several studies on distributed RSA key generation [BDF+23, FLOP18, CDK+22], which requires arithmetic MPC with large moduli for the multiplication of two large prime numbers.

1.1. Our Contribution

In this paper, we propose a new offline phase for SPDZ, which we denote as TopGear 2.0, improving upon the previous state-of-the-art construction [BCS19], TopGear. In our protocol, we address the recent demands for matrix multiplication and large modulus fields.

Generalized BFV. To instantiate large modulus arithmetic in SPDZ, authenticated triples must be generated within large fields, which requires HE schemes designed for such large fields. To achieve this, we devise a new variant of the BFV homomorphic encryption scheme, which we denote as generalized BFV. Our generalized BFV scheme can efficiently handle messages from large prime fields by employing a novel packing method for large prime fields, following the idea of [BCIV20].

Let p be the modulus of the message space and q be the modulus of the ciphertext space. In the ordinary BFV scheme [Bra12, FV12], q must be set much larger than p, as the available depth for homomorphic multiplication is roughly determined by the factor $\log q / \log p$. However, both the computational complexity of homomorphic multiplication and the space complexity of the evaluation key scale approximately with $O(\log^2 q)$. Due to this nature, performance significantly degrades when handling messages from large moduli because of the huge increase in the ciphertext modulus q.

Our packing method effectively addresses this issue by reducing the overhead of homomorphic multiplication from $\log p$ to $\log b$, when $p = b^M + 1$. Then, the available multiplicative depth is now determined by the factor $\log q / \log b$, which allows for a much smaller q for the same field size p. As a result, we can use a ciphertext modulus with a bit length ten times smaller than the ordinary BFV scheme when p is a 1024-bit prime. This results in approximately a threefold speedup in generating authenticated triples, and a reduction in the evaluation key size by about 125 times.

Faster Homomorphic Matrix Multiplication. To accommodate the demand for efficient matrix operations, previous work [CKR+20] proposed a protocol for authenticated matrix triple generation, which utilized the homomorphic matrix multiplication algorithm from [JKLS18]. To further accelerate performance, we devise a new homomorphic matrix multiplication algorithm that asymptotically improves computational complexity.

For $d \times d$ matrix multiplication, the previous algorithm in [JKLS18] required $\widetilde{O}(d^3 \cdot \log^2 q)$ complexity and $3d - 2$ evaluation keys. In this work, we reduce the computational complexity to $\widetilde{O}(d^3 \cdot \log q)$ and the number of evaluation keys to d by introducing a new matrix encoding map called *shifted diagonal encoding*. This improvement particularly synergizes with our generalized BFV scheme, which effectively reduces the size of $\log q$. As a result, we achieve approximately a 30-fold speedup in computing 128×128 matrix multiplication for a field size $p \approx 2^{128}$.

PoPK for Generalized BFV. To employ the generalized BFV scheme for authenticated triple generation, an efficient PoPK protocol optimized for the generalized BFV scheme is required. In the PoPK protocol, there exists a gap between the honest statement and the proven statement, known as soundness slack. Since the soundness slack influences parameter settings, particularly the ciphertext modulus q, it is crucial to minimize this slack. Otherwise, it can degrade the performance of HE operations and increase communication costs. In a previous study [KLSS23], a method for achieving constant-sized soundness slack was proposed for a special case of the BFV scheme. However, this method does not directly apply to our case, as we have modified the structure of ciphertexts in our generalized BFV scheme. To achieve constant-sized soundness slack in generalized BFV, we adapt a recently proposed technique by [HSS24] that randomizes the packing procedure during encryption. As a result, we achieve constant-sized soundness slack for our generalized BFV, similar to the previous work [KLSS23].

1.2. Related Works

TopGear. The most relevant works to ours are TopGear [BCS19] and its variant for matrix triples [CKR+20]. TopGear is designed to support a large number of parties, so its performance scales well as the number of participants grows. The core idea of TopGear is to run a PoPK protocol in which all parties participate, producing a single joint proof that validates all parties' HE ciphertexts. Additionally, the generation of authenticated triples in TopGear is done in a similar manner, utilizing homomorphic multiplication. This contrasts with other SPDZ offline phases [DPSZ12, KPR18], where proofs and authenticated triples are generated pairwise between parties.

LowGear. There is another line of work for the SPDZ offline phase, called LowGear [KPR18, RRKK23]. In the LowGear protocol, each party pairwise generates PoPK proofs and authenticated triples. This approach has advantages with a small number of parties, as it can replace costly homomorphic multiplications with cheaper linear homomorphic operations. However, its performance degrades rapidly as the number of parties increases. We expect that our generalized BFV scheme could also be applied to the LowGear protocol and its variants.

Pseudorandom Correlation Generator. Aside from homomorphic encryption, authenticated triples can be generated using the pseudorandom correlation generator [BCG+19], which enables the generation of correlated randomness with

sublinear communication costs. Abram and Scholl [AS22] proposed a protocol for authenticated triple generation based on the pseudorandom correlation generator. Their protocol requires a total communication cost of $O(n^4\sqrt{T})$ to generate T triples, compared to $O(n^2T)$ in the TopGear and LowGear protocols, where n is the number of parties. However, the generation of authenticated matrix triples is not addressed in the literature.

Matrix Multiplication. There has been numerous research on secure matrix multiplication using HE. [HS14, HS18] introduced an algorithm for linear transformation of an encrypted vector, which can be trivially extended to matrix multiplication with the number of ciphertexts equal to the dimension. On the other hand, [JKLS18, JLK+22] studied the case where the entire matrix can be packed in a single ciphertext. For more detailed explanation and comparison, we refer to Section 4.1. Alternatively, Zheng et al. [ZLW23] showed that trace maps can be used to accelerate the matrix multiplication if the matrix dimension is even smaller. However, their method requires arithmetic over non-power-of-two cyclotomic rings, which is inefficient in practice.

2. Background

2.1. Notation

Let N be a power of two and q be an integer. We denote by $K = \mathbb{Q}[X]/(X^N + 1)$ the $(2N)$-th cyclotomic field, $R = \mathbb{Z}[X]/(X^N + 1)$ the ring of integers of K, and $R_q = R/qR$ the quotient ring of R. We identify an element $a = \sum_{0\le i<N} a_i \cdot X^i \in R_q$ with the vector of its coefficients $(a_0, \ldots, a_{N-1}) \in \mathbb{Z}_q^N$. We use $\mathbb{Z} \cap (-q/2, q/2]$ as a representative of $\mathbb{Z}_q$ for an integer q, and denote by $[a]_q$ the reduction of an integer a modulo q. For $a \in R$, we define $\|a\|_\infty$ as the ℓ_∞-norm of its coefficient vector. For a real number r, $\lfloor r \rceil$ denotes the nearest integer to r, rounding upwards in case of a tie. For a finite set S, we use $x \leftarrow \mathcal{U}(S)$ to denote the sampling x according to the uniform distribution over S.

2.2. Authenticated Shares in SPDZ

We review the definitions of authenticated share and authenticated triple used in the online phase of the SPDZ protocol [DPSZ12]. Let n be the number of parties involved in the multi-party computation protocol, and let α be the global MAC key such that the i-th party holds an additive share α_i, where $\sum_{i=0}^{n-1} \alpha_i = \alpha \pmod p$. For an input $x \in \mathbb{Z}_p$, an authenticated share $[\![x]\!]$ is defined as $(x_i, m_i)_{0\le i<n}$, where $\sum_{i=0}^{n-1} x_i = x \pmod p$ and $\sum_{i=0}^{n-1} m_i = \alpha \cdot x \pmod p$. A triple $([\![a]\!], [\![b]\!], [\![c]\!])$ is called an authenticated triple if $ab = c \pmod p$ holds. For a matrix $\mathbf{X} \in \mathbb{Z}_p^{d\times d}$, an authenticated share $[\![\mathbf{X}]\!]$ is defined as $(\mathbf{X}_i, \mathbf{M}_i)_{0\le i<n}$, where $\sum_{i=0}^{n-1} \mathbf{X}_i = \mathbf{X} \pmod p$ and $\sum_{i=0}^{n-1} \mathbf{M}_i = \alpha \cdot \mathbf{M} \pmod p$. A triple $([\![\mathbf{A}]\!], [\![\mathbf{B}]\!], [\![\mathbf{C}]\!])$ is called an authenticated matrix triple if $\mathbf{AB} = \mathbf{C} \pmod p$ holds. We note that the security of the MAC is determined by the size of p, so large prime numbers, such as 128-bit primes, are utilized in the SPDZ protocol.

2.3. Discrete Gaussian Distribution

We define the n-dimensional spherical Gaussian function $\rho : \mathbb{R}^n \to (0, 1]$ as $\rho(\mathbf{x}) := \exp(-\pi \cdot \mathbf{x}^\top \mathbf{x})$. In general, for a positive definite matrix $\mathbf{\Sigma} \in \mathbb{R}^{n\times n}$, we define the elliptical Gaussian function $\rho_{\sqrt{\mathbf{\Sigma}}} : \mathbb{R}^n \to (0, 1]$ as $\rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{x}) := \exp(-\pi \cdot \mathbf{x}^\top \mathbf{\Sigma}^{-1}\mathbf{x})$. Let $\Lambda \subseteq \mathbb{R}^n$ be a lattice and $\mathbf{c} \in \mathbb{R}^n$. The discrete Gaussian distribution $\mathcal{D}_{\mathbf{c}+\Lambda,\sqrt{\mathbf{\Sigma}}}$ is defined as a distribution over the coset $\mathbf{c} + \Lambda$, whose probability mass function is $\mathcal{D}_{\mathbf{c}+\Lambda,\sqrt{\mathbf{\Sigma}}}(\mathbf{x}) = \rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{x})/\rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{c}+\Lambda)$ for $\mathbf{x} \in \mathbf{c}+\Lambda$ where $\rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{c}+\Lambda) := \sum_{\mathbf{v}\in\mathbf{c}+\Lambda} \rho_{\sqrt{\mathbf{\Sigma}}}(\mathbf{v}) < \infty$. When $\mathbf{\Sigma} = \sigma^2 \cdot \mathbf{I}_n$ for $\sigma > 0$ where $\mathbf{I}_n$ is the n-dimensional identity matrix, then we substitute $\sqrt{\mathbf{\Sigma}}$ by σ in the subscript and refer to σ as the width parameter. For a polynomial $a \in R$, we denote by $a \leftarrow \mathcal{D}_{\mathbf{c}+\Lambda,\sqrt{\mathbf{\Sigma}}}$ if we sample its coefficient vector from $\mathcal{D}_{\mathbf{c}+\Lambda,\sqrt{\mathbf{\Sigma}}}$. Below, we provide useful lemmas for tail bounds of discrete Gaussian distributions.

Definition 1 (Definition 3.1 [MR07])**.** *For an n-dimensional lattice Λ and positive real $\varepsilon > 0$, the smoothing parameter $\eta_\varepsilon(\Lambda)$ is the smallest s such that $\rho_{1/s}(\Lambda^*\backslash\{0\}) \le \varepsilon$.*

Lemma 1 (Lemma 3.3 [MR07])**.** *For any n-dimensional lattice Λ and $\varepsilon > 0$,*

$$\eta_\varepsilon(\Lambda) \le \sqrt{\frac{\ln(2n(1 + 1/\varepsilon))}{\pi}} \cdot \lambda_n(\Lambda)$$

where $\lambda_n(\Lambda)$ is the smallest real number $r > 0$ such that $\dim(\mathrm{span}(\Lambda \cap \mathcal{B}(r))) = n$ and $\mathcal{B}(r)$ is the n-dimensional ball with radius r centered at the origin.

Lemma 2 (Lemmma 2.4 [Ban95])**.** *Let $\Lambda \subseteq \mathbb{R}^n$ be any n-dimensional lattice. For any $0 < \varepsilon < 1/3$, $\sigma \ge \eta_\varepsilon(\Lambda)$, and any $\mathbf{c} \in \mathbb{R}^n$,*

$$\Pr\left[\|\mathbf{x}\|_\infty > 6\sigma \mid \mathbf{x} \leftarrow \mathcal{D}_{\mathbf{c}+\Lambda,\sigma}\right] \le n \cdot 2^{-161}$$

2.4. Ring Learning With Errors

Let χ and ψ be distributions over R. The ring learning with errors (RLWE) problem, denoted as $\mathsf{RLWE}_{R,q,\chi,\psi}$, is to distinguish between the following two distributions.

$$\{(a, b) | a \leftarrow \mathcal{U}(R_q), b \leftarrow \mathcal{U}(R_q)\}$$
$$\{(a, b) | a \leftarrow \mathcal{U}(R_q), s \leftarrow \chi, e \leftarrow \psi, b = -a \cdot s + e \pmod q\}$$

We refer to a pair $(c_0, c_1) \in R_q^2$ as an RLWE encryption of μ under s if $c_0 = -c_1 s + \mu + e \pmod q$ holds for a small error term $e \in R$. Then, it is decryptable by computing $c_0 + c_1 s \approx \mu \pmod q$.

2.5. Gadget Decomposition

Below, we provide the definition of gadget decomposition and external product, which are key building blocks for

implementing nonlinear operations in RLWE-based homomorphic encryption schemes

Definition 2 (Gadget Decomposition)**.** *For a modulus q, a function $h : R_q \to R^L$ is called a gadget decomposition if there exists a fixed vector $\mathbf{g} = (g_0, g_1, \ldots, g_{L-1}) \in R_q^d$ and a real $B > 0$ such that the followings hold for any $a \in R_q$ and its decomposition $h(a) = (b_0, b_1, \ldots, b_{L-1})$.*

$$\langle h(a), \mathbf{g} \rangle = \sum_{i=0}^{L-1} b_i \cdot g_i = a \pmod{q} \quad \textit{and} \quad \|\mathbf{b}\|_\infty \leq B$$

We call $\mathbf{g}$ a *gadget vector* and $B > 0$ a bound of h.

Definition 3 (External Product)**.** *For $a \in R_q$ and $\mathbf{u} = (u_0, \ldots, u_{d-1}) \in R_q^L$, a binary operation external product $\boxdot : R_q \times R_q^L \to R_q$ is defined as follows.*

$$a \boxdot \mathbf{u} := \langle h(a), \mathbf{u} \rangle \pmod{q}$$

By an abuse of notation, we write $a \boxdot \mathbf{U} = (a \boxdot \mathbf{u}_0, a \boxdot \mathbf{u}_1) \in R_q^2$ for any $a \in R_q$ and $\mathbf{U} = [\mathbf{u}_0 | \mathbf{u}_1] \in R_q^{L \times 2}$.

The *key-switching* procedure is a major use case of the external product, which is used for implementing nonlinear HE operations. Given an RLWE ciphertext $(c_0, c_1) \in R^2$, which is an encryption of μ under the secret key s, we can generate another RLWE ciphertext (c_0', c_1') with a different secret key s' such that $c_0 + c_1 s \approx c_0' + c_1' s' \pmod{q}$. This is achieved by computing $(c_0', c_1') = (c_0, 0) + c_1 \boxdot \mathbf{U} \pmod{q}$, where $\mathbf{u}_0 = -s \cdot \mathbf{u}_1 + s' \cdot \mathbf{g} + \mathbf{e} \pmod{q}$ for some small error term $\mathbf{e}$. Then, (c_0', c_1') can be interpreted as the RLWE encryption of μ under the new secret key s', switching the secret key from s to s'.

In a real-world implementation, a popular choice for a gadget decomposition is the RNS (Residue Number System) gadget decomposition [HPS19]. Suppose that $q = \prod_{0 \leq i < L} q_i$ for some word-sized primes q_i's, where $0 \leq i < L$. Then, a gadget decomposition and a gadget vector are defined as $h(a) = ([a]_{q_0}, \ldots, [a]_{q_{L-1}})$ and $\mathbf{g} = (\widetilde{q}_0, \ldots, \widetilde{q}_{L-1})$, where $\widetilde{q}_i = q/q_i \cdot [(q/q_i)^{-1}]_{q_i}$. It follows that $\langle h(a), \mathbf{g} \rangle = a \pmod{q}$ and $\|h(a)\|_\infty < \max\{q_0, \ldots, q_{L-1}\}$, so h becomes a gadget decomposition over R_q. The computation of the external product involves two steps: decomposition and inner product. The decomposition step computes $h(a)$, and the inner product step computes $\langle h(a), \mathbf{u} \rangle$. The complexity of the decomposition step is $\widetilde{O}(LN)$ or $\widetilde{O}(L^2 N)$ depending on the algorithm used, while the inner product computation requires $\widetilde{O}(L^2 N)$. Thus, the overall complexity is $\widetilde{O}(L^2 N)$. For a detailed analysis, we refer to [KLSS23].

3. Generalized BFV for Large Primes

In this section, we present a variant of the BFV scheme [Bra12, FV12] that can efficiently handle large prime plaintext moduli. For the ordinary BFV scheme, the plaintext space is defined as R_p, so the plaintext modulus p is usually set to be much smaller than the ciphertext modulus q, as the available multiplicative depth is roughly determined by the ratio $\log(q)/\log(p)$. Conversely, using large primes for plaintext moduli results in an increase in the ciphertext modulus q, but as noted in Section 2.5, this leads to quadratic performance overhead because the gadget level L roughly follows $O(\log q)$.

To overcome this problem, we present a variant of the BFV scheme that can accommodate messages from a large prime field, following the idea of the previous work by Bootland et al. [BCIV20]. The core idea is to design a variant of the BFV scheme with a plaintext space of $R_{X^D - b} = \mathbb{Z}[X]/(X^N + 1, X^D - b)$ for some $D \mid N$ and an integer b, instead of $R_p = \mathbb{Z}[X]/(X^N + 1, p)$. Since $\mathbb{Z}[X]/(X^N + 1, X^D - b) = \mathbb{Z}[X]/(b^M + 1, X^D - b)$, where $M = N/D$, if we set $p = b^M + 1$, it follows that $R_{X^D - b} = \mathbb{Z}_p[X]/(X^D - b)$. This allows us to perform modular arithmetic over p using the plaintext space $R_{X^D - b}$. The advantage of this plaintext space is that the multiplicative depth is roughly determined by the ratio $\log q / \log b$, rather than $\log q / \log p$. Thus, by choosing a suitable b, we can construct an efficient HE scheme that can handle large prime plaintext moduli. For example, in our experiments, we set $b \approx 2^{16}$ for a prime $p \approx 2^{4096}$ with $M = 256$, which allows us to use a much smaller q compared to the ordinary BFV scheme.

We note that our variant of the BFV scheme can be viewed as a generalization of the ordinary BFV scheme, as the latter corresponds to the case when $M = 1$. Hence, we refer to it as the generalized BFV scheme. In the rest of this section, we provide more details about our generalized BFV scheme, including how we instantiate a SIMD-like packing method and investigate the automorphism group on $R_{X^D - b}$ for implementing homomorphic rotation operations.

3.1. Packing Method

Although the previous work [BCIV20] utilized the plaintext space $R_{X^D - b}$, its packing method was designed for handling complex-valued messages. Thus, our first goal is to instantiate a HElib-like packing method [HS15] over $R_{X^D - b} = \mathbb{Z}_p[X]/(X^D - b)$ to support messages from large prime fields rather than complex numbers. More precisely, we aim to pack a vector $\mathbf{m} \in \mathbb{Z}_p^D$ into a single polynomial $\mu \in R_{X^D - b}$ using the Chinese Remainder Theorem (CRT), so that we can perform homomorphic operations over $\mathbb{Z}_p$ in a SIMD manner. To achieve this, the polynomial $X^D - b$ must fully split modulo p, so we investigate the conditions on p, D, and b required for $X^D - b$ to split modulo p below. We note that all proofs in this subsection are deferred to Appendix B.

Lemma 3. *Let b, D, and M be positive integers such that $N = M \cdot D$, and $p = b^M + 1$ is a prime. If $2N$ divides $(p - 1)$, then $X^D - b$ splits in $\mathbb{Z}_p[X]$. Moreover, there exists an element $\xi \in \mathbb{Z}_p$ such that $\xi, \xi^{2M+1}, \ldots, \xi^{2(D-1)M+1}$ are distinct roots for $X^D - b = 0 \pmod{p}$ for $0 \leq i < D$.*

From the above lemma, we obtain the following isomor-

phism relations due to the CRT.

$$\mathbb{Z}_p[X]/(X^D - b) \cong \prod_{i=0}^{D-1} \mathbb{Z}_p[X]/(X - \xi^{2iM+1}) \cong \mathbb{Z}_p^D$$

Now, a packing strategy for a vector in $\mathbb{Z}_p^D$ into an element in R_{X^D-b} can be devised from the above relations. Briefly speaking, given a vector $\mathbf{m} \in \mathbb{Z}_p^D$, a degree $D-1$ polynomial $\mu \in R_{X^D-b}$ is computed, such that each evaluation value of μ at the roots $\{\xi^{2iM+1}\}_{0 \le i < D}$ is each entry of the vector. Then, the homomorphic operations such as addition and multiplication over R_{X^D-b} directly translates into the component-wise arithmetic in $\mathbb{Z}_p^D$, so we refer to D as the number of slots. Finding such a polynomial μ can be efficiently done by adapting the Number Theoretic Transform (NTT) algorithm.

For the purpose of homomorphic matrix multiplication, we need to introduce an additional homomorphic operation over R_{X^D-b} called *rotation*. The rotation operation ρ on a vector $\mathbf{m} = (m_0, \ldots, m_{D-1})$ is defined as $\rho(\mathbf{m}) := (m_1, \ldots, m_{D-1}, m_0)$, which cyclically shifts the entries of the input vector. In the following, we demonstrate how to instantiate the rotation operation using an automorphism evaluation over the plaintext space R_{X^D-b}, following the strategy outlined in [HS15]. First, we investigate when a map $\theta_i : X \mapsto X^i$ becomes an automorphism over R_{X^D-b}. For θ_i to be an automorphism, it should map the roots $\{\xi^{2jM+1}\}_{0 \le j < D}$ of $X^D - b$ modulo p to themselves. Applying θ_i to this set yields $\{\xi^{i(2jM+1)}\}_{0 \le j < D}$. To ensure that $\{\xi^{i(2jM+1)}\}_{0 \le j < D} = \{\xi^{2jM+1}\}_{0 \le j < D}$, each $i(2jM+1)$ must be of the form $2jM+1$, which implies $i = 1 \pmod{2M}$. Furthermore, if $i = 1 \pmod{2M}$, the elements $\xi^{i(2jM+1)}$ will not overlap, making θ_i an automorphism. Below, we show that the set of automorphisms $\{X \mapsto X^{2iM+1} \mid 0 \le i < D\}$ indeed forms a cyclic group.

Lemma 4. *Let $D, M \ge 2$ be power-of-2 integers such that $N/M = D$. Then, $G = \{2iM+1 \mid 0 \le i < d\}$ is a cyclic multiplicative subgroup of $\mathbb{Z}_{2N}^\times$ and $5^{M/2}$ is its generator.*

From the above lemma, the set $\{X \mapsto X^{2iM+1} \mid 0 \le i < D\}$ forms a cyclic group generated by the automorphism $\varphi : X \mapsto X^{5^{M/2}}$. We can now describe our packing method with cyclic rotation as follows.

Theorem 1. *Let b, D, and M be positive integers such that $N = MD$ and $p = b^M + 1$ is a prime. If $2N$ divides $p-1$, then $X^D - b$ splits modulo p, and $R_{X^D-b} = \mathbb{Z}_p[X]/(X^D - b) \simeq \mathbb{Z}_p^D$. Moreover, there exists $\xi \in \mathbb{Z}_p$ such that the mapping $\tau : f \mapsto \left(f(\xi), f(\xi^{5^{M/2}}), \cdots, f(\xi^{5^{(D-1)M/2}})\right)$, which maps $f \in \mathbb{Z}_p[X]/(X^D - b)$ to an element in $\mathbb{Z}_p^D$, is an isomorphism.*

This theorem implies that if we pack a vector in $\mathbb{Z}_p^D$ using the map τ^{-1}, we can perform rotation ρ by the automorphism $\varphi : X \mapsto X^{5^{M/2}}$. To be precise, if a message $\mathbf{m} = (m_0, m_1, \ldots, m_{D-1})$ is packed into a plaintext $\tau^{-1}(\mathbf{m}) = \mu(X)$, then $\varphi(\mu) = \mu(X^{5^{M/2}})$ corresponds to the rotated message $\rho(\mathbf{m}) = (m_1, \cdots, m_{D-1}, m_0)$, as illustrated in Fig. 1.

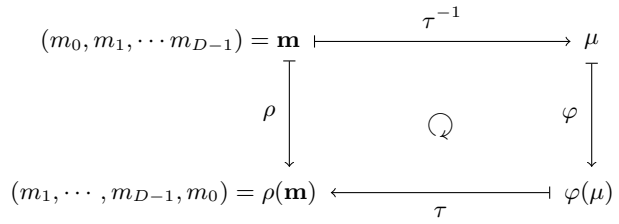

Figure 1. Rotation operation in R_{X^D-b}

We define the packing and unpacking algorithms for our scheme as follows. We note that τ and τ^{-1} can be computed with a time complexity of $O(D \log D)$ by modifying the NTT algorithm.

- $\underline{\mathtt{Pack}}(\mathbf{m}) \to \mu$: Given a message $\mathbf{m} = (m_0, m_1, \ldots, m_{D-1}) \in \mathbb{Z}_p^D$, return a plaintext $\mu = \tau^{-1}(\mathbf{m}) \in R_{X^D-b}$.
- $\underline{\mathtt{Unpack}}(\mu) \to \mathbf{m}$: Given a plaintext $\mu \in R_{X^D-b}$, return a message $\mathbf{m} = \tau(\mu) \in \mathbb{Z}_p^D$.

3.2. Scheme Description

We present our generalized BFV scheme that supports efficient arithmetic over large primes, built upon our new packing method. Our construction basically follows the construction of [BCIV20].

Setup. The algorithms for setup are given as follows:

- $\underline{\mathtt{Setup}}(1^\lambda) \to \mathsf{pp}$: Given a security parameter λ, choose the ring dimension N, the ciphertext modulus q, the secret key distribution χ, and the error distribution ψ. Choose a plaintext space R_{X^D-b}, and define $\Delta = \left\lfloor \frac{q}{X^D-b} \right\rfloor = \left\lfloor -\frac{q}{b^M+1}(X^{N-D} + bX^{N-2D} + \cdots + b^{M-1}) \right\rfloor \in R$. Choose a gadget vector $\mathbf{g} \in R_q^L$ and a gadget decomposition $h : R_q \to R^L$, with bound B. Return public parameters $\mathsf{pp} = (N, q, \chi, \psi, R_{X^D-b}, h, \mathbf{g}, B)$.
- $\underline{\mathtt{KeyGen}}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$: Sample $s \leftarrow \chi$, $a \leftarrow \mathcal{U}(R_q)$ and $e \leftarrow \psi$, and set $\mathsf{sk} = (1, s)$, and $\mathsf{pk} = (-as + e, a) \in R_q^2$. Return a key pair $(\mathsf{sk}, \mathsf{pk})$.
- $\underline{\mathtt{RelinKeyGen}}(\mathsf{sk}) \to \mathsf{rlk}$: Given a secret key $\mathsf{sk} = (1, s)$, sample $\mathbf{a} \leftarrow \mathcal{U}(R_q^L)$, and $\mathbf{e} \leftarrow \psi^L$. Return a relinearization key $\mathsf{rlk} = (-s \cdot \mathbf{a} + s^2 \cdot \mathbf{g} + \mathbf{e}, \mathbf{a})$.
- $\underline{\mathtt{RotKeyGen}}(\mathsf{sk}, r) \to \mathsf{rtk}_r$: Given a secret key $\mathsf{sk} = (1, s)$ and a rotation index r, sample $\mathbf{a} \leftarrow \mathcal{U}(R_q^L)$, and $\mathbf{e} \leftarrow \psi^L$. Return a rotation key $\mathsf{rtk}_r = (-s \cdot \mathbf{a} + \varphi^r(s) \cdot \mathbf{g} + \mathbf{e}, \mathbf{a})$.

While there is no need to generate the rotation key when the rotation index r is zero, rtk_0 will be used in the algorithm description for an easier notation. We remark that rotation with rtk_0 is essentially outputting the input ciphertext.

Encryption. To encrypt a plaintext $\mu \in R_{X^D-b}$, it needs to be represented as an element in R. There are multiple

possible representations, but we want it to have a small norm to minimize noise growth during encryption. We observe that every $\mu \in R_{X^D-b}$ can have a representation $[\mu]_R \in R$ such that $\|[\mu]_R\|_\infty \leq b$, leveraging the fact that $X^D = b \pmod{X^D - b}$, and we use this representation for the encryption algorithm. The algorithms for encryption and decryption are given as follows:

- $\underline{\mathtt{Enc}}(\mu, \mathbf{r}) \to \mathsf{ct}$: Given a public key pk, a plaintext $\mu \in R$, and a randomness triple $\mathbf{r} = (r_0, r_1, r_2) \in R^3$, return $\mathsf{ct} = r_0 \cdot \mathsf{pk} + (\Delta \cdot \mu + r_1, r_2) \in R_q^2$.
- $\underline{\mathtt{Dec}}(\mathsf{ct}) \to \mu$: Given a ciphertext $\mathsf{ct} = (c_0, c_1)$ and a secret key $\mathsf{sk} = (1, s)$, return a plaintext $\mu = \left\lfloor \frac{X^D-b}{q}(c_0 + c_1 \cdot s) \right\rceil \pmod{X^D - b}$.

We analyze the correctness of the encryption and decryption algorithms in Appendix B.4.

Homomorphic Operations. Below, we present algorithms for homomorphic operations. We incorporate a key-switching procedure for nonlinear homomorphic operations, such as multiplication and rotations.

- $\underline{\mathtt{Add}}(\mathsf{ct}, \mathsf{ct}') \to \mathsf{ct}''$: Given ciphertexts $\mathsf{ct}, \mathsf{ct}' \in R_q^2$, return $\mathsf{ct}'' = \mathsf{ct} + \mathsf{ct}' \pmod q$.
- $\underline{\mathtt{PMul}}(\mu, \mathsf{ct}) \to \mathsf{ct}'$: Given a plaintext $\mu \in R_{X^D-b}$ and a ciphertext $\mathsf{ct} \in R_q^2$, return $\mathsf{ct}' = [\mu]_R \cdot \mathsf{ct} \pmod q$.
- $\underline{\mathtt{Mul}}(\mathsf{ct}, \mathsf{ct}') \to \mathsf{ct}''$: Given ciphertexts $\mathsf{ct} = (c_0, c_1)$, $\mathsf{ct}' = (c_0', c_1')$ and a relinearization key rlk, let $d_0 = \left\lfloor \frac{X^D-b}{q} c_0 c_0' \right\rceil \pmod q$, $d_1 = \left\lfloor \frac{X^D-b}{q}(c_0 c_1' + c_1 c_0') \right\rceil \pmod q$ and $d_2 = \left\lfloor \frac{X^D-b}{q} c_1 c_1' \right\rceil \pmod q$. Return $\mathsf{ct}'' = (d_0, d_1) + d_2 \boxdot \mathsf{rlk} \pmod q$.
- $\underline{\mathtt{Rot}}(\mathsf{ct}, r) \to \mathsf{ct}'$: Given a ciphertext $\mathsf{ct} = (c_0, c_1)$, a rotation index r and the corresponding rotation key rtk_r, let $d_0 = \varphi^r(c_0)$, and $d_1 = \varphi^r(c_1)$. Return $\mathsf{ct}' = (d_0, 0) + d_1 \boxdot \mathsf{rtk}_r \pmod q$.

Noise Analysis. For the noise analysis, we use the notion of invariant noise introduced in [BCIV20].

Definition 4 (Invariant Noise)**.** *Let* $\mathsf{ct} = (c_0, c_1)$ *be a ciphertext with a secret key* $\mathsf{sk} = (1, s)$*. For* $\mu \in R$*, we define the invariant noise* $\mathsf{Err}(\mathsf{ct}, \mu)$ *as an element* $v \in K$ *with the smallest canonical norm, which satisfies the following for some* $I \in R$*:*

$$\frac{X^D - b}{q}(c_0 + c_1 s) = \mu + v + (X^D - b)I$$

The following lemma shows that the decryption works correctly if the canonical norm of the invariant noise is less than $1/2$.

Lemma 5 (Decryption Bound [BCIV20])**.** *Let* ct *be a ciphertext and* $\mu \in R$ *be a plaintext. If* $\|\mathsf{Err}(\mathsf{ct}, \mu)\|_\infty^{\mathsf{can}} < 1/2$*, then* $\mathtt{Dec}(\mathsf{sk}, \mathsf{ct}) = \mu \pmod{X^D - b}$.

We show the correctness of each homomorphic operation, along with the noise growth in Appendix B.4. We remark that the noise growth of homomorphic multiplication now depends on the size of b rather than p, as analyzed in Lemma 8.

4. Improved Matrix Multiplication

In this section, we present a new homomorphic square matrix multiplication algorithm, which can be utilized for authenticated matrix triple generation. Our algorithm improves upon the previous approach [HS18, JKLS18] by introducing a new matrix encoding method optimized for square matrix multiplication. As a result, we reduce the total time complexity by a factor of L, efficiently reducing the overhead of inner products performed during the external product.

Throughout this section, we use d to denote the matrix dimension, and the complexity of the algorithm is analyzed in terms of d. We first briefly review the previous approach [HS18, JKLS18] to homomorphic square matrix multiplication, and then demonstrate how our method improves performance, along with a detailed analysis.

4.1. Previous Approach

In this subsection, we review two previous approaches [HS18, JKLS18] on matrix-matrix multiplication.

Halevi-Shoup Method. Halevi and Shoup [HS18] introduced a method to homomorphically compute a linear transformation on an encrypted vector. For a matrix $\mathbf{A} = (a_{i,j}) \in \mathbb{Z}_p^{d \times d}$, it first defines a diagonal encoding map $\iota : \mathbb{Z}_p^{d \times d} \to (\mathbb{Z}_p^d)^d$ as

$$\iota : \mathbf{A} = (a_{i,j}) \mapsto ((a_{j,j+i})_{0 \leq j < d})_{0 \leq i < d}$$

where each index is computed modulo d. For simplicity, we denote $\iota(\mathbf{A}) = (\iota(\mathbf{A})_0, \ldots, \iota(\mathbf{A})_{d-1})$ where each $\iota(\mathbf{A})_i$ is a vector in $\mathbb{Z}_p^d$. Then, for a vector $\mathbf{v} \in \mathbb{Z}_p^d$, we have

$$\mathbf{A}\mathbf{v} = \sum_{k=0}^{d-1} \iota(\mathbf{A})_k \odot \rho^k(\mathbf{v}) \tag{1}$$

where ρ^k denotes the left rotation of the elements of a vector by k, and $\odot$ denotes the component-wise multiplication between vectors. To translate Eq. (1) into homomorphic operations, we need ciphertexts that encrypt $\iota(\mathbf{A})_0, \ldots, \iota(\mathbf{A})_{d-1}$, $\mathbf{v}$, and perform homomorphic rotations $\mathtt{Rot}(\cdot, k)$ on the encryption of $\mathbf{v}$ to obtain an encryption of $\rho^k(\mathbf{v})$. Additionally, the number of slots D should be set to d. We also note that this method can be trivially extended to encrypted matrix-matrix multiplication by applying Eq. (1) d times, once for each column of the matrix.

Jiang et al. Method. Jiang et al. [JKLS18] proposed an optimized matrix-matrix multiplication algorithm for cases where the number of slots is a multiple of d^2, focusing on small-dimensional matrix multiplication. For a matrix $\mathbf{A} = (a_{i,j}) \in \mathbb{Z}_p^{d \times d}$, consider its flattening map $\sigma : \mathbb{Z}_p^{d \times d} \to \mathbb{Z}_p^{d^2}$ as follows.

$$\sigma : \mathbf{A} = (a_{i,j}) \mapsto (a_{d \cdot i + j})_{0 \leq i,j < d}$$

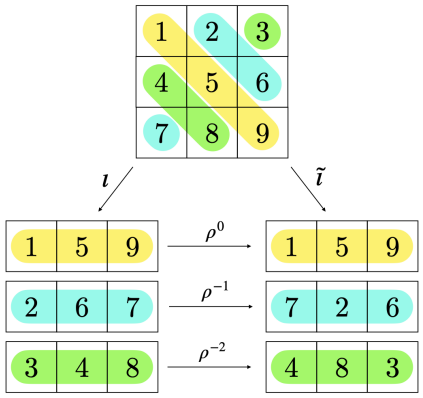

Figure 2. Comparison between diagonal encoding map ι and shifted diagonal encoding map $\widetilde{\iota}$.

In addition, it defines four $d^2 \times d^2$ permutation matrices $\mathbf{M}, \mathbf{N}, \mathbf{V}^{(k)}, \mathbf{W}^{(k)}$ with entries defined as follows.

$$\mathbf{M}_{d\cdot i+j,\ell} = \begin{cases} 1 & \text{if } \ell = d\cdot i + j + [i+j]_d \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{N}_{d\cdot i+j,\ell} = \begin{cases} 1 & \text{if } \ell = d\cdot [i+j]_d + j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{V}^{(k)}_{d\cdot i+j,\ell} = \begin{cases} 1 & \text{if } \ell = d\cdot i + j + [j+k]_d \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{W}^{(k)}_{d\cdot i+j,\ell} = \begin{cases} 1 & \text{if } \ell = d\cdot [i+k]_d + j \\ 0 & \text{otherwise} \end{cases}$$

Then, for any $d \times d$ matrices $\mathbf{A}, \mathbf{B}$, the following holds.

$$\sigma(\mathbf{AB}) = \sum_{k=0}^{d-1} (\mathbf{V}^{(k)}\mathbf{M}\sigma(\mathbf{A})) \odot (\mathbf{W}^{(k)}\mathbf{N}\sigma(\mathbf{B})) \tag{2}$$

For computing $\mathbf{V}^{(k)}\mathbf{M}\sigma(\mathbf{A}), \mathbf{W}^{(k)}\mathbf{N}\sigma(\mathbf{B})$, it uses the method by Halevi and Shoup discussed earlier.

4.2. Our Method

In the previous approaches, the main performance bottlenecks are key-switching procedures in homomorphic rotations. Hence, we focus on reducing the overhead of key-switching by developing a new algorithm for homomorphic matrix multiplication. For this, we start by defining a new encoding map, called the *shifted diagonal encoding map*. Our algorithm assumes that the number of slots is equal to the matrix dimension d, as in the Halevi-Shoup method. The shifted diagonal encoding map $\widetilde{\iota} : \mathbb{Z}_p^{d\times d} \to (\mathbb{Z}_p^d)^d$ is defined as follows

$$\widetilde{\iota} : \mathbf{A} = (a_{i,j}) \mapsto ((a_{j-i,j})_{0\le j<d})_{0\le i<d}$$

or equivalently, $\widetilde{\iota}_i(\mathbf{A}) = \rho^{-i}(\iota_i(\mathbf{A}))$. We denote $\widetilde{\iota}(\mathbf{A}) = (\widetilde{\iota}(\mathbf{A})_0, \cdots, \widetilde{\iota}(\mathbf{A})_{d-1})$. Then, the multiplication between $\mathbf{A} = (a_{i,j})$ and $\mathbf{B} = (b_{i,j})$ can be written as

$$\iota(\mathbf{A}\cdot\mathbf{B})_i = \sum_{k=0}^{d-1} \iota(\mathbf{A})_k \odot \rho^i(\widetilde{\iota}(\mathbf{B})_{i-k}) \tag{3}$$

since

$$\begin{aligned}\left(\sum_{k=0}^{d-1} \iota(\mathbf{A})_k \odot \rho^i(\widetilde{\iota}(\mathbf{B})_{i-k})\right)_j &= \sum_{k=0}^{d-1} \iota(\mathbf{A})_{k,j} \cdot \widetilde{\iota}(\mathbf{B})_{i-k,j+i} \\ &= \sum_{k=0}^{d-1} a_{j,j+k} \cdot b_{j+k,j+i} \\ &= (\mathbf{A}\cdot\mathbf{B})_{j,j+i} = \iota(\mathbf{A}\cdot\mathbf{B})_{i,j}.\end{aligned}$$

The translation of Eq. (3) in terms of homomrphic operation is as follows. We need encryptions of $\mathtt{Pack}(\iota(\mathbf{A})_i)$, denoted as $\mathsf{CT}_{\iota(\mathbf{A})} = (\mathsf{ct}_{\iota(\mathbf{A})_0}, \cdots \mathsf{ct}_{\iota(\mathbf{A})_{d-1}})$, and similarly encryptions of $\mathtt{Pack}(\widetilde{\iota}(\mathbf{B})_i)$, denoted as $\mathsf{CT}_{\widetilde{\iota}(\mathbf{B})} = (\mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_0}, \cdots, \mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_{d-1}})$. A naive way of computing Eq. (3) is to first compute rotations of $\mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_{i-k}}$ for all k as

$$\begin{aligned}&(b^{(i)}_{i-k,0}, b^{(i)}_{i-k,1}) = \mathtt{Rot}(\mathsf{ct}_{\widetilde{\iota}(\mathbf{B}),i-k}, i) \\ &= (\varphi^i(b_{i-k,0}), 0) + \varphi^i(b_{i-k,1}) \boxdot \mathsf{rtk}_i \pmod q\end{aligned}$$

where $\mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_{i-k}} = (b_{i-k,0}, b_{i-k,1})$. Then, we compute the sum of the product between $\mathsf{ct}_{\iota(\mathbf{A})_k}$ and $\mathtt{Rot}(\mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_{i-k}}, i)$ as

$$\begin{aligned}\mathsf{ct}_{\iota(\mathbf{AB})_i} = \sum_{k=0}^{d-1} \Bigg(&\left\lfloor \frac{X^D - b}{q} (a_{k,0} b^{(i)}_{i-k,0}, a_{k,0} b^{(i)}_{i-k,1} + b^{(i)}_{i-k,0} a_{k,1}) \right\rceil \\ &+ \left\lfloor \frac{X^D - b}{q} a_{k,1} b^{(i)}_{i-k,1} \right\rceil \boxdot \mathsf{rlk} \Bigg) \pmod q\end{aligned}$$

where $\mathsf{ct}_{\iota(\mathbf{A})_k} = (a_{k,0}, a_{k,1})$. This approach requires d^2 $\mathtt{Mul}$ operations and d^2 $\mathtt{Rot}$ operations for each, resulting in a total of $2d^2$ key-switching operations.

Optimization via Lazy Key-switching. We then observe that the overhead of key-switching can be significantly reduced by employing the well-known optimization technique called *lazy key-switching*. Let us explain this in detail. If we defer all the key-switching procedures for lazy key-switching, it can be rewritten as

$$\begin{aligned}(d'_0, d'_1, d'_2, d'_3) = \Bigg\lfloor \frac{X^D - b}{q} \Bigg(\sum_{k=0}^{d-1} \Big(&a_{k,0}\varphi^i(b_{i-k,0}), a_{k,1}\varphi^i(b_{i-k,0}), \\ &a_{k,0}\varphi^i(b_{i-k,1}), a_{k,0}\varphi^i(b_{i-k,1}) \Big) \Bigg) \Bigg\rceil \pmod q\end{aligned}$$

so that the following holds.

$$d'_0 + d'_1 s + d'_2 \varphi^i(s) + d'_3 s\varphi^i(s) \approx \Delta\cdot\mathtt{Pack}(\widetilde{\iota}(\mathbf{AB})_i) \pmod q.$$

Then, we apply key-switching to the ciphertext (d'_0, d'_1, d'_2, d'_3), changing the secret key from $(1, s, \varphi^i(s), s\varphi^i(s))$ to the original secret key $(1, s)$. More specifically, we compute

$$(d'_0, d'_1 + v_0) + v_1 \boxdot \mathsf{rlk} + d'_2 \boxdot \mathsf{rtk}_i \pmod q$$

for $(v_0, v_1) = d_3' \boxdot \mathsf{rtk}_i$. We briefly show the correctness below.

$$\begin{aligned}
&\langle (1, s), (d_0', d_1' + v_0) + v_1 \boxdot \mathsf{rlk} + d_2' \boxdot \mathsf{rtk}_i \rangle \\
&\approx d_0' + d_1' \cdot s + s \cdot v_0 + s^2 \cdot v_1 + d_2' \cdot \varphi^i(s) \\
&= d_0' + d_1' \cdot s + s \cdot (v_0 + s \cdot v_1) + d_2' \cdot \varphi^i(s) \\
&\approx d_0' + d_1' \cdot s + d_2' \cdot \varphi^i(s) + d_3' \cdot s\varphi^i(s) \pmod q
\end{aligned}$$

Consequently, we require 3 key-switching operations to homomorphically compute Eq. (3) for each $0 \le i < d$. Therefore, a total of $3d$ key-switching operations are needed for the entire matrix multiplication, which is a reduction from the $2d^2$ key-switching operations required for naive computation. The full algorithm for our encrypted matrix multiplication is provided in Algorithm 1, which defines the following homomorphic operation.

- $\underline{\mathtt{MatMul}}(\mathsf{CT}_{\iota(\mathbf{A})}, \mathsf{CT}_{\widetilde{\iota}(\mathbf{B})}) \to \mathsf{CT}_{\iota(\mathbf{C})}$: Given ciphertexts $\mathsf{CT}_{\iota(\mathbf{A})}$ and $\mathsf{CT}_{\widetilde{\iota}(\mathbf{B})}$, run Algorithm 1, and return the result $\mathsf{CT}_{\iota(\mathbf{C})}$.

Algorithm 1 New Matrix Multiplication `MatMul`

Input: $\mathsf{CT}_{\iota(\mathbf{A})}$, $\mathsf{CT}_{\widetilde{\iota}(\mathbf{B})}$, and $\mathsf{evk} = \{\mathsf{rlk}, \mathsf{rtk}_0, \ldots, \mathsf{rtk}_{d-1}\}$.
Output: $\mathsf{CT}_{\iota(\mathbf{C})}$ where $\mathbf{C} = \mathbf{AB}$.

for $0 \le i < d$ **do**
  $(d_0, d_1, d_2, d_3) \leftarrow (0, 0, 0, 0) \in R_q^4$
  for $0 \le k < d$ **do**
    $(a_0, a_1) \leftarrow \mathsf{ct}_{\iota(\mathbf{A})_k}$
    $(b_0, b_1) \leftarrow \mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_{i-k}}$
    $(d_0, d_1, d_2, d_3) \leftarrow (d_0, d_1, d_2, d_3) + \big(a_0\varphi^i(b_0), a_0\varphi^i(b_1), a_1\varphi^i(b_0), a_1\varphi^i(b_1)\big)$
  end for
  $(d_0', d_1', d_2', d_3') \leftarrow \left\lfloor \frac{X^D - b}{q}(d_0, d_1, d_2, d_3) \right\rceil \pmod q$
  `// key-switching from` $\varphi^i(s)$ `to` s
  $(u_0, u_1) \leftarrow d_2' \boxdot \mathsf{rtk}_i \pmod q$
  `// key-switching from` $s \cdot \varphi^i(s)$ `to` s^2
  $(v_0, v_1) \leftarrow d_3' \boxdot \mathsf{rtk}_i \pmod q$
  `// key-switching from` s^2 `to` s
  $(w_0, w_1) \leftarrow v_1 \boxdot \mathsf{rlk} \pmod q$
  $\mathsf{ct}_{\iota(\mathbf{C}),i} \leftarrow (d_0' + u_0 + w_0, d_1' + u_1 + v_0 + w_1) \pmod q$
end for

Noise Analysis. We analyze the noise growth of `MatMul` in Appendix B.5.

Complexity Analysis. Now, we provide a detailed complexity analysis of our method, together with a comparison to the previous method. For a fair comparison, we first explain the various optimization that can be made to previous approaches explained in Section 4.1. We first recall that a key-switching operation involves two steps: decomposition and inner product, as explained in Section 2.5.

In the method by Halevi and Shoup, we require d^2 `Rot` operations and d^2 `Mul` operations for each. Similar to our matrix multiplication method, we can also apply the lazy key-switching technique. However, since the rotation index of the summation term is different, we can only compute the relinearization in a lazy manner. Instead, [HS18] proposed a techinque called *hoisting*, which precomputes the decomposition of a ciphertext before applying several rotations. Using these optimizations, the number of decomposition becomes $2d$. However, the number of innner product operation still remains $d^2 + d$. Finally, we require $d - 1$ rotation keys, since we need to compute every rotation from 1 to $d - 1$. [1]

In the method by Jiang et al., we require d `Mul` operations and $2d$ linear transformations using Halevi-Shoup method, since $\mathbf{M}\sigma(\mathbf{A})$ and $\mathbf{N}\sigma(\mathbf{B})$ can be precomputed. A crucial observation made in [JKLS18] is that $\mathbf{V}^{(k)}$ and $\mathbf{W}^{(k)}$ has extremely sparse diagonal entries. Concretely, $\iota(\mathbf{V}^{(k)})$ has only two nonzero vectors $\iota(\mathbf{V}^{(k)})_k$ and $\iota(\mathbf{V}^{(k)})_{k-d}$, and $\iota(\mathbf{W}^{(k)})$ has only one nonzero vector $\iota(\mathbf{W}^{(k)})_{k \cdot d}$. Therefore, using Halevi-Shoup method, we can compute the linear transform $\mathbf{V}^{(k)}\mathbf{M}\sigma(\mathbf{A})$ and $\mathbf{W}^{(k)}\mathbf{N}\sigma(\mathbf{B})$ for all $0 \le k < d$ with $2d$ and d key-switchings repsectively. Moreover, we can further apply hoisting and lazy key-switching. This brings the total number of decomposition down to 3, and inner product to $3d+1$. Finally, since we need to compute rotation indices of the form $k, k - d, k \cdot d$ for every $0 \le k < d$, the total number of rotation keys required is $3d - 3$. We note that the method by Jiang et al. requires two multiplicative depths, unlike other methods that only require one.

In contrast, our method performs only three key-switchings at the end of the summation. Therefore, we require just $3d$ decompositions and inner products, achieving a quadratic speedup compared to the Halevi-Shoup method. Similar to the Halevi-Shoup method, we require $d - 1$ rotation keys and one multiplication depth, which is lower than the method by Jiang et al.

We summarize the complexity analysis in Table 1. For a fair comparison, we also analyze the complexity of inner products and decompositions in terms of d. For both the Halevi-Shoup method and our method, the number of slots D is set to d and $D = O(N)$, so both decomposition and inner products have a complexity of $\widetilde{O}(dL^2)$, as analyzed in Section 2.5, where L is the level of gadget decomposition. Meanwhile, for the method by Jiang et al., the number of slots is set to d^2, so both decomposition and inner products have a complexity of $\widetilde{O}(d^2L^2)$. We compute the complexity of decompositions and inner products based on this analysis in Table 1. Note that the complexity of basic ring arithmetic over R_q is $\widetilde{O}(d^3L)$ for all methods.

We note that in the previous method, the inner product is the most dominant factor, with a total complexity of $\widetilde{O}(d^3L^2)$. However, our method reduces this complexity to $\widetilde{O}(d^2L^2)$, making it quadratic in d. As a result, ring arithmetic now becomes the dominant factor in our method with complexity $\widetilde{O}(d^3L)$, which improves the total complexity by a factor of L from $\widetilde{O}(d^3L^2)$.

4.3. Batched Matrix Multiplication

As previously mentioned, our algorithm focused on the case where the matrix dimension d equals the number of

1. There are some methods to reduce the rotation keys, such as BSGS(Baby-Step, Giant-Step) algorithms. Since these algorithms degrade performance as a trade-off, we omit them from the comparison here.

	Jiang et al. [JKLS18]	Halevi-Shoup [HS18]	Ours
Decomp.	$\widetilde{O}(d^2L^2)$	$\widetilde{O}(d^2L^2)$	$\widetilde{O}(d^2L^2)$
Inner Product	$\widetilde{O}(d^3L^2)$	$\widetilde{O}(d^3L^2)$	$\widetilde{O}(d^2L^2)$
Ring Arithmetic	$\widetilde{O}(d^3L)$	$\widetilde{O}(d^3L)$	$\widetilde{O}(d^3L)$
Depth	1 PMul + 1 Mul	1 Mul	1 Mul
# Rotation Key	$3d-3$	$d-1$	$d-1$

TABLE 1. COMPARISON OF NUMBER OF BASIC OPERATIONS REQUIRED FOR MULTIPLYING TWO $d \times d$ MATRICES.

slots D. However, in practice, the number of slots can exceed the matrix dimension. A naive approach to address this issue would be to use sparse packing by encrypting a zero-padded version of the encoded matrix. The problem with this approach is that it incurs a blow-up in ciphertext size, because it essentially wastes the available slots. Instead, we propose batching multiple matrices into a single ciphertext, inspired by the batched multiplication strategy presented in [JKLS18]. With this technique, we can perform multiple matrix multiplication at once in batched form.

Let us elaborate our method in detail. Suppose that we are given ℓ matrices $\mathbf{A}^{(0)}, \ldots, \mathbf{A}^{(\ell-1)}$ of size $d \times d$, where $\ell = D/d$. Then, the diagonal encoding map ι and shifted diagonal encoding map $\widetilde{\iota}$ in Algorithm 1 can be naturally extended to the batched versions $\iota^{(\ell)} : (\mathbb{Z}_p^{d\times d})^\ell \to (\mathbb{Z}_p^D)^d$ and $\widetilde{\iota}^{(\ell)} : (\mathbb{Z}_p^{d\times d})^\ell \to (\mathbb{Z}_p^D)^d$ as follows.

$$\iota^{(\ell)} : (\mathbf{A}^{(0)}, \cdots, \mathbf{A}^{(\ell-1)}) \mapsto \left((a_{i,j+i}^{(0)}, \cdots, a_{i,j+i}^{(\ell-1)})_{0\le j<d}\right)_{0\le i<d}$$

$$\widetilde{\iota}^{(\ell)} : (\mathbf{A}^{(0)}, \cdots, \mathbf{A}^{(\ell-1)}) \mapsto \left((a_{j-i,j}^{(0)}, \cdots, a_{j-i,j}^{(\ell-1)})_{0\le j<d}\right)_{0\le i<d}$$

Now, using the analogous logic we used in the proof of Eq. (3), we obtain the following equality.

$$\iota(\mathbf{A}^{(0)}\mathbf{B}^{(0)}, \cdots, \mathbf{A}^{(\ell-1)}\mathbf{B}^{(\ell-1)})_i = \sum_{k=0}^{d-1} \iota(\mathbf{A}^{(0)}, \cdots, \mathbf{A}^{(\ell-1)})_k \odot \rho^{i\ell}(\widetilde{\iota}(\mathbf{B}^{(0)}, \cdots, \mathbf{B}^{(\ell-1)})_{i-k})$$

Then, we can compute ℓ matrix multiplications at once by calling a single instance of Algorithm 1 utilizing the equation above. This way, only $3d$ decomposition and inner product operations with d rotation keys are sufficient to perform ℓ matrix multiplications.

5. Protocol Specification

In this section, we present protocols for generating authenticated scalar and matrix triples. Our protocol is based on the TopGear [BCS19] framework and its variant [CKR+20] for matrix triples. We begin by explaining how we modify the existing PoPK protocol to support the generalized BFV scheme. Then, we introduce a protocol for authenticated triple generation, incorporating all the optimization techniques discussed in Section 3 and 4.

Π_{PoPK}

This protocol is run between n parties, where each of them acts as both a prover and a verifier simultaneously. The protocol is parameterized by integers U, V, and $\mathsf{Flag} \in \{\mathsf{Diag}, \perp\}$. A public parameter pp and an encryption key pk are pre-distributed to each party.

Sampling phase

1) If $\mathsf{Flag} = \perp$, party P_ℓ samples a message $\mathbf{x}_i^{(\ell)} \leftarrow \mathcal{U}(\mathbb{Z}_p^D)$ for $0 \le i < U$. If $\mathsf{Flag} = \mathsf{Diag}$, party P_ℓ samples a message $x_i^{(\ell)} \leftarrow \mathcal{U}(\mathbb{Z}_p)$ and sets $\mathbf{x}_i^{(\ell)} = (x_i^{(\ell)}, \ldots, x_i^{(\ell)}) \in \mathbb{Z}_p^D$ for $0 \le i < U$.
2) Party P_ℓ samples a randomness $\mathbf{r}_i^{(\ell)} \leftarrow \mathcal{D}^3_{\mathbb{Z}^N,\sigma_1}$ and a plaintext $\mu_{\mathbf{x}_i}^{(\ell)} \leftarrow \mathtt{RandPack}(\mathbf{x}_i^{(\ell)}, \mathfrak{s}_1)$ for $0 \le i < U$.
3) Party P_ℓ computes a ciphertext $\mathsf{ct}_i^{(\ell)} = \mathtt{Enc}(\mu_{\mathbf{x}_i}^{(\ell)}, \mathbf{r}_i^{(\ell)})$, broadcasts it, and sets $\mathsf{ct}_i = \sum_{\ell=0}^{n-1} \mathsf{ct}_i^{(\ell)}$ for $0 \le i < U$.

Commitment phase

1) If $\mathsf{Flag} = \perp$, party P_ℓ samples a pseudo-message $\mathbf{y}_j^{(\ell)} \leftarrow \mathcal{U}(\mathbb{Z}_p^D)$ for $0 \le j < V$. If $\mathsf{Flag} = \mathsf{Diag}$, party P_ℓ samples a pseudo-message $y_j^{(\ell)} \leftarrow \mathcal{U}(\mathbb{Z}_p)$ and sets $\mathbf{y}_j^{(\ell)} = (y_j^{(\ell)}, \ldots, y_j^{(\ell)}) \in \mathbb{Z}_p^D$ for $0 \le j < V$.
2) Party P_ℓ samples a pseudo-randomness $\mathbf{s}_j^{(\ell)} \leftarrow \mathcal{D}^3_{\mathbb{Z}^N,\sqrt{U+1}\cdot\sigma_2}$ and a pseudo-plaintext $\mu_{\mathbf{y}_j}^{(\ell)} \leftarrow \mathtt{RandPack}(\mathbf{y}_j^{(\ell)}, \sqrt{U+1}\cdot\mathfrak{s}_2)$ for $0 \le j < V$.
3) Party P_ℓ computes a commitment $\mathsf{comm}_j^{(\ell)} = \mathtt{Enc}(\mu_{\mathbf{y}_j}^{(\ell)}, \mathbf{s}_j^{(\ell)})$, and broadcasts it for $0 \le j < V$.

Challenge phase

1) Parties call $\mathcal{F}_{\mathsf{Rand}}$ to obtain a challenge $w_{i,j}$ for $0 \le i < U$ and $0 \le j < V$.
2) If $\mathsf{Flag} = \perp$, each challenge is sampled from $\{1, X, X^2, \ldots, X^{2N-1}\}$. If $\mathsf{Flag} = \mathsf{Diag}$, each challenge is sampled from $\{1, X^D, X^{2D}, \ldots, X^{2N-D}\}$.

Response phase

1) Party P_ℓ computes $\mu_{\mathbf{z}_j}^{(\ell)} = \mu_{\mathbf{y}_j}^{(\ell)} + \sum_{i=0}^{U-1} w_{i,j} \cdot \mu_{\mathbf{x}_i}^{(\ell)}$, $\mathbf{t}_j^{(\ell)} = \mathbf{s}_j^{(\ell)} + \sum_{i=0}^{U-1} w_{i,j} \cdot \mathbf{r}_i^{(\ell)}$, and broadcasts a response $\mathsf{resp}_j^{(\ell)} = (\mu_{\mathbf{z}_j}^{(\ell)}, \mathbf{t}_j^{(\ell)})$ for $0 \le j < V$.

Verification phase

1) The parties compute $\mathsf{comm}_j = \sum_{\ell=0}^{n-1} \mathsf{comm}_j^{(\ell)}$, $\mathbf{t}_j = \sum_{\ell=0}^{n-1} \mathbf{t}_j^{(\ell)}$, and $\mu_{\mathbf{z}_j} = \sum_{\ell=0}^{n-1} \mu_{\mathbf{z}_j}^{(\ell)}$ for $0 \le j < V$.
2) The parties check whether $\mathtt{Enc}(\mu_{\mathbf{z}_j}, \mathbf{t}_j) = \mathsf{comm}_j + \sum_{i=0}^{U-1} w_{i,j} \cdot \mathsf{ct}_i$, and the following inequalities hold for $0 \le j < V$.
$$\left\|\mu_{\mathbf{z}_j}\right\|_\infty < B_{\mathbf{z}} = 6(b+1)n(U\cdot\mathfrak{s}_1 + \sqrt{U+1}\cdot\mathfrak{s}_2)$$
$$\|\mathbf{t}_j\|_\infty < B_{\mathbf{t}} = 6n(U\cdot\sigma_1 + \sqrt{U+1}\cdot\sigma_2)$$
3) If $\mathsf{Flag} = \mathsf{Diag}$ then additionally check that $\mu_{\mathbf{z}_j}$ is a diagonal plaintext for $0 \le j < V$.
4) If all checks pass, the parties accept otherwise they reject.

Figure 3. Protocol for proof of plaintext knowledge

5.1. Proof of Plaintext Knowledge

A proof of plaintext knowledge (PoPK) protocol is essential for achieving adaptive security in the generation of authenticated triples, as it proves the well-formedness of the input ciphertexts. Its basic structure is a 3-move sigma protocol consisting of five phases: sampling, commitment, challenge, response, and verification. We provide a simplified overview of the PoPK protocol. In the sampling phase, a prover generates a ciphertext $\mathsf{ct} = \mathtt{Enc}(\mu_{\mathbf{x}}, \mathbf{r})$ with the input message $\mathbf{x}$ by sampling a plaintext $\mu_{\mathbf{x}}$ and encryption ran-

domness $\mathbf{r}$. In the commitment phase, the prover generates random masks $\mu_{\mathbf{y}}$ and $\mathbf{s}$ for the plaintext $\mu_{\mathbf{x}}$ and encryption randomness $\mathbf{r}$, and sends a commitment $\mathsf{comm} = \mathtt{Enc}(\mu_{\mathbf{y}}, \mathbf{s})$ to the verifier. After that, in the challenge phase, the verifier replies with a random challenge w. Subsequently, the prover responds with $\mu_{\mathbf{z}} = \mu_{\mathbf{y}} + w \cdot \mu_{\mathbf{x}}$ and $\mathbf{t} = \mathbf{s} + w \cdot \mathbf{r}$ to the verifier. Finally, the verifier runs verification algorithms to determine the validity of the ciphertext ct.

In the actual protocol, several modifications are added. First, since there are a total of n parties involved, the protocol is adapted to an n-prover version. Secondly, due to the limited size of the challenge sets, the sigma protocol must be repeated several times to achieve a reasonable security level, and this repetition count is denoted as V. Finally, to utlize the amortization technique in [BCS19], there are usually multiple ciphertexts as input, with the number of input ciphertexts denoted by U. A detailed description of our PoPK protocol Π_{PoPK} is presented in Fig. 3.

Simulatability. We first demonstrate how we achieve simulatability with our generalized BFV scheme. In the PoPK protocol, the response $\mathsf{resp} = (\mu_{\mathbf{z}}, \mathbf{t})$ may contain partial information about the plaintext $\mu_{\mathbf{x}}$ and the encryption randomness $\mathbf{r}$, which should be kept secret to hide the input message $\mathbf{x}$. Therefore, it must be simulatable without knowledge of the input message $\mathbf{x}$ to ensure that no information about the input messages is leaked. Achieving simulatability incurs paramter overhead, because the proven upper bounds for encryption noise are determined by $\mu_{\mathbf{z}}$ and $\mathbf{t}$. This results in an increase in the ciphertext modulus q, which increases the overall communication cost compared to the semi-honest setting. This overhead can be measured by the size ratio between $(\mu_{\mathbf{z}}, \mathbf{t})$ and $(\mu_{\mathbf{x}}, \mathbf{r})$, referred to as the *soundness slack*. Thus, achieving a small soundness slack is crucial for overall performance.

In [BCS19], the noise flooding method is used to achieve simulatability, which incurs exponential soundness slack for both $\mathbf{r}$ and $\mu_{\mathbf{x}}$. The exponential soundness slack for $\mathbf{r}$ is reduced to a constant factor in [KLSS23] by introducing the Hint-MLWE problem, which proves that if both $\mathbf{r}$ and $\mathbf{s}$ are sampled from a discrete Gaussian distribution, $\mathbf{t}$ is computationally indistinguishable from a simulated one. For $\mu_{\mathbf{x}}$, the exponential soundness slack is partially addressed in [CKR+20] for BFV ciphertexts when $p \mid q$, i.e., $\Delta = q/p$.

However, in our generalized BFV scheme, the scaling factor Δ is of the form $\frac{q}{X^d - b}$, making the approach in [CKR+20] inapplicable. To achieve simulatability for the plaintext part with constant overhead, we adapt the recently proposed technique by Hwang et al. [HSS24], called *randomized packing*. In [HSS24], it is used to achieve simulatability in the proof of opening knowledge protocol for the Ajtai commitment scheme. However, we observe that the same technique can be utilized in the PoPK protocol for our generalized BFV scheme. The randomized packing is defined as follows.

- $\underline{\mathtt{RandPack}}(\mathbf{m}, \mathfrak{s}) \to \mu$: Given a message $\mathbf{m} \in \mathbb{Z}_p^D$ and a positive real $\mathfrak{s} > 0$, return $\mu \in R$, which is sampled from $\mathcal{D}_{[\mathtt{Pack}(\mathbf{m})]_R + P\mathbb{Z}^N, \mathfrak{s}P}$, where $P \in \mathbb{Z}^{N \times N}$ is the negacyclic matrix corresponding to $X^D - b$, and $[\mathtt{Pack}(\mathbf{m})]_R \in \mathbb{Z}_p^N$ is interpreted as a coefficient vector.

We first note that $\mathtt{Unpack}(\mathtt{RandPack}(\mathbf{x}, \mathfrak{s})) = \mathbf{x}$ is preserved for any message $\mathbf{x} \in \mathbb{Z}_p^D$, allowing $\mathtt{RandPack}$ to be used in place of $\mathtt{Pack}$ when packing messages into plaintexts. The core idea of $\mathtt{RandPack}$ is to randomize the output plaintext so that it follows a discrete Gaussian distribution. By generating both $\mu_{\mathbf{x}}$ and $\mu_{\mathbf{y}}$ using randomized packing, the convolution lemma from [Pei10] can be applied to simulate $\mu_{\mathbf{z}} = \mu_{\mathbf{y}} + w \cdot \mu_{\mathbf{x}}$, similar to the Hint-MLWE-based analysis for $\mathbf{t} = \mathbf{s} + w \cdot \mathbf{r}$ in [KLSS23]. We also note that the size of the randomized plaintext is determined by the width parameter $\mathfrak{s}$ in $\mathtt{RandPack}$. Since we can set similarly sized width parameters for $\mu_{\mathbf{x}}$ and $\mu_{\mathbf{y}}$, this allows to decrase the soundness slack for $\mu_{\mathbf{x}}$ from an exponential scale to a constant scale.

Handling Diagonal Plaintexts. In the PoPK protocol for authenticated triples, an additional step is required to prove a special structure of plaintexts known as diagonal plaintexts, where each slot component contains the same element. This case is marked in Fig. 3 by setting $\mathsf{Flag} = \mathsf{Diag}$. For an ordinary BFV scheme, diagonal plaintexts are simply constant polynomials, but in our generalized BFV scheme, the structure of diagonal plaintexts changes. Specifically, they must be polynomials in X^D to ensure invariance under the automorphism φ, which implies that all slots consist of the same value. Our PoPK protocol must preserve this structure when proving knowledge of diagonal plaintexts. To achieve this, we sample the mask $\mu_{\mathbf{y}}$ from diagonal plaintexts and set the challenge w to be a power of X^D. This modification is illustrated in Fig. 3. We note that this modification effectively reduces communication costs compared to the previous PoPK protocol, because we can set the repetition count V much lower due to the increased size of the challenge set. In the previous PoPK, the challenge set for diagonal plaintexts was limited to $\{0, 1\}$, with a size of 2. In contrast, our PoPK expands this to $\{1, X^D, \ldots, X^{2N-D}\}$, with a size of $2M$. This larger challenge set allows for a reduced repetition count V while maintaining the same security level.

Security. We prove the security of our PoPK protocol Π_{PoPK} in Appendix B.6, following the definition of security for n-party PoPK described in [BCS19].

5.2. Generation of Authenticated Triples

We present our authenticated triple generation protocol Π_{Triple} in Fig. 4, which incorporates the generalized BFV scheme discussed in Section 3, the optimized homomorphic matrix multiplication algorithm in Section 4, and our PoPK protocol Π_{PoPK} tailored for the generalized BFV scheme. Compared to prior work [CKR+20], which utilizes the ordinary BFV scheme and the matrix multiplication algorithm by Jiang et al. [JKLS18], we have modified the protocol to accommodate these changes. For matrix triples, we assume that the matrix dimension $d = D$, but it can be easily generalized to support smaller dimensions such that $d \mid D$

using the batching technique described in Section 4.3. In this case, the parameter U is adjusted to $2dT$, where $T \mid D/d$, and the number of rotation keys is reduced to $d-1$. We omit the security proof for Π_{Triple} as it is identical to the proofs in [BCS19, CKR+20], except for the noise bound analysis.

Key Generation. For an authenticated triple generation protocol, previous work [BCS19, CKR+20] assumes the existence of an ideal key generation functionality, deferring detailed instantiation as an independent research topic, such as in [RST+22]. Similarly, we also assume the existence of such functionality $\mathcal{F}_{\mathsf{KeyGen}}$ for our triple generation protocol, as described in Fig. 5. In the functionality $\mathcal{F}_{\mathsf{KeyGen}}$, it not only generates the shared public key pk but also the shared evaluation key evk, which includes the relinearization key rlk for homomorphic multiplication and the rotation key rtk_r for homomorphic rotation. As analyzed in Section 4, our functionality generates $d-1$ rotation keys, whereas prior work [CKR+20] generates $3d-3$ rotation keys for matrix multiplications, as it utilizes the algorithm in [JKLS18]. Thus, our new matrix multiplication algorithm reduces the number of rotation keys by one-third.

Elimination of Reshare. In our protocol, we eliminate all occurrences of the reshare functionality by adopting the approach from [CKR+20]. In [DPSZ12, BCS19], the reshare functionality is utilized, which converts a ciphertext into secret shares of its message and a fresh encryption of it. This optimization allows for more compact parameter settings where the ciphertext modulus supports just one multiplicative depth, resulting in reduced communication costs for scalar triple generation. However, in [CKR+20], this functionality is not used for matrix triple generation, because it rather degrades performance, given that the matrix multiplication algorithm [JKLS18] used in [CKR+20] consumes multiple depths.

Although our matrix multiplication algorithm consumes only one multiplicative depth, we choose not to use the reshare functionality because the overhead for additional multiplicative depth is particularly low in our generalized BFV scheme. As analyzed in Section 3, noise growth in our generalized BFV scheme is proportional to b, rather than p as in the ordinary BFV scheme. This difference significantly reduces the increase in ciphertext modulus q due to the additional depth. Based on this observation, we completely eliminate all occurrences of the reshare functionality in our protocol by setting the ciphertext modulus to support the additional multiplicative depth, which provides better performance.

Noise Bound. In the protocol Π_{Triple}, authenticated scalar and matrix triples are generated through homomorphic operations and then distributed to each party via the distributed decryption protocol described in Fig. 6. To ensure the correctness of the protocol, we need to analyze an upper bound for ciphertext noise to guarantee that distributed decryption correctly outputs the results. Among all the ciphertexts, the noise of $\mathsf{CT}_{\alpha\cdot\iota(\mathbf{C}_i)}$ is the most dominant. Therefore, we focus on analyzing the noise upper bound for these ciphertexts.

We provide a detailed analysis in Appendix B.7. The noise for $\mathsf{CT}_{\alpha\cdot\iota(\mathbf{C}_i)}$ is bounded as follows, which becomes the upper bound B_{Dec} for ciphertext noise in Π_{Triple}.

$$\|\mathsf{Err}(\mathsf{ct}_{\alpha\cdot\iota(\mathbf{C}_i)_j}, \iota(\mathbf{C}_i)_j)\|_\infty^{\mathsf{can}} \lesssim \frac{d(b+1)^3}{q} \cdot 8N^4\sqrt{3N} \cdot B_{\mathbf{z}} = B_{\mathsf{Dec}}$$

To ensure correct decryption in Π_{DDec}, the condition $B_{\mathsf{Dec}} \cdot 2^{\lambda_{\mathsf{DDec}}} < 1/2$ must hold, as stated in Lemma 5. This implies that the ciphertext modulus q should satisfy the following inequality: $q \gtrsim d(b+1)^3 \cdot 16N^4\sqrt{3N} \cdot B_{\mathbf{z}} \cdot 2^{\lambda_{\mathsf{DDec}}}$.

6. Experimental Results

We present the concrete parameters and experimental results of our new matrix triplet generation protocol. We implemented our generalized BFV scheme and improved homomorphic matrix multiplication algorithm on top of the Lattigo library [MBTPH20] Our source code is available at https://github.com/SNUCP/matrix-triplet. All experiments were conducted on a machine with an Intel(R) Xeon(R) Platinum 8268 CPU at 2.90 GHz and 378 GB RAM, using a single thread. We first provide the concrete parameter settings used in our experiments and then analyze the costs for setup, communication, and computation.

6.1. Paramseter Setting

For the parameters of Π_{Triple}, we first set the security parameters λ_{Snd} and λ_{Sim} to 128, and λ_{DDec} to 80, as in the previous work [CKR+20]. Next, we assume the matrix dimension is given as $d = 128$ for the comparison with the previous work [CKR+20]. When the number of slots D exceeds $d = 128$, we use the batched matrix multiplication algorithm described in Section 4.3. For the parameter T, we set $T = D/d$ for matrix triples, which is the smallest values for T, then U is set to $2dT = 2D$.

For the parameters of Π_{PoPK}, we set the values of $\mathfrak{s}_1$, $\mathfrak{s}_2$, σ_1, and σ_2 as follows so that they satisfy the conditions for simulatability.

$$\mathfrak{s}_1 = \sqrt{2} \cdot \frac{b+1}{b-1} \cdot B_\eta, \quad \mathfrak{s}_2 = \sqrt{2V} \cdot \frac{b+1}{b-1} \cdot B_\eta$$
$$\sigma_1 = 2\sqrt{2} \cdot B_\eta, \quad \sigma_2 = 2\sqrt{2V} \cdot B_\eta$$

The upper bound B_η for the smoothing parameter is set to $\sqrt{\frac{\ln(2N(1+2^{\lambda_{\mathsf{Sim}}}))}{\pi}}$, following Lemma 1.

For the generalized BFV parameters, we prepare three sets: Params I, II, and III, as shown in Table 2. Params I is optimized for performance, Params II is optimzed for setup cost, and Params III corresponds to ordinary BFV for comparison with our generalized BFV. For Params I and II, we choose parameters such that $b \approx 2^{64}$ for Params I and $b \approx 2^{16}$ for Params II, for each field size $\log p$, so that $p = b^M + 1$ is a prime number satisfying the condition in Lemma 3. Then, we set $q \geq d(b+1)^3 \cdot 16N^4\sqrt{3N} \cdot B_{\mathbf{z}} \cdot 2^{\lambda_{\mathsf{DDec}}}$ following the analysis in Section 5.2. For the key-switching procedure, we use the RNS gadget decomposition described in Section 2.5. Specifically, we set $q = \prod_{0 \leq i < L} q_i$, where

$\log p$	Params I					Params II					Params III		
	$\log q$	N	L	D	b	$\log q$	N	L	D	b	$\log q$	$N = D$	L
128	426	2^{14}	7	2^{13}	$2^{64} - 3072$	230	2^{13}	4	2^{10}	$2^{16} - 196$	686	2^{15}	12
256				2^{12}	$2^{64} - 64$				2^{9}	$2^{16} - 22$	1202	2^{16}	20
512				2^{11}	$2^{64} - 428$				2^{8}	$2^{16} - 72$	2231	2^{17}	37
1024				2^{10}	$2^{64} - 8$				2^{7}	$2^{16} - 28$	4279	2^{18}	71
2048				2^{9}	$2^{64} - 22$				2^{6}	$2^{16} - 190$	-		
4096				2^{8}	$2^{64} - 56$				2^{5}	$2^{16} - 288$			

TABLE 2. PARAMETER SETS FOR BENCHMARKS

q_i's are prime numbers, each sized between 50 and 60 bits. Finally, the ring degree N is chosen to ensure the hardness of RLWE when the key distribution $\chi = \mathcal{U}(R_3)$. According to the estimation from [APS15], Params I provides 118-bit security, and Params II, III provide 128-bit security.

6.2. Setup Cost

Params I	Params II	Prev [CKR$^+$20]
1.45 GB	0.22 GB	27.4 GB

TABLE 3. SETUP COSTS FOR EACH PARAMETER.

We first analyze the setup cost for authenticated triple generation, which corresponds to the initialization phase of Π_{Triple}. In this phase, each party receives a shared public key pk and an evaluation key $\mathsf{evk} = \mathsf{rlk}, \mathsf{rtk}_1, \cdots, \mathsf{rtk}_{d-1}$ from the key generation functionality $\mathcal{F}_{\mathsf{KeyGen}}$. Thus, the total setup cost is measured by the combined size of pk and evk.

The public key pk consists of a pair of R_q elements, so its size is calculated as $2N/8 \cdot \log q$ bytes. For rlk and each rtk_i, which are pairs of R_q^L elements, their size is computed as $2LN/8 \cdot \log q$ bytes each. In Table 3, we present the total setup cost for each parameter set, together with the setup cost of the previous work [CKR$^+$20]. For the previous work [CKR$^+$20], we estimate the cost based on generating $3d - 3$ rotation keys, as it employs the matrix multiplication algorithm from [JKLS18]. Params I provides about 19 times smaller setup cost, and Params II offers 125 times smaller setup cost compared to the previous work [CKR$^+$20]. We attribute these reductions in setup costs to both our generalized BFV scheme and the new matrix multiplication algorithm. As analyzed in Section 3, the generalized BFV scheme allows for a much smaller ciphertext modulus q relative to a large plaintext modulus p. This not only reduces $\log q$, but also decreases L and N. Additionally, the new matrix multiplication algorithm requires only $d - 1$ rotation keys, compared to $3d - 3$ in the previous work. These improvements collectively result in a significant reduction in setup costs. In particular, for Params II, we minimized the size of $\log q$ by choosing a smaller b compared to Params I. This results in about 6.6 times smaller key sizes compared to Params I, which may be useufl for less powerful machines, such as IoT devices.

6.3. Communication Cost

	Params I	Params II	Prev [CKR$^+$20]
Init	121 MB	16.6 MB	816 MB
Matrix triple	13.6 MB	29.4 MB	12.5 MB

TABLE 4. COMMUNICATION COSTS FOR EACH PARAMETER

Next, we analyze the communication cost of Π_{Triple} when generating 128×128 dimensioned matrix triples, which is measured by the size of data sent by each party. First, we examine the communication cost of the PoPK protocol Π_{PoPK}, which is frequently used as a sub-protocol in Π_{Triple}. In the sampling phase, each party broadcasts U elements in R_q^2, resulting in a size of $\frac{2UN}{8} \cdot \log q$ bytes. For the commitment phase, each party broadcasts V elements in R_q^2, leading to a size of $\frac{2VN}{8} \cdot \log q$ bytes. Finally, in the response phase, each party broadcasts V pairs in R, bounded by $B_{\mathbf{z}}$ and $B_{\mathbf{t}}$ respectively, resulting in a size of $\frac{VN}{8} \cdot (\log B_{\mathbf{t}} + \log B_{\mathbf{z}})$ bytes. To summarize, each invocation of Π_{PoPK} requires a communication cost of $\frac{2(U+V)N}{8} \cdot \log q + \frac{VN}{8} \cdot (\log B_{\mathbf{t}} + \log B_{\mathbf{z}})$ bytes.

Now, we analyze the communication cost in Π_{Triple} based on the analysis for Π_{PoPK}. In the initialization phase of Π_{Triple}, where it collectively generates the encryption ct_α of the MAC key α, Π_{PoPK} is invoked with parameters $U = 1$ and $V = \lceil (\lambda_{\mathsf{Snd}}+)/2M \rceil$, where $M = N/D$. For the matrix triple phase of Π_{Triple}, Π_{PoPK} is invoked with parameters $U = 2dT$ and $V = \lceil (\lambda_{\mathsf{Snd}}+)/2N \rceil$, and it ends with the distributed decryption protocol Π_{DDec} for $2U$ ciphertexts, adding a communication cost of $\frac{4UN}{8} \cdot \log q$ bytes.

In Table 4, we summarize the communication costs for the initialization and matrix triple phases. For the matrix triple phase, we report the amortized cost, which is obtained by dividing the total cost by the number T of batched matrices, as done in previous work [CKR$^+$20]. For the costs of previous work [CKR$^+$20], we measure the cost of initialization based on our analysis as they did not analyze the initialization phase. The cost for the matrix triple phase is directly taken from the analysis in [CKR$^+$20].

$\log p$	Params I		Params II		Params III	
	Unit	Amortized	Unit	Amortized	Unit	Amortized
128	56.62ms	6.91μs	12.72ms	12.22μs	0.25s	7.85μs
256	56.58ms	13.81μs	12.76ms	24.91μs	1.34s	20.52μs
512	56.53ms	27.60μs	12.75ms	49.79μs	7.62s	58.15μs
1024	56.49ms	55.17μs	12.74ms	99.57μs	53.22s	203.03μs
2048	56.46ms	110.27μs	12.74ms	199.03μs	-	-
4096	56.33ms	220.05μs	12.75ms	398.49μs	-	-

TABLE 5. BENCHMARK RESULTS FOR `Mul`. THE AMORTIZED COSTS ARE OBTAINED FROM DIVIDING THE UNIT COSTS BY THE NUMBER OF SLOTS D.

For the initialization phase, Params I achieves about 6.7 times smaller costs, and Params II achieves about 49 times smaller costs compared to the previous work. This saving is particularly meaningful for the initialization phase, where communication overhead dominates over computational overhead since it does not perform any nonlinear homomorphic operations. We attribute these savings to our optimization for diagonal plaintexts in Appendix B.6, which reduces the size of V by a factor of $\log M$. Since the parameter U is fixed at 1 during the initialization phase, V becomes the most crucial factor in determining the communication cost. Consequently, our optimization leads to a significant reduction in communication cost during the setup phase.

For matrix triples, our approach incurs higher costs compared to the previous work because, in the matrix triple phase, U is usually set to be much larger than V. In this case, the generalized BFV scheme rather increases the cost since the overhead for distributed decryption is multiplied by M. However, we note that this does not lead to significant performance degradation in the matrix triple phase, as computational overhead is the most dominant factor due to homomorphic matrix multiplications. For example, in the benchmarks of [CKR$^+$20], the communication overhead typically accounts for about one percent of the total elapsed time in a single-thread setting.

6.4. Computation Cost

$\log p$	Params I		Params II	
	Unit	Amortized	Unit	Amortized
128	47.77s	0.75s	12.21s	1.53s
256	47.64s	1.49s	12.31s	3.08s
512	48.34s	3.02s	12.31s	6.16s
1024	48.45s	6.06s	12.30s	12.30s
2048	47.83s	11.06s	-	-
4096	48.23s	24.12s	-	-

TABLE 6. BENCHMARK RESULTS FOR `MatMul` WHEN $d = 128$. THE AMORTIZED COSTS ARE OBTAINED FROM DIVIDING THE UNIT COSTS BY THE NUMBER T OF BATCHED MATRICES.

d	Params I	Prev [CKR$^+$20]	Speedup
128	0.99s	34.25s	34.5×
256	6.84s	207.14s	30.28×
512	48.62s	1459.63s	30.02×

TABLE 7. BENCHMARK RESULTS FOR AUTHENTICATED MATRIX TRIPLE GENERATIONS WHEN $\log p = 128$.

At last, we analyze the computation cost for Π_{Triple}, especially the cost of homomorphic operations in generating scalar and matrix triples.

Scalar Triples. For scalar triples, we present the effect of the generalized BFV scheme when generating authenticated triples for large prime fields, extending the limits of field size to 4096 bits. Since authenticated triple generation requires only two `Mul` operations, we measure the performance of `Mul` for each parameter set in Table 5. For the unit cost of `Mul`, which generates D scalar triples, Params I and II dominate Params III, which corresponds to the ordinary BFV scheme. This is because the asymptotic complexity follows $\widetilde{O}(L^2N)$, as analyzed in Section 2.5, and the generalized BFV scheme offers much smaller values for N and L.

When it comes to the amortized cost, which is the cost per single triple, Params I achieves about a 1.14 times speedup for $\log p = 128$, and this gap gradually increases to a 3.6 times speedup for $\log p = 1024$ as $\log p$ grows. This is because the complexity of the amortized cost follows $\widetilde{O}(L^2M)$, where $M = N/D$. For the ordinary BFV scheme, doubling the size of $\log p$ results in doubling L, while M is fixed at 1, which quadruples the amortized cost due to the L^2 factor. In contrast, for the generalized BFV scheme, we can double the size of $\log p$ by doubling M while keeping L fixed, which only doubles the amortized cost since the complexity is linear in M. This effect is well demonstrated in Table 5, where the gap between the generalized BFV and ordinary BFV schemes widens as $\log p$ increases. Although we could not report benchmark results for Params III when $\log p = 2048$ and 4096 due to parameter blow-up, we expect Params I to achieve about a 7 times speedup for $\log p = 2048$ and a 14 times speedup for $\log p = 4096$ compared to Params III, following the complexity analysis.

Matrix Triples. For matrix triples, we measure the perfor-

mance of MatMul and the generation of authenticated matrix triples, which is accomplished by additionally multiplying the encryption $\mathtt{ct}_\alpha$ of the MAC key. Thanks to the generalized BFV scheme, the performance of MatMul scales linearly with the field size $\log p$ as demonstrated in Table 6.

To demonstrate the effect of the new matrix multiplication algorithm, we compare the performance of authenticated matrix triple generation with the previous work [CKR+20] for the same matrix dimensions and field size in Table 7. The benchmark results for the previous work are taken from Table 3 in [CKR+20], and all costs are measured in an amortized sense, dividing the total cost by the number of batched matrices. As shown in Table 7, our method achieves an approximately 30 to 34 times improvement in each dimension compared to previous work. We attribute this speedup to the complexity improvements in our algorithm, which reduce the total complexity from $\widetilde{O}(d^3L^2)$ to $\widetilde{O}(d^3L)$. This improvement synergizes particularly well with our generalized BFV scheme, which effectively reduces the value of L for the same field size $\log p$. As a result, our approach achieves a significant performance boost over the previous work by leveraging both the generalized BFV scheme and the new matrix multiplication algorithm.

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.

[AS22] Damiano Abram and Peter Scholl. Low-communication multiparty triple generation for spdz from ring-lpn. In *IACR International Conference on Public-Key Cryptography*, pages 221–251. Springer, 2022.

[Ban95] Wojciech Banaszczyk. Inequalities for convex bodies and polar reciprocal lattices in r n. *Discrete & Computational Geometry*, 13:217–231, 1995.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. In *Annual International Cryptology Conference*, pages 489–518, 2019.

[BCIV20] Carl Bootland, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Efficiently processing complex-valued data in homomorphic encryption. *Journal of Mathematical Cryptology*, 14(1):55–65, 2020.

[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.

[BDF+23] Jakob Burkhardt, Ivan Damgård, Tore Kasper Frederiksen, Satrajit Ghosh, and Claudio Orlandi. Improved distributed rsa key generation using the miller-rabin test. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2501–2515, 2023.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology—CRYPTO'91: Proceedings 11*, pages 420–432. Springer, 1992.

[BP23] Luıs Brandao and Rene Peralta. Nist first call for multi-party threshold schemes. *doi*, 10:6028, 2023.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual cryptology conference*, pages 868–886. Springer, 2012.

[CDK+22] Megan Chen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, Abhi Shelat, and Ran Cohen. Multiparty generation of an rsa modulus. *Journal of Cryptology*, 35(2):12, 2022.

[CKR+20] Hao Chen, Miran Kim, Ilya Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. Maliciously secure matrix multiplication with applications to private deep learning. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 31–59, 2020.

[DEK20] Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020.

[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.

[FLOP18] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed rsa key generation for semi-honest and malicious adversaries. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II 38*, pages 331–361. Springer, 2018.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[HPS19] Shai Halevi, Yuriy Polyakov, and Victor Shoup. An improved rns variant of the bfv homomorphic encryption scheme. In *Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings*, pages 83–105. Springer, 2019.

[HS14] Shai Halevi and Victor Shoup. Algorithms in helib. In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*, pages 554–571. Springer, 2014.

[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.

[HS18] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in HElib. In *Annual International Cryptology Conference*, pages 93–120. Springer, 2018.

[HSS24] Intak Hwang, Jinyeong Seo, and Yongsoo Song. Concretely efficient lattice-based polynomial commitment from standard assumptions. In *Annual International Cryptology Conference*, pages 414–448. Springer, 2024.

[JKLS18] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1209–1222, 2018.

[JLK+22] Jaehee Jang, Younho Lee, Andrey Kim, Byunggook Na, Donggeon Yhee, Byounghan Lee, Jung Hee Cheon, and Sungroh Yoon. Privacy-preserving deep sequential model with matrix homomorphic encryption. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 377–391, 2022.

[KLSS23] Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-mlwe. In *Annual International Cryptology Conference*, pages 549–580. Springer, 2023.

[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.

[MBTPH20] Christian Vincent Mouchet, Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Lattigo: A multiparty homomorphic encryption library in go. In *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, pages 64–70, 2020.

[MG23] Johannes Mono and Tim Güneysu. Implementing and optimizing matrix triples with homomorphic encryption. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, pages 29–40, 2023.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007.

[MR18] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 35–52, 2018.

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In *Annual Cryptology Conference*, pages 80–97. Springer, 2010.

[RRKK23] Pascal Reisert, Marc Rivinius, Toomas Krips, and Ralf Küsters. Overdrive lowgear 2.0: Reduced-bandwidth mpc without sacrifice. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, pages 372–386, 2023.

[RST+22] Dragos Rotaru, Nigel P Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for spdz. *Journal of Cryptology*, 35(1):5, 2022.

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019.

[ZLW23] Xiaopeng Zheng, Hongbo Li, and Dingkang Wang. A new framework for fast homomorphic matrix multiplication. *Cryptology ePrint Archive*, 2023.

Appendix

1. Deferred Protocols

2. Deferred Proofs

2.1. Proof of Lemma 3.

Proof. From the relation $p = b^M + 1$, it follows that the order of b modulo p is $2M$. Therefore, there exists a primitive root g modulo p such that $g^{\frac{p-1}{2M}} = b$. Then, $\xi := g^{\frac{p-1}{2N}}$ is a root of the equation $X^D - b = 0$ modulo p. Moreover, ξ^{2iM+1} is also a root since $\xi^{(2iM+1)D} = \xi^{2iN+D} = \xi^D$ for any $i \in \mathbb{Z}$. Since the order of ξ is N, $\{\xi^{2iM+1}\}_{0 \le i < D}$ are distinct and they are essentially the roots of the given equation. □

2.2. Proof of Lemma 4.

Proof. For any two elements $2iM + 1,\ 2jM + 1 \in G$, $(2iM + 1)(2jM + 1) = 2(2ijM + i + j)M + 1 \in G$ holds, so G is closed under multiplication. Also, G contains the identity $1 \in G$. For an element $2iM + 1 \in G$, it has a multiplicative inverse $x \in \mathbb{Z}_{2N}^{\times}$ such that $(2iM + 1)x = 1$

Π_{Triple}

Initialize: All parties invoke $\mathcal{F}_{\mathsf{KeyGen}}$ to obtain a shared public key pk, and evaluation key $\mathsf{evk} = \{\mathsf{rlk}, \mathsf{rtk}_1, \ldots, \mathsf{rtk}_{D-1}\}$. Then, each party P_ℓ does the followings for $0 \le \ell < n$:

1) Party P_ℓ invokes the sampling phase of Π_{PoPK} with parameter $U = 1$, $V = \lceil(\lambda_{\mathsf{Snd}} + 2)/\log_2(2M)\rceil$, and $\mathsf{Flag} = \mathsf{Diag}$ to obtain a ciphertext $\mathsf{ct}_\alpha^{(\ell)}$, and an additive share $\alpha^{(\ell)} \in \mathbb{Z}_p$.
2) All parties compute $\mathsf{ct}_\alpha = 2 \cdot \sum_{\ell=0}^{n-1} \mathsf{ct}_\alpha^{(\ell)}$, and run the remaining phases of Π_{PoPK}.

Scalar Triples: Parties run this protocol to generate T random authenticated scalar triples in $\mathbb{Z}_p$ in one invocation, where $D \mid T$.

1) Party P_ℓ invokes the sampling phase of Π_{PoPK} with parameter $U = 2T/D$, $V = \lceil(\lambda_{\mathsf{Snd}} + 2)/\log_2(2N)\rceil$, and $\mathsf{Flag} = \perp$ to obtain ciphertexts $\mathsf{ct}_{\mathbf{a}_i}^{(\ell)}$, $\mathsf{ct}_{\mathbf{b}_i}^{(\ell)}$, and additive shares $\mathbf{a}_i^{(\ell)}, \mathbf{b}_i^{(\ell)} \in \mathbb{Z}_p^D$ for $0 \le i < T/D$, and run the remaining phases of Π_{PoPK}.
2) All parties compute $\mathsf{ct}_{\mathbf{a}_i} = 2 \cdot \sum_{\ell=0}^{n-1} \mathsf{ct}_{\mathbf{a}_i}^{(\ell)}$, $\mathsf{ct}_{\mathbf{b}_i} = 2 \cdot \sum_{\ell=0}^{n-1} \mathsf{ct}_{\mathbf{b}_i}^{(\ell)}$ for $0 \le i < T/D$.
3) All parties compute $\mathsf{ct}_{\mathbf{c}_i} \leftarrow \mathtt{Mul}(\mathsf{ct}_{\mathbf{a}_i}, \mathsf{ct}_{\mathbf{b}_i})$, $\mathsf{ct}_{\alpha\mathbf{a}_i} \leftarrow \mathtt{Mul}(\mathsf{ct}_\alpha, \mathsf{ct}_{\mathbf{a}_i})$, $\mathsf{ct}_{\alpha\mathbf{b}_i} \leftarrow \mathtt{Mul}(\mathsf{ct}_\alpha, \mathsf{ct}_{\mathbf{b}_i})$, and $\mathsf{ct}_{\alpha\mathbf{c}_i} \leftarrow \mathtt{Mul}(\mathsf{ct}_\alpha, \mathsf{ct}_{\mathbf{c}_i})$ for $0 \le i < T/D$.
4) Parties run Π_{DDec} with ciphertexts $\mathsf{ct}_{\mathbf{c}_i}, \mathsf{ct}_{\alpha\mathbf{a}_i}, \mathsf{ct}_{\alpha\mathbf{b}_i}, \mathsf{ct}_{\alpha\mathbf{c}_i}$ so that each P_ℓ obtains additive shares $\mathbf{c}_i^{(\ell)}, (\alpha\mathbf{a}_i)^{(\ell)}, (\alpha\mathbf{b}_i)^{(\ell)}, (\alpha\mathbf{c}_i)^{(\ell)}$ for $0 \le i < T/D$.

Matrix Triples: Parties run this protocol to generate T random authenticated matrix triples in $\mathbb{Z}_p^{D\times D}$ in one invocation.

1) Party P_ℓ invokes the sampling phase of Π_{PoPK} with parameters $U = 2DT$, $V = \lceil(\lambda_{\mathsf{Snd}} + 2)/\log_2(2N)\rceil$, and $\mathsf{Flag} = \perp$ to obtain ciphertexts $\mathsf{CT}_{\iota(\mathbf{A}_i)}^{(\ell)}$, $\mathsf{CT}_{\widetilde{\iota}(\mathbf{B}_i)}^{(\ell)}$, and additive shares $\iota(\mathbf{A}_i)^{(\ell)}$, $\widetilde{\iota}(\mathbf{B}_i)^{(\ell)}$ for $0 \le i < T$, and runs the remaining phases of Π_{PoPK}.
2) All parties compute $\mathsf{CT}_{\iota(\mathbf{A}_i)} = 2 \cdot \sum_{\ell=0}^{n-1} \mathsf{CT}_{\iota(\mathbf{A}_i)}^{(\ell)}$, $\mathsf{CT}_{\widetilde{\iota}(\mathbf{B}_i)} = 2 \cdot \sum_{\ell=0}^{n-1} \mathsf{CT}_{\widetilde{\iota}(\mathbf{B}_i)}^{(\ell)}$ for $0 \le i < T$.
3) All parties compute $\mathsf{CT}_{\iota(\mathbf{C}_i)} \leftarrow \mathtt{MatMul}(\mathsf{CT}_{\iota(\mathbf{A}_i)}, \mathsf{CT}_{\widetilde{\iota}(\mathbf{B}_i)})$ for $0 \le i < T$.
4) All parties compute the followings for $0 \le i < T$ and $0 \le j < D$,

$$\mathsf{ct}_{\alpha\cdot\iota(\mathbf{A}_i)_j} \leftarrow \mathtt{Mul}(\mathsf{ct}_\alpha, \mathsf{ct}_{\iota(\mathbf{A}_i)_j})$$
$$\mathsf{ct}_{\alpha\cdot\widetilde{\iota}(\mathbf{B}_i)_j} \leftarrow \mathtt{Mul}(\mathsf{ct}_\alpha, \mathsf{ct}_{\widetilde{\iota}(\mathbf{B}_i)_j})$$
$$\mathsf{ct}_{\alpha\cdot\iota(\mathbf{C}_i)_j} \leftarrow \mathtt{Mul}(\mathsf{ct}_\alpha, \mathsf{ct}_{\iota(\mathbf{C}_i)_j})$$

and set $\mathsf{CT}_{\alpha\cdot\iota(\mathbf{A}_i)} = (\mathsf{ct}_{\alpha\cdot\iota(\mathbf{A}_i)_j})_{0\le j<D}$, $\mathsf{CT}_{\alpha\cdot\widetilde{\iota}(\mathbf{B}_i)} = (\mathsf{ct}_{\alpha\cdot\widetilde{\iota}(\mathbf{B}_i)_j})_{0\le j<D}$, and $\mathsf{CT}_{\alpha\cdot\iota(\mathbf{C}_i)} = (\mathsf{ct}_{\alpha\cdot\iota(\mathbf{C}_i)_j})_{0\le j<D}$.
5) Parties run Π_{DDec} with ciphertexts $\mathsf{CT}_{\iota(\mathbf{C}_i)}$, $\mathsf{CT}_{\alpha\cdot\iota(\mathbf{A}_i)}$, $\mathsf{CT}_{\alpha\cdot\widetilde{\iota}(\mathbf{B}_i)}$, $\mathsf{CT}_{\alpha\cdot\iota(\mathbf{C}_i)}$ so that each P_ℓ obtains additive shares $\iota(\mathbf{C}_i)^{(\ell)}$, $(\alpha \cdot \iota(\mathbf{A}_i))^{(\ell)}$, $(\alpha \cdot \widetilde{\iota}(\mathbf{B}_i))^{(\ell)}$, $(\alpha \cdot \iota(\mathbf{C}_i))^{(\ell)}$ for $0 \le i < T$.

Figure 4. Protocol for authenticated triples generation

$\mathcal{F}_{\mathsf{KeyGen}}$

1) Upon receiving start from all honest parties, sample $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{KeyGen}(\mathsf{pp})$, $\mathsf{rlk} \leftarrow \mathtt{RelinKeyGen}(\mathsf{sk})$, $\mathsf{rtk}_r \leftarrow \mathtt{RotKeyGen}(\mathsf{sk}, r)$ for $1 \le r < D$, send $\mathsf{pk}, \mathsf{evk} = \{\mathsf{rlk}, \mathsf{rtk}_1, \ldots, \mathsf{rtk}_{D-1}\}$ to the adversary, and store $\mathsf{sk} = (1, s)$.
2) Receive the shares of secret key $s^{(\ell)}$ from the adversary.
3) Construct a complete set of shares $(s^{(0)}, \ldots, s^{(\ell-1)})$ consistent with the adversary's choice and s.
4) Send pk, evk and $s^{(\ell)}$ to each honest party P_ℓ.

Figure 5. The ideal functionality for key generation

Π_{DDec}

This protocol is run between n parties, where each P_ℓ holds a shared ciphertext $\mathsf{ct} = (c_0, c_1)$ and a secret key share $s^{(\ell)}$ for $0 \le \ell < n$. At the end of the protocol, each party obtains an additive share of the message $\mathbf{m}$ of ct.

1) Party P_ℓ samples $e^{(\ell)} \leftarrow \mathcal{U}(R_{B_{\mathsf{Dec}} \cdot 2^{\lambda_{\mathsf{DDec}}}})$, $\mathbf{x}^{(\ell)} \leftarrow \mathcal{U}(\mathbb{Z}_p^D)$, and sets $\mu_{\mathbf{x}}^{(\ell)} \leftarrow \mathtt{Pack}(\mathbf{x}^{(\ell)})$.
2) Party P_ℓ computes $d^{(\ell)} = c_1 s^{(\ell)} + e^{(\ell)} + \Delta \mu_{\mathbf{x}}^{(\ell)} \pmod q$, and distributes it.
3) Party P_ℓ computes $\mu = \left\lfloor \frac{X^D - b}{q} (c_0 + \sum_{\ell=0}^{n-1} d^{(\ell)}) \right\rceil \pmod{X^D - b}$. If $\ell = 0$, P_0 sets $\mathbf{m}^{(0)} = \mathtt{Unpack}(\mu) - \mathbf{x}^{(0)} \pmod p$. Otherwise, P_ℓ sets $\mathbf{m}^{(\ell)} = -\mathbf{x}^{(\ell)} \pmod p$.

Figure 6. Protocol for distributed decryption

$\pmod{2N}$, then $(2iM+1)x = x = 1 \pmod{2M}$ as $2M$ divides $2N$, so $x \in G$. Then each element in G has an inverse in G. Therefore, G is a subgroup of $\mathbb{Z}_{2N}^\times = \langle 5, -1 \rangle$. Since $-1 \notin G$ as $M \ge 2$, G is a (cyclic) subgroup of $\langle 5 \rangle$. As $|G| = D$, it follows that G is generated by $5^{M/2}$. □

2.3. Proof of Theorem 1.

Proof. By Lemma 3 and Lemma 4, $\xi \in \mathbb{Z}_p$ such that $\xi, \xi^{5^{M/2}}, \dots, \xi^{5^{(D-1)M/2}}$ are distinct roots of $X^D - b$ modulo p. Then, $(X^D - b) = (X - \xi) \dots (X - \xi^{5^{(D-1)M/2}}) \pmod p$ and each linear factors are mutually coprime in $\mathbb{Z}_p[X]$. By the Chinese remainder theorem, $\mathbb{Z}_p[X]/(X^D - b) \simeq \mathbb{Z}_p[X]/(X - \xi) \times \cdots \times Z_p[X]/(X - \xi^{5^{(D-1)M/2}})$. Note that $\mathbb{Z}_p[X]/(X - \xi^{5^{iM/2}})$ is isomorphic to Z_p by the isomorphism $f \mapsto f(\xi^{5^{iM/2}})$, it follows that $R_{X^D - b} = \mathbb{Z}_p[X]/(X^D - b) \simeq \mathbb{Z}_p^d$ with an isomorphism $\tau : f \mapsto (f(\xi), f(\xi^{5^{M/2}}), \cdots, f(\xi^{5^{(D-1)M/2}}))$. □

2.4. Noise bound of Generalized BFV Scheme. We analyze the noise growth of each algorithm under the secret key distribution $\chi = \mathcal{U}(R_3)$ and the error distribution $\psi = \mathcal{D}_\sigma$. The noise growth during the encryption algorithm is as follows.

Lemma 6 (Encryption Noise)**.** *Let $\mu \in R$, $\mathbf{r} \in R^3$ and $\mathsf{ct} \leftarrow \mathtt{Enc}(\mathsf{pk}, \mu, \mathbf{r})$, then the following holds with very high probability*

$$\|\mathsf{Err}(\mathsf{ct}, \mu)\|_\infty^{\mathsf{can}} < \frac{b+1}{q} \left(\|\mu\|_\infty \sqrt{3N^3} + \|\mathbf{r}\|_\infty \left(\left(\sqrt{\frac{18}{\pi}} \sigma + 2 \right) \sqrt{N^3} + N \right) \right)$$

Proof. Let $\mathbf{r} = (r_0, r_1, r_2)$ and $\mathsf{ct} = (c_0, c_1)$, then the following holds for some $I \in R$ and $e_{rd} = q/(X^D - b) - \Delta$.

$$\frac{X^D - b}{q}(c_0 + c_1 s) = \mu + \frac{X^D - b}{q}(e_{rd} \cdot \mu + r_0 \cdot e + r_1 + r_2 \cdot s) + (X^D - b) I$$

Following the analysis in [BCIV20], we get $\|e\|_\infty^{\mathsf{can}} < \sqrt{18/\pi}\sigma\sqrt{N}$, $\|s\|_\infty^{\mathsf{can}} < 2\sqrt{6N}$, and $\|e_{rd}\|_\infty^{\mathsf{can}} < \sqrt{3N}$ with very high probability. Also, it holds that $\|\mathbf{r}\|_\infty^{\mathsf{can}} \le N\|\mathbf{r}\|_\infty$ and $\|\mu\|_\infty^{\mathsf{can}} \le N\|\mu\|_\infty$. Thus, the given inequality holds with very high probability. □

The noise growth of homomorphic operations are as follows.

Lemma 7 (Addition Noise [BCIV20])**.** *Let ct and ct' be ciphertexts with plaintexts μ and μ', respectively. For $\mathsf{ct}'' \leftarrow \mathtt{Add}(\mathsf{ct}, \mathsf{ct}')$, it holds that*

$$\|\mathsf{Err}(\mathsf{ct}'', \mu + \mu')\|_\infty^{\mathsf{can}} \le \|\mathsf{Err}(\mathsf{ct}, \mu)\|_\infty^{\mathsf{can}} + \|\mathsf{Err}(\mathsf{ct}', \mu')\|_\infty^{\mathsf{can}}$$

Lemma 8 (Multiplication Noise [BCIV20])**.** *Let ct and ct' be ciphertexts with plaintexts μ and μ', respectively. Let rlk be a relinearization key. For $\mathsf{ct}'' \leftarrow \mathtt{Mul}(\mathsf{rlk}, \mathsf{ct}, \mathsf{ct}')$, the following holds with very high probability.*

$$\begin{aligned}\|\mathsf{Err}(\mathsf{ct}'', \mu\mu')\|_\infty^{\mathsf{can}} \le & (b+1)\sqrt{2N^2 + 3N} \left(\|\mathsf{Err}(\mathsf{ct}, \mu)\|_\infty^{\mathsf{can}} + \|\mathsf{Err}(\mathsf{ct}', \mu')\|_\infty^{\mathsf{can}} \right) \\ & + \frac{b+1}{q} \sqrt{4N^3/3 + 2N^2 + 3N} + \frac{b+1}{q} \sigma B N \sqrt{\frac{3L}{2\pi}} \\ & + 3\|\mathsf{Err}(\mathsf{ct}, \mu)\|_\infty^{\mathsf{can}} \cdot \|\mathsf{Err}(\mathsf{ct}', \mu')\|_\infty^{\mathsf{can}}\end{aligned}$$

Lemma 9 (Rotation Noise)**.** *Let ct be a ciphertext with a plaintext μ. Let rtk_r be a rotation key with a rotation index r. For $\mathsf{ct}' \leftarrow \mathtt{Rot}(\mathsf{rtk}_r, \mathsf{ct})$, the following holds with very high probability*

$$\|\mathsf{Err}(\mathsf{ct}', \varphi^r(\mu)\|_\infty^{\mathsf{can}} \le \|\mathsf{Err}(\mathsf{ct}, \mu)\|_\infty^{\mathsf{can}} + \frac{b+1}{q} \sigma B N \sqrt{\frac{3L}{2\pi}}$$

Proof. Let $\mathsf{ct} = (c_0, c_1)$, $\mathsf{ct}' = (c_0', c_1')$, and $\mathsf{rtk}_r = (-s \cdot \mathbf{a} + \varphi^r(s) \cdot \mathbf{g} + \mathbf{e}, \mathbf{a})$. Then, it holds that $c_0' + c_1' s = \varphi^r(c_0) + \varphi^r(c_1) \cdot \varphi^r(s) + \langle h(c_1'), \mathbf{e} \rangle \pmod q$. Following the analysis of noise growth during key switching as detailed in [BCIV20], the given inequality holds. □

2.5. Noise bound of `MatMul`.

Lemma 10 (Noise Growth of Algorithm 1)**.** *For the input encrypted matrix $\mathsf{CT}_{\iota(\mathbf{A})}$ and $\mathsf{CT}_{\widetilde{\iota}(\mathbf{B})}$, and the output encrypted matrix $\mathsf{CT}_{\iota(\mathbf{C})}$ of the algorithm 1, let $e_{\mathbf{A}}$ and $e_{\mathbf{B}}$ be reals such that*

$$\|\mathsf{Err}(\mathsf{ct}_{\iota(\mathbf{A})_i}, \iota(\mathbf{A})_i)\|_\infty^{\mathsf{can}} \le e_{\mathbf{A}} \quad \textit{and} \quad \|\mathsf{Err}(\mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_i}, \widetilde{\iota}(\mathbf{B})_i)\|_\infty^{\mathsf{can}} \le e_{\mathbf{B}}$$

for all $0 \le i < d$. Then, it follows that

$$\begin{aligned}\|\mathsf{ct}_{\iota(\mathbf{C})_i}\|_\infty^{\mathsf{can}} \le & \, d(b+1)\sqrt{2N^2 + 3N}(e_{\mathbf{A}} + e_{\mathbf{B}}) + 3de_{\mathbf{A}}e_{\mathbf{B}} \\ & + \frac{b+1}{q} \sqrt{\frac{4N^3}{3} + 4N^2 + 3N} + \frac{b+1}{q} \sigma B N \sqrt{\frac{3(N+2)L}{2\pi}}\end{aligned}$$

for all $0 \le i < d$.

Proof. Note that the computation algorithm remains the same for every $\mathsf{ct}_{\iota(\mathbf{C})_i}$, we estimate the noise for a single ciphertext $\mathsf{ct}_{\iota(\mathbf{C})_i}$ without loss of generality. First, the noise growth from a single ciphertext tensoring of $\mathsf{ct}_{\iota(\mathbf{A})_j}$ and $\mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_{i-j}}$ is

$$\begin{aligned}& (b+1)\sqrt{2N^2 + 3N}(\|\mathsf{ct}_{\iota(\mathbf{A})_j}\|_\infty^{\mathsf{can}} + \|\mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_{i-j}}\|_\infty^{\mathsf{can}}) \\ & + 3\|\mathsf{ct}_{\iota(\mathbf{A})_j}\|_\infty^{\mathsf{can}} \cdot \|\mathsf{ct}_{\widetilde{\iota}(\mathbf{B})_{i-j}}\|_\infty^{\mathsf{can}}\end{aligned}$$

Therefore, after the for loop (lines 3-6), the noise is bounded by

$$d\left((b+1)\sqrt{2N^2 + 3N}(e_{\mathbf{A}} + e_{\mathbf{B}}) + 3e_{\mathbf{A}}e_{\mathbf{B}} \right).$$

Now, note that the tensoring ciphertext after the for loop is encrypted under the key $(1, s, \varphi^i(s), s\varphi^i(s))$ as specified,

the modulus changing performed in line 7 introduces the rounding noise bounded as follows.

$$\frac{b+1}{q}\sqrt{4N^3/3+4N^2+3N}$$

Finally, we analyze the noise from the key-switching. Let e_1, e_2 and e_3 denote that error from key-switching operations in lines 8-10, respectively, i.e., $u_0 + u_1 \cdot s = d_2' \cdot \varphi^i(s) + e_1 \pmod q$, $v_0 + v_1 \cdot s = d_3' \cdot \varphi^i(s) + e_2 \pmod q$ and $w_0 + w_1 \cdot s = v_1 \cdot s^2 + e_3 \pmod q$. Then, the error from key-switching is $e_1 + e_2 \cdot s + e_3$ since the following holds.

$$\begin{aligned} &d_0' + u_0 + w_0 + s \cdot (d_1' + u_1 + v_0 + w_1) \\ &= d_0' + d_1' \cdot s + d_2' \cdot \varphi^i(s) + d_3'' \cdot s\varphi^i(s) + e_1 + e_2 \cdot s + e_3 \end{aligned}$$

Since the variance of $e_1 + e_2 \cdot s + e_3$ is equal to $(N+2) \cdot L(\sigma BN)^2/24\pi$, the canonical norm of the key-switching is bounded by $(b+1)/q\sigma BN\sqrt{3(N+2)L/2\pi}$. Therefore, the final noise bound can be computed as follows.

$$\begin{aligned} &d(b+1)\sqrt{2N^2+3N}(e_{\mathbf{A}} + e_{\mathbf{B}}) + 3de_{\mathbf{A}}e_{\mathbf{B}} \\ &+ \frac{b+1}{q}\sqrt{4N^3/3+4N^2+3N} + \frac{b+1}{q}\sigma BN\sqrt{\frac{3(N+2)L}{2\pi}} \end{aligned}$$

We refer to Appendix A of [BCIV20] for a detailed analysis of the rounding noise. □

2.6. Security Proof of Π_{PoPK}.

Theorem 2. *The n-party PoPK-protocol Π_{PoPK} defined in Fig. 3 satisfies the following three properties:*

- ***Correctness:*** *If all parties P_ℓ, with inputs sampled from the sampling phase in Π_{PoPK}, follow the protocol honestly, then the protocol ends with acceptance with high probability, which is at least $1 - 2^{-128}$.*
- ***Knowledge Soundness:*** *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ be a tuple of PPT algorithms and let $\varepsilon \in [0, 1)$. Consider the following game:*
  1) *$\mathcal{A}_1$ takes no input and outputs $C \subset [n]$, $\{(\mathsf{ct}_i^{(\ell)})_{0\le i<U}\}_{\ell\in C}$, and $\mathsf{stat}_{\mathcal{A}_1}$.*
  2) *Sample $(\mathsf{ct}_i^{(\ell)}, \mu_{\mathbf{x}_i}^{(\ell)}, \mathbf{r}_i^{(\ell)})_{0\le i<U}$ from the sampling phase for each $\ell \notin C$.*
  3) *Compute $((\mathsf{comm}_j^{(\ell)})_{0\le j<V}, \mathsf{stat}^{(\ell)})$ from the commitment phase for each $\ell \notin C$.*
  4) *$\mathcal{A}_2$ on input $\mathsf{stat}_{\mathcal{A}_1}$,$(\mathsf{ct}_i^{(\ell)})_{0\le i<U}$, $(\mathsf{comm}_j^{(\ell)})_{0\le j<V}$ for $i \notin C$, outputs $\mathsf{stat}_{\mathcal{A}_2}$ and $(\mathsf{comm}_j^{(\ell)})_{0\le j<V}$ for $\ell \in C$.*
  5) *Choose a uniformly random $(w_{i,j})_{0\le i<U, 0\le j<V}$ and compute $(\mathsf{resp}_j^{(\ell)})_{0\le j<V}$ from the response phase for $\ell \notin C$.*
  6) *$\mathcal{A}_3$ on input $\mathsf{stat}_{\mathcal{A}_2}$, $(w_{i,j})$, $(\mathsf{resp}_j^{(\ell)})$ for $0 \le i < U$, $0 \le j < V$, and $\ell \notin C$, outputs $(\mathsf{resp}_j^{(\ell)})$ for $0 \le j < V$ and $\ell \in C$.*
  7) *$\mathcal{A}$ wins the game if the verification phase ends with acceptance.*

  Suppose $\mathcal{A}$ wins the game with probability $\delta > \varepsilon$. Then, there exists a PPT algorithm $\mathcal{E}$ which for any fixed output of $\mathcal{A}_1$, honestly generated inputs given by $\{(\mathsf{ct}_i^{(\ell)}, \mu_{\mathbf{x}_i}^{(\ell)}, \mathbf{r}_i^{(\ell)})_{0\le i<U}, (\mathsf{comm}_j^{(\ell)})_{0\le j<V}, \mathsf{stat}^{(\ell)}\}_{\ell\notin C}$, and black-box access to $\mathcal{A}_2, \mathcal{A}_3$, outputs $\{(\mu_{\mathbf{x}_i}^{(\ell)}, \mathbf{r}_i^{(\ell)})_{0\le i<U}\}_{\ell\in C}$ such that

  $$\mathsf{ct}_i^{(\ell)} = \mathtt{Enc}(\mu_{\mathbf{x}_i}^{(\ell)}, \mathbf{r}_i^{(\ell)}),\ \left\|2\mu_{\mathbf{x}_i}^{(\ell)}\right\|_\infty < 2NB_{\mathbf{z}},\ \left\|2\mathbf{r}_i^{(\ell)}\right\|_\infty < 2NB_{\mathbf{t}}$$

  holds for $0 \le i < U$ and $\ell \in C$ in at most $f(\lambda_{\mathsf{Snd}})/(\delta - \varepsilon)$ steps, where $f(\cdot)$ is a positive polynomial and $\varepsilon = 2^{-\lambda_{\mathsf{Snd}}}$.

$\mathcal{S}_{\mathsf{PoPK}}$

The inputs are a challenge $(w_{i,j})_{0\le i<U, 0\le j<V}$ and a set $C \subset [n]$.

1) Sample $\mu_{\mathbf{x}_i}^{(\ell)} \leftarrow \mathtt{RandPack}(\mathbf{0}, \mathfrak{s}_1)$, $\mathbf{r}_i^{(\ell)} \leftarrow \mathcal{D}_{\mathbb{Z}^N, \sigma_1}$ for $0 \le i < U$, $\ell \notin C$.
2) Sample $\mathsf{ct}_i^{(\ell)} \leftarrow \mathcal{U}(R_q^2)$ for $0 \le i < U$, $\ell \notin C$.
3) Sample $\mathbf{y}_j^{(\ell)} \leftarrow \mathcal{U}(\mathbb{Z}_p^D)$, $\mu_{\mathbf{y}_j}^{(\ell)} \leftarrow \mathtt{RandPack}(\mathbf{y}_j^{(\ell)}, \sqrt{U+1} \cdot \mathfrak{s}_2)$, $\mathbf{s}_j^{(\ell)} \leftarrow \mathcal{D}_{\mathbb{Z}^N, \sqrt{U+1}\cdot\sigma_2}$ for $0 \le j < V$, $\ell \notin C$.
4) Set $\mu_{\mathbf{z}_j}^{(\ell)} = \mu_{\mathbf{y}_j}^{(\ell)} + \sum_{i=0}^{U-1} w_{i,j} \cdot \mu_{\mathbf{x}_i}^{(\ell)}$, $\mathbf{t}_j^{(\ell)} = \mathbf{s}_j^{(\ell)} + \sum_{i=0}^{U-1} w_{i,j} \cdot \mathbf{r}_i^{(\ell)}$ for $0 \le j < V$, $\ell \notin C$.
5) Set $\mathsf{comm}_j^{(\ell)} = \mathtt{Enc}(\mu_{\mathbf{z}_j}^{(\ell)}, \mathbf{t}_j^{(\ell)}) - \sum_{i=0}^{U-1} w_{i,j} \cdot \mathsf{ct}_i^{(\ell)} \pmod q$ and $\mathsf{resp}_j^{(\ell)} = (\mu_{\mathbf{z}_j}^{(\ell)}, \mathbf{s}_j^{(\ell)})$ for $0 \le j < V$, $\ell \notin C$.
6) Output $(\mathsf{ct}_i^{(\ell)})_{0\le i<U}$, $(\mathsf{comm}_j^{(\ell)})_{0\le j<V}$, and $(\mathsf{resp}_j^{(\ell)})_{0\le j<V}$ for $\ell \notin C$.

Figure 7. Simulator for Π_{PoPK}

- ***Simulatability:*** *Let $\varepsilon = 2^{-\lambda_{\mathsf{Sim}}}$ be negligible, and let σ_1, σ_2, $\mathfrak{s}_1$, and $\mathfrak{s}_2$ satisfy the following conditions.*
  - *$\frac{\mathfrak{s}_2}{\sqrt{2}}, \mathfrak{s} \ge \frac{1}{b-1} \cdot \eta_\varepsilon(P\mathbb{Z}^N)$, where $\mathfrak{s} = (\mathfrak{s}_1^{-2} + V\mathfrak{s}_2^{-2})^{-1/2}$*
  - *$\frac{\sigma_2}{\sqrt{2}}, \frac{\sigma}{\sqrt{2}} \ge \eta_\varepsilon(\mathbb{Z}^N)$, where $\sigma = \frac{1}{\sqrt{2}} \cdot \left(\sigma_1^{-2} + V\sigma_2^{-2}\right)^{-1/2}$*

  Then, there exists a PPT algorithm $\mathcal{S}$, which takes as input a challenge $(w_{i,j})_{0\le i<U, 0\le j<V}$ and a set $C \subset [n]$, and outputs $(\mathsf{ct}_i^{(\ell)})_{0\le i<U}$, $(\mathsf{comm}_j^{(\ell)}, \mathsf{resp}_j^{(\ell)})_{0\le j<V}$ for $\ell \notin C$, which is computationally indistinguishable from a valid execution of the protocol assuming the hardness of $\mathsf{RLWE}_{R,q,\sigma,\sigma}$.

Proof. We prove the correctness, knowledge soundness, and simulatability of Π_{PoPK} as follows.

- **Correctness:** By Lemma 2, the followings hold

  $$\begin{aligned} &\left\|\mu_{\mathbf{x}_i}^{(\ell)}\right\|_\infty < 6(b+1)\mathfrak{s}_1,\ \left\|\mu_{\mathbf{y}_j}^{(\ell)}\right\|_\infty < 6(b+1)\sqrt{U+1} \cdot \mathfrak{s}_2 \\ &\left\|\mathbf{r}_i^{(\ell)}\right\|_\infty < 6\sigma_1,\ \left\|\mathbf{s}_j^{(\ell)}\right\|_\infty < 6\sqrt{U+1} \cdot \sigma_2 \end{aligned}$$

  with high probability. Thus, it holds that $\left\|\mu_{\mathbf{z}_j}\right\|_\infty < B_{\mathbf{z}}$ and $\left\|\mathbf{t}_j\right\|_\infty < B_{\mathbf{t}}$ with high probability.
- **Knowledge Soundness:** We refer to the proof of Theorem 4 in [CKR+20].
- **Simulatability:** We can show that $\mathcal{S}_{\mathsf{PoPK}}$ in Fig. 7 is a simulator for Π_{PoPK} in a similar manner to Theorem 4 in [HSS24]. Below, we provide a brief sketch of the proof. We prove the simulatability of the protocol using the hybrid arguments $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2$ in Fig. 8. First, we show that the real transcript is statistically indistinguishable from $\mathcal{H}_0$ using the convolution lemma [Pei10] for discrete Gaussian distributions. Second, we show that $\mathcal{H}_0$ is computationally indistinguishable from $\mathcal{H}_1$ due to the hardness of the Hint-RLWE problem. Next, $\mathcal{H}_1$ and $\mathcal{H}_2$ are statistically indistinguishable thanks to the randomized packing method. Finally, $\mathcal{H}_2$ and $\mathcal{S}_{\mathsf{PoPK}}$ are statistically indistinguishable due to the convolution lemma again. Therefore, the real transcript is computationally indistinguishable from the simulated one. For a more detailed analysis of each hybrid argument, we refer to [HSS24].

□

2.7. Noise bound of $\mathsf{CT}_{\alpha\cdot\iota(\mathbf{C}_i)}$. We begin by analyzing the encryption noise. As analyzed in Lemma 6, the initial encryption noise is dominated by the term $\frac{b+1}{q}\cdot N\sqrt{3N}\cdot\|\mu\|_\infty$. Thus, we bound the noise for $\mathsf{CT}_{\iota(\mathbf{A}_i)} = (\mathsf{ct}_{\iota(\mathbf{A}_i)_j})_{0\le j<D}$ and $\mathsf{CT}_{\widetilde{\iota}(\mathbf{B}_i)} = (\mathsf{ct}_{\widetilde{\iota}(\mathbf{B}_i)_j})_{0\le j<D}$ as follows.

$$\|\mathsf{Err}(\mathsf{ct}_{\iota(\mathbf{A}_i)_j}, \iota(\mathbf{A}_i)_j)\|_\infty^{\mathsf{can}},\ \|\mathsf{Err}(\mathsf{ct}_{\widetilde{\iota}(\mathbf{B}_i)_j}, \widetilde{\iota}(\mathbf{B}_i)_j)\|_\infty^{\mathsf{can}} \lesssim \frac{b+1}{q}\cdot 2N^2\sqrt{3N}\cdot B_{\mathbf{z}}$$

This holds because Π_{PoPK} provides an upper bound for plaintexts as $2NB_{\mathbf{z}}$. Next, we consider the noise growth from `MatMul`. As analyzed in Lemma 10, the dominant term is $d(b+1)\cdot\sqrt{2N^2+3N}(\mathbf{e}_{\mathbf{A}}+\mathbf{e}_{\mathbf{B}})$, so we bound noise for $\mathsf{CT}_{\iota(\mathbf{C}_i)} = (\mathsf{ct}_{\iota(\mathbf{C}_i)_j})_{0\le j<D}$ as follows since $\mathsf{CT}_{\iota(\mathbf{C}_i)} \leftarrow \mathtt{MatMul}(\mathsf{CT}_{\iota(\mathbf{A}_i)}, \mathsf{CT}_{\widetilde{\iota}(\mathbf{B}_i)})$.

$$\|\mathsf{Err}(\mathsf{ct}_{\iota(\mathbf{C}_i)_j}, \iota(\mathbf{C}_i)_j)\|_\infty^{\mathsf{can}} \lesssim \frac{d(b+1)^2}{q}\cdot 4N^3\sqrt{6N}\cdot B_{\mathbf{z}}$$

Lastly, we consider the noise growth from `Mul` when multiplying ct_α to $\mathsf{CT}_{\iota(\mathbf{C}_i)}$. As analyzed in Lemma 8, the dominant term is $(b+1)\sqrt{2N^2+3N}(\ \|\mathsf{Err}(\mathsf{ct},\mu)\|_\infty^{\mathsf{can}} + \|\mathsf{Err}(\mathsf{ct}',\mu')\|_\infty^{\mathsf{can}})$, so the noise for $\mathsf{CT}_{\alpha\cdot\iota(\mathbf{C}_i)} = (\mathsf{ct}_{\alpha\cdot\iota(\mathbf{C}_i)_j})_{0\le j<D}$ is bounded as follows, which becomes the upper bound B_{Dec} for ciphertext noise in Π_{Triple}.

$$\|\mathsf{Err}(\mathsf{ct}_{\alpha\cdot\iota(\mathbf{C}_i)_j}, \iota(\mathbf{C}_i)_j)\|_\infty^{\mathsf{can}} \lesssim \frac{d(b+1)^3}{q}\cdot 8N^4\sqrt{3N}\cdot B_{\mathbf{z}} = B_{\mathsf{Dec}}$$

$\mathcal{H}_0$

The inputs are a challenge $(w_{i,j})_{0\le i<U,0\le j<V}$, a set $C\subset[n]$, and messages $(\mathbf{x}_i^{(\ell)})_{0\le i<U}$ for $\ell\notin C$.

1) Sample $\mu_{\mathbf{x}_i}^{(\ell)}\leftarrow\mathtt{RandPack}(\mathbf{x}_i^{(\ell)},\mathfrak{s}_1)$, $\mathbf{r}_i^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_1}$ for $0\le i<U$, $\ell\notin C$.
2) Set $\mathsf{ct}_i^{(\ell)} = \mathtt{Enc}(\mu_{\mathbf{x}_i}^{(\ell)},\mathbf{r}_i^{(\ell)})$ for $0\le i<U$, $\ell\notin C$.
3) Sample $\mathbf{y}_j^{(\ell)}\leftarrow\mathcal{U}(\mathbb{Z}_p^D)$, $\mu_{\mathbf{y}_{j,i}}^{(\ell)}\leftarrow\mathtt{RandPack}(-\mathbf{W}_{i,j}\mathbf{x}_i^{(\ell)},\mathfrak{s}_2)$, $\mu_{\mathbf{y}_{j,U}}^{(\ell)}\leftarrow\mathtt{RandPack}(\mathbf{y}_j^{(\ell)}+\sum_{i=0}^{U-1}\mathbf{W}_{i,j}\mathbf{x}_i^{(\ell)},\mathfrak{s}_2)$, $\mathbf{s}_{j,i}^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_2}$, and $\mathbf{s}_{j,U}^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_2}$ for $0\le i<U$, $0\le j<V$, $\ell\notin C$, where $\mathbf{W}_{i,j}$ is the negacyclic matrix of $w_{i,j}$.
4) Set $\mu_{\mathbf{z}_j}^{(\ell)} = \mu_{\mathbf{y}_{j,U}}^{(\ell)} + \sum_{i=0}^{U-1}(\mu_{\mathbf{y}_{j,i}}^{(\ell)} + w_{i,j}\cdot\mu_{\mathbf{x}_i}^{(\ell)})$, $\mathbf{t}_j^{(\ell)} = \mathbf{s}_{j,U}^{(\ell)} + \sum_{i=0}^{U-1}(\mathbf{s}_{j,i} + w_{i,j}\cdot\mathbf{r}_i^{(\ell)})$ for $0\le j<V$, $\ell\notin C$.
5) Set $\mathsf{comm}_j^{(\ell)} = \mathtt{Enc}(\mu_{\mathbf{z}_j}^{(\ell)},\mathbf{t}_j^{(\ell)}) - \sum_{i=0}^{U-1} w_{i,j}\cdot\mathsf{ct}_i^{(\ell)} \pmod q$ and $\mathsf{resp}_j^{(\ell)} = (\mu_{\mathbf{z}_j}^{(\ell)},\mathbf{s}_j^{(\ell)})$ for $0\le j<V$, $\ell\notin C$.
6) Output $(\mathsf{ct}_i^{(\ell)})_{0\le i<U}$, $(\mathsf{comm}_j^{(\ell)})_{0\le j<V}$, and $(\mathsf{resp}_j^{(\ell)})_{0\le j<V}$ for $\ell\notin C$.

$\mathcal{H}_1$

The inputs are a challenge $(w_{i,j})_{0\le i<U,0\le j<V}$, a set $C\subset[n]$, and messages $(\mathbf{x}_i^{(\ell)})_{0\le i<U}$ for $\ell\notin C$.

1) Sample $\mu_{\mathbf{x}_i}^{(\ell)}\leftarrow\mathtt{RandPack}(\mathbf{x}_i^{(\ell)},\mathfrak{s}_1)$, $\mathbf{r}_i^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_1}$ for $0\le i<U$, $\ell\notin C$.
2) Sample $\mathsf{ct}_i^{(\ell)}\leftarrow\mathcal{U}(R_q^2)$ for $0\le i<U$, $\ell\notin C$.
3) Sample $\mathbf{y}_j^{(\ell)}\leftarrow\mathcal{U}(\mathbb{Z}_p^D)$, $\mu_{\mathbf{y}_{j,i}}^{(\ell)}\leftarrow\mathtt{RandPack}(-\mathbf{W}_{i,j}\mathbf{x}_i^{(\ell)},\mathfrak{s}_2)$, $\mu_{\mathbf{y}_{j,U}}^{(\ell)}\leftarrow\mathtt{RandPack}(\mathbf{y}_j^{(\ell)}+\sum_{i=0}^{U-1}\mathbf{W}_{i,j}\mathbf{x}_i^{(\ell)},\mathfrak{s}_2)$, $\mathbf{s}_{j,i}^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_2}$, and $\mathbf{s}_{j,U}^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_2}$ for $0\le i<U$, $0\le j<V$, $\ell\notin C$, where $\mathbf{W}_{i,j}$ is the negacyclic matrix of $w_{i,j}$.
4) Set $\mu_{\mathbf{z}_j}^{(\ell)} = \mu_{\mathbf{y}_{j,U}}^{(\ell)} + \sum_{i=0}^{U-1}(\mu_{\mathbf{y}_{j,i}}^{(\ell)} + w_{i,j}\cdot\mu_{\mathbf{x}_i}^{(\ell)})$, $\mathbf{t}_j^{(\ell)} = \mathbf{s}_{j,U}^{(\ell)} + \sum_{i=0}^{U-1}(\mathbf{s}_{j,i} + w_{i,j}\cdot\mathbf{r}_i^{(\ell)})$ for $0\le j<V$, $\ell\notin C$.
5) Set $\mathsf{comm}_j^{(\ell)} = \mathtt{Enc}(\mu_{\mathbf{z}_j}^{(\ell)},\mathbf{t}_j^{(\ell)}) - \sum_{i=0}^{U-1} w_{i,j}\cdot\mathsf{ct}_i^{(\ell)} \pmod q$ and $\mathsf{resp}_j^{(\ell)} = (\mu_{\mathbf{z}_j}^{(\ell)},\mathbf{s}_j^{(\ell)})$ for $0\le j<V$, $\ell\notin C$.
6) Output $(\mathsf{ct}_i^{(\ell)})_{0\le i<U}$, $(\mathsf{comm}_j^{(\ell)})_{0\le j<V}$, and $(\mathsf{resp}_j^{(\ell)})_{0\le j<V}$ for $\ell\notin C$.

$\mathcal{H}_2$

The inputs are a challenge $(w_{i,j})_{0\le i<U,0\le j<V}$, and a set $C\subset[n]$ for $\ell\notin C$.

1) Sample $\mu_{\mathbf{x}_i}^{(\ell)}\leftarrow\mathtt{RandPack}(\mathbf{0},\mathfrak{s}_1)$, $\mathbf{r}_i^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_1}$ for $0\le i<U$, $\ell\notin C$.
2) Set $\mathsf{ct}_i^{(\ell)}\leftarrow\mathcal{U}(R_q^2)$ for $0\le i<U$, $\ell\notin C$.
3) Sample $\mathbf{y}_j^{(\ell)}\leftarrow\mathcal{U}(\mathbb{Z}_p^D)$, $\mu_{\mathbf{y}_{j,i}}^{(\ell)}\leftarrow\mathtt{RandPack}(\mathbf{0},\mathfrak{s}_2)$, $\mu_{\mathbf{y}_{j,U}}^{(\ell)}\leftarrow\mathtt{RandPack}(\mathbf{y}_j^{(\ell)},\mathfrak{s}_2)$, $\mathbf{s}_{j,i}^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_2}$, and $\mathbf{s}_{j,U}^{(\ell)}\leftarrow\mathcal{D}_{\mathbb{Z}^N,\sigma_2}$ for $0\le i<U$, $0\le j<V$, $\ell\notin C$.
4) Set $\mu_{\mathbf{z}_j}^{(\ell)} = \mu_{\mathbf{y}_{j,U}}^{(\ell)} + \sum_{i=0}^{U-1}(\mu_{\mathbf{y}_{j,i}}^{(\ell)} + w_{i,j}\cdot\mu_{\mathbf{x}_i}^{(\ell)})$, $\mathbf{t}_j^{(\ell)} = \mathbf{s}_{j,U}^{(\ell)} + \sum_{i=0}^{U-1}(\mathbf{s}_{j,i} + w_{i,j}\cdot\mathbf{r}_i^{(\ell)})$ for $0\le j<V$, $\ell\notin C$.
5) Set $\mathsf{comm}_j^{(\ell)} = \mathtt{Enc}(\mu_{\mathbf{z}_j}^{(\ell)},\mathbf{t}_j^{(\ell)}) - \sum_{i=0}^{U-1} w_{i,j}\cdot\mathsf{ct}_i^{(\ell)} \pmod q$ and $\mathsf{resp}_j^{(\ell)} = (\mu_{\mathbf{z}_j}^{(\ell)},\mathbf{s}_j^{(\ell)})$ for $0\le j<V$, $\ell\notin C$.
6) Output $(\mathsf{ct}_i^{(\ell)})_{0\le i<U}$, $(\mathsf{comm}_j^{(\ell)})_{0\le j<V}$, and $(\mathsf{resp}_j^{(\ell)})_{0\le j<V}$ for $\ell\notin C$.

Figure 8. Hybrid arguments for the simulatability of Π_{PoPK}