IACR Communications in Cryptology
ISSN 3006-5496, Vol. 2, No. 2, 35 pages.

https://doi.org/10.62056/ay7qjp10

Revisiting Module Lattice-based Homomorphic Encryption and Application to Secure-MPC

Anisha Mukherjee and Sujoy Sinha Roy

Graz University of Technology, Graz, Austria

Abstract. Homomorphic encryption (HE) schemes have gained significant popularity in modern privacy-preserving applications across various domains. While research on HE constructions based on learning with errors (LWE) and ring-LWE has received major attention from both cryptographers and software-hardware designers alike, their module-LWE-based counterpart has remained comparatively under-explored in the literature. A recent work provides a module-LWE-based instantiation (MLWE-HE) of the Cheon-Kim-Kim-Song (CKKS) scheme and showcases several of its advantages such as parameter flexibility and improved parallelism. However, a primary limitation of this construction is the quadratic growth in the size of the relinearization keys. Our contribution is *two-pronged*: first, we present a new relinearization key-generation technique that addresses the issue of quadratic key size expansion by reducing it to linear growth. Second, we extend the application of MLWE-HE in a multi-group homomorphic encryption (MGHE) framework, thereby generalizing the favorable properties of the single-keyed HE to a multi-keyed setting as well as investigating additional flexibility attributes of the MGHE framework.

Keywords: Homomorphic encryption · module lattices · multi-party homomorphic encryption · multi-key homomorphic encryption

1 Introduction

Homomorphic encryption (HE) offers a novel solution to the predicament of protecting sensitive data versus processing it in untrusted environments such as a cloud server. By allowing 'homomorphic' evaluation of the client's data, HE eradicates the server's need for a plain view of this data, in other words, the server can operate any arbitrary function on the encrypted data without needing to decrypt it. Fully homomorphic encryption (FHE), first introduced by Craig Gentry [Gen09] in 2009, has since evolved with many lattice-based schemes such as [BV11], [FV12] (BFV), [BGV11] (BGV), [CGGI20] (TFHE) and [CKKS17, CHK+18] (CKKS), primarily based on the ring learning with errors (RLWE) problem [LPR10]. Very recently, the authors of [MAM+24] proposed an instantiation of the CKKS scheme based on the module learning with errors (MLWE) problem [LS15]. Their underlying idea was to be able to fix a base polynomial degree and vary only the rank of the associated module with respect to the ciphertext modulus so as to adapt to different parameters required by different applications while complying with the security requirements of the scheme. The authors termed this design paradigm 'hardware-friendly' as in a hardware implementation, it translates to an optimized architecture design targeting the fixed polynomial degree and instantiating multiple of these units based on the dimension of a matrix or a vector of such fixed-degree polynomials as per desired application parameters. In contrast, RLWE-based hardware accelerators are typically optimized for

E-mail: anisha.mukherjee@tugraz.at (Anisha Mukherjee), sujoy.sinharoy@tugraz.at (Sujoy Sinha Roy)

Received: 2025-01-14 Accepted: 2025-06-02

a narrow range of parameters, restricting their usability to specific applications due to limitations in handling large parameter variations. Their work highlighted several benefits of an MLWE-based HE scheme such as more possibilities to process each component of the module element in parallel and independent of each other until an accumulation is required, and increased flexibility in parameter selection before and during the execution of homomorphic operations. It also discussed module-specific additional flexibility of dynamic ciphertext compression in the form of a partial key-switch-like subroutine which they refer to as 'rank-reduction'. Since the ciphertext modulus decreases with a decrease in the multiplicative depth, rank reduction can be used to reduce the module rank proportionately with respect to the lattice dimension required for maintaining the security at that depth. A reduction in the module rank results in a decrease in the number of components in each module ciphertext element which further reduces computation complexity and key sizes in the subsequent homomorphic evaluations. Moreover, the hardness of the MLWE problem could offer better security assurances compared to RLWE under various algebraic attacks on structural lattices [CDW17]. Hence, *MLWE-based HE provides scalable security and a more granular functionality without the need to change the underlying algebra.* However, as noted in [MAM+24], an increased memory requirement posed a major limitation. Due to the module structure, the number of relinearization keys increase quadratically with respect to the rank of the module. Therefore, recognizing the potentials of an MLWE-based HE scheme in the fast-changing security requirements and emerging technologies, in this work, we propose a tweaked relinearization procedure that reduces the quadratic increase of the relinearization keys to linear with respect to the module rank. Furthermore, being in its nascent stage, application of MLWE-based HE in privacy-preserving systems, that is, at a protocol-level, still remains completely unexplored. We take the first step in this direction by applying the improved MLWE-based HE in privacy-preserving multi-party computations (MPC). Hereafter, we use 'MLWE-HE' to refer to the construction by [MAM+24].

HE primitives have been applied for new-age privacy-preserving technologies in an increasingly interconnected landscape, such as in health-care, financial or automotive sectors. Recently, it has become a driving force behind advanced privacy-preserving applications, even capable of catering to the privacy requirements of multiple stakeholders in the form of secure multiparty computation (MPC), which have been explored in works such as [ZPGS19, FTSH20, CLR17]. Secure MPC enables multiple participants to collaboratively compute a function on their private inputs without disclosing these inputs to each other. Moreover, usual MPC protocols often employ a secret-sharing scheme requiring secure private channels to share secret input data among the parties. This in turn results in multiple rounds of interaction between parties with communication complexities which are directly proportional to the number of involved parties. Homomorphic encryption helps to reduce the number of these interaction rounds among parties [FH96, CDN01] and eliminates the need for a trusted setup that delegates the secret keys to all parties securely. The research of HE-based MPC has developed in two directions: the *threshold* variants (hereafter, we refer to such schemes as multi-party HE schemes or MPHE) and the *multi-key* variants (MKHE). While MKHE schemes face increased computational complexity, they provide flexibility in the number of parties. MPHE schemes on the other hand behave like single-keyed HE and hence do not have additional computational complexity but the number of participating entities is fixed at the setup phase of these schemes. The use of HE in a multi-party and multi-key setting was investigated in a line of works such as [AJLA+12, BS23, PLZ24, MTBH21, CDKS19, LTV12]. However, these two directions are not mutually exclusive and a combined scheme with the functionality of both can benefit from the best of both worlds. In the most recent work, [KLSW24], the authors explore this direction and propose a *hybrid* called *multi-group* homomorphic encryption scheme (MGHE) consisting of multiple *groups* of parties. Thus, a multi-group HE scheme enjoys the flexibility of a multi-key encryption and the simplicity of a multi-party encryption and

hence is beneficial for applications that require interpolation between an MPHE and MKHE setting. For example, consider a smartphone company that employs *encrypted federated learning* to enhance voice recognition and predictive typing features by securely aggregating encrypted model updates from users' devices, ensuring that personal data remains private while still improving the overall user experience. Note that in this digital age, a single user often has several devices (smartphones, smartwatch) from the same company. In such a scenario, each user can train the model using threshold/multi-party encryption, whereas the prediction or aggregation of the encrypted model updates from multiple users can take place in the server under multi-key encryption. A multi-group setting could ensure seamless transition between the two encryption techniques. Furthermore, quite recently, the authors of [Fer23] have showcased how MGHE schemes can be utilized to develop verifiable multi-party schemes using a replication encoding authenticator, in which the encryption and decryption functions are replaced by an authentication algorithm and a verification. This way, the flexibility provided by the multi-group FHE scheme is extended and modified into a verifiable scheme, that is, such a scheme can support multiple parties as well as multiple groups of parties on top of safeguarding clients from malicious cloud servers.

Motivation for designing an MLWE-based MGHE scheme: On one hand, an MLWE-based HE scheme provides parameter-flexibility, thus allowing hardware-reusability. It also accommodates the feature of dynamic ciphertext compression which can be considered an additional layer of flexibility over an RLWE-based HE scheme. On the other hand, a multi-group HE scheme combines the advantages of the multi-party and multi-key approaches. Within a group, it employs a multi-party encryption and thus mimics a single-keyed HE scheme whereas it allows homomorphic evaluation among ciphertexts of different groups encrypted using their own joint public keys thereby following a multi-key approach. Thus, in this work, we set out to explore the following possibility: *can the scheme-level flexibility of MLWE-HE (requirement-1) be combined with the protocol-level flexibility of MGHE (requirement-2) to obtain a multi-group MLWE-based HE that exhibits the advantages of all of the aforementioned features and incorporates multiple layers of flexibility within the protocol?* Table 1 summarizes the potential features of such a combined scheme.

1.1 Our contributions

We set out to improve and extend the single-key MLWE-HE scheme to support homomorphic computations of encrypted data involving multiple owners and discuss interesting observations about *requirement-1* and *requirement-2* from Table 1 along the way. Our contributions are as follows.

- **Reduced number of keys for MLWE-HE:** We address a major limitation of the MLWE-based homomorphic encryption scheme proposed in [MAM+24]. In MLWE-HE, the secret vector is a module element consisting of r polynomials in the underlying ring, that is, $\mathbf{s} = (s_0, \cdots, s_{r-1})$, where r is the rank. The size of relinearization keys grows quadratically with the rank as they are required to contain the product of every two of the secret polynomials, that is, $\sum_{i,j=0}^{r-1} s_i \cdot s_j$. In this work, we reduce this growth from $\mathcal{O}(r^2)$ to $\mathcal{O}(r)$ by proposing a different structure of the relinearization keys.

- **MGHE based on MLWE-HE:** We construct an MLWE-based multi-group HE scheme [KLSW24]. A multi-group HE scheme involves multiple groups that come together to perform homomorphic operations on ciphertexts. Each group internally consists of several parties. Within each group, an MGHE scheme acts as a MPHE-based scheme, whereas among groups, it can be seen as an MKHE-based scheme.

We provide all homomorphic subroutines associated with the MLWE-based MGHE scheme and also show how to keep the relinearization key sizes linear in the number of groups instead of quadratic in a multi-group scenario.

- **Additional flexibility of MLWE-MGHE:** We further show that an MLWE-based MGHE scheme enjoys additional flexibility due to the module and multi-group structure. By taking inspiration from MLWE-HE subroutines and the method of task delegation in a multi-group setting, we show how a multi-party computation protocol based on MGHE-MLWE can support a dynamic access-structure among the groups participating in the protocol.

Table 1: Comparison of features. The abbreviations MP, MK, MG and SK refer to the terms multi-party, multi-key, multi-group and single-key; the suffixes RL or ML mean these schemes are RLWE or MLWE-based respectively. The symbol $\rightarrow$ shows availability of a feature at scheme or protocol levels. * means feature was not explored in [KLSW24].

Scheme/Features	MP-RL	MK-RL	MG-RL	MG-ML	SK-RL	SK-ML
Flexibility $\rightarrow$ parties	$\times$	✓	$\times^*$	✓	$\times$	$\times$
Flexibility $\rightarrow$ groups	$\times$	$\times$	✓	✓	$\times$	$\times$
Flexibility $\rightarrow$ parameters	$\times$	$\times$	$\times$	✓	$\times$	✓
Security assumptions	RLWE	RLWE	RLWE	MLWE	RLWE	MLWE

Organization: In Sec. 2, we provide background related to MLWE-HE and the different ways HE schemes have been applied in privacy-preserving MPC protocols. Then, we describe the improved relinearization technique in Sec. 3. Sec. 4 introduces our proposed MLWE-based multi-group homomorphic encryption scheme, MLWE-MGHE and presents the flexibility features in Sec. 5. In Sec. 6 we outline the design of an MPC protocol based on our proposed MGHE scheme. We discuss some constraints related to our proposal in Sec. 7. We conclude our work and discuss opportunities for future work in Sec. 8.

2 Background

2.1 Notation

Let $N \in \mathbb{N}$ be a power of two. For a number field $K = \mathbb{Q}[X]/(\phi_{2N}(X))$ we denote $\mathcal{R} = \mathbb{Z}[X]/(\phi_{2N}(X))$ as its ring of integers consisting of polynomials modulo the $2N$-th cyclotomic polynomial, $\phi_{2N}(X) = X^N + 1$. We use r to denote the rank of the $\mathcal{R}$-module $M \subseteq K^r$. Also, let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ be the residue ring of $\mathcal{R}$ modulo an integer q. An element of $\mathcal{R}_q$ is a polynomial of the form, $a(X) = \sum_{i=0}^{N-1} a_i X_i$ with each of its coefficients in $\mathbb{Z}_q$. The Euclidean norm on the coefficient vector (a_i) is denoted simply by $\|a\|$ while the l_∞ norm is $\|a\|_\infty$ such that $\|a\|_\infty = sup_i|a_i|$. The l_∞ norm of a polynomial a under the canonical embedding is denoted by $\|a\|_\infty^{can}$. We provide bounds for the induced error during homomorphic operations with respect to the l_∞ norm following [CKKS17, CHK+18, MAM+24]. Unless stated otherwise, we will use q [1] to denote a ciphertext modulus. Plain lowercase names denote polynomials in a ring

[1]Note that for the example instantiation using CKKS [CKKS17], we use $q = p^l \cdot q_0$ to denote the modulus at any multiplicative level $0 \leq l \leq L$ for a fixed base p, a modulus q_0. [MAM+24] uses a slightly different notation of q_l to denote the ciphertext modulus at a level $0 \leq l \leq L$. In this work, we keep the notation simpler as we work on the scheme and protocol-level and we do not change or modify the parameter setup phase of MLWE-HE.

while bold lowercase names refer to module elements. A matrix is denoted by a bold uppercase letter. We use the '$\texttt{texttt}$' or its math-mode equivalent, '$\texttt{mathtt}$' fonts to formally denote keys or ciphertexts; we will use the notation of a polynomial or module element when the context requires explicit reference to the membership information of the key or ciphertext component. The notation '$\cdot$' is used as an umbrella term for the following types of multiplications: polynomial-polynomial, matrix-vector of polynomials, integer-polynomial or integer-integer multiplications based on the context. The notations $\langle \cdot, \cdot \rangle$ and $\otimes$ are used to explicitly denote inner and tensor products between vectors.

2.2 Overview of an MLWE-based HE construction

An MLWE-based HE scheme involves homomorphic operations over a module $M \subseteq \mathcal{R}_q^r$ over a polynomial ring $\mathcal{R}$ of degree N. Thus, the public and secret components are elements in $\mathcal{R}_q^r$ consisting of vectors of polynomials in the base ring. The encryption procedure and the homomorphic subroutines closely follow the general algorithmic flow of RLWE-based homomorphic encryption schemes. Homomorphic operations between module elements consist of several individual operations among polynomials in the underlying ring. We give a general outline of the CKKS-based example instantiation of the MLWE-HE scheme [MAM+24] in the following paragraph. Concrete subroutines can be found in the appendix A.

Let us consider a security parameter λ that determines the choice of the ring degree N and the module rank r required to securely support a certain size of ciphertext modulus q. A client uses their secret key, $\texttt{sk} = (1, \mathbf{s}) \in \mathcal{R}^r$ and public key, $(b = -\mathbf{A} \cdot \mathbf{s} + \mathbf{e}, \mathbf{A}) \in \mathcal{R}_q^r \times \mathcal{R}_q^{r \times r}$ to encrypt a message m into a ciphertext of the form $\texttt{ct} = (c_0, \mathbf{c_1}) \in \mathcal{R}_q \times \mathcal{R}_q^r$ through an RLWE-HE-like encryption procedure. The example instantiation of MLWE-HE in [MAM+24] follows RLWE-based CKKS [CKKS17, CHK+18] and so upon decryption using $\texttt{sk}$, the client obtains an approximation of the initial message, that is, $(c_0 + \mathbf{c_1} \cdot \mathbf{s}) \pmod q \approx m$. In the cloud server, ciphertexts, say $\texttt{ct}$ and $\texttt{ct}'$ can be added or multiplied based on the application's requirements. Addition produces another ciphertext of the same form $\texttt{ct}_{\texttt{add}} \in \mathcal{R}_q \times \mathcal{R}_q^r$, while a homomorphic ciphertext-ciphertext multiplication results in a slightly different form of ciphertext consisting of four components of the form $\texttt{ct}_{\texttt{mult}} = (d_0, \mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}) \in \mathcal{R}_q \times \mathcal{R}_q^r \times \mathcal{R}_q^r \times \mathcal{R}_q^{r(r-1)/2}$. Such a ciphertext can be decrypted using $(1, \mathbf{s}, \mathbf{s} \otimes \mathbf{s})$ such that module components $\mathbf{d_2}$ and $\mathbf{d_3}$ are decryptable using the non-linear secret terms arising from $\mathbf{s} \otimes \mathbf{s}$. Similar to RLWE-based HE, a relinearization procedure is used to transform $\texttt{ct}_{\texttt{mult}}$ into a two-component ciphertext $\texttt{ct}_{\texttt{relin}} = (d_0', \mathbf{d_1'}) \in \mathcal{R}_q \times \mathcal{R}_q^r$ which has a linear decryption complexity with respect to the secret key $\texttt{sk}$. Such a transformation requires key-switching or relinearization keys encrypting special functions of $\mathbf{s}$. The multiplication and relinearization procedure in MLWE-HE is significantly different from its RLWE-based counterparts because of the difference in structure of module elements compared to simple polynomials in a ring. Hence, it is this step that introduces additional computational and memory complexities in the scheme, which we elaborate in the next paragraph. Apart from the aforementioned algorithms, intermediate sub-routines such as rescaling and modulus-switch are also performed to maintain correctness of the result in the modular domain. Finally, after all such evaluations, the server sends the resultant ciphertext to the client for decryption. In practical HE implementations, the concept of Residue Number System (RNS) is used to express the ciphertext modulus q as product of a fixed number of smaller co-prime moduli q_i (that is, $q = \prod_i q_i$) such that all polynomial arithmetic can be efficiently carried out modulo q_i's instead of a single large modulus q. We provide more details about RNS decomposition in appendix A.

The drawback of $\mathcal{O}(r^2)$ relinearization keys: Unlike in RLWE-based CKKS, the rank r of the module affects the total number of keys in MLWE-HE. Depending on the module rank r, there are $\mathcal{O}(r^2)$ non-linear components of ring degree N to be relinearized after a

ciphertext-ciphertext multiplication. In contrast, in the case of RLWE-HE, only one non-linear component of polynomial degree $N \cdot r$ must be relinearized. In the previous paragraph, we discussed that in MLWE-HE the secret vector is of the form $\mathbf{s} = (s_0, \cdots, s_{r-1})$ and hence, we require relinearization keys containing the product of every two of its components, that is, $s_i \cdot s_j$, considering that the terms of the form $s_i \cdot s_j$ and $s_j \cdot s_i$ for a pair of indices (i, j) are absorbed into the same relinearization key. Thus, the total number of keys would be $\frac{r \cdot (r+1)}{2}$ relinearization keys with respect to each distinct quadratic secret component. The total size of relinearization keys in the RNS version of the scheme as reported in [MAM+24] is equal to $r \cdot (r+1)^2 \cdot L \cdot (L+1) \cdot N \cdot \log(q_i)/2$ bits when compared to $2 \cdot L \cdot (L+1) \cdot N \cdot r \cdot \log(q_i)$ bits in the RLWE setting, where L denotes the number of elements in the basis w.r.t the RNS decomposition of the ciphertext modulus $q = \prod_i q_i$. This results in a quadratic increase in the number of keys proportional to r and therefore also results in an increased memory consumption compared to RLWE-HE. Note that we consider the quadratic space complexity corresponding to the $r(r+1)/2$ non-linearized components instead of cubic-in-r (as evident from the aforementioned bit-size) because one widely-used approach to compress public or relinearization keys is to use seeded-LWE or its variants. This involves sending and storing a pseudo-random seed that can be deterministically expanded to obtain the public material (matrix $\mathbf{A}$ in case of LWE/MLWE and polynomial a in case of RLWE), which trivially reduces the size of the keys by a factor of r. Recently, the authors of [TZF+24] showcased some asymptotic reduction in the size of relinearization keys. We compare their approach to ours later in Sec. 3. In this paper, we investigate methods to tweak the relinearization sub-routine in order to reduce the total size of MLWE relinearization keys even further asymptotically.

2.3 Use of HE in computation protocols with multiple parties

In this section, we give a brief description of the different types of homomorphic encryption procedures that have been adopted in MPC protocols. Specifically, we distinguish three different (but not mutually exclusive) directions based on the scheme's *access structure* following [MTBH21]. Let us consider a set $P = \{P_1, P_2, \cdots, P_k\}$ of k parties that take part in the protocol. The secret key $\texttt{sk}$ of an HE scheme being used in the protocol will be a function of k secret keys corresponding to k parties. Let $\texttt{sk} = \mathcal{F}(s_1, \cdots, s_k)$ such that the secret key s_i belongs to party i in the protocol and $1 \leq i \leq k$. The access structure of the scheme is then defined as the set $\mathcal{P} \subset \texttt{PowerSet}(P))$ of all such parties that can collectively reconstruct the secret-key. The homomorphic evaluation sub-routines are tweaked accordingly to accommodate the access structure. We use the term *multi-client* for a collective reference to all the three research directions.

Multi-party homomorphic encryption (MPHE): A *threshold* or *multi-party* refers to a t-of-k or k-of-k access structure respectively, with $t < k$. In a k-of-k access structure, all k parties contribute their individual public keys to generate a single shared or joint public key. In a t-of-k structure however, t out of the total k parties have the ability to reconstruct the secret key such that a ciphertext can be decrypted using only t out of the k secret key shares. A multi-party access structure needs to be fixed in the setup phase of the protocol and requires all parties in $\mathcal{P}$ during the decryption phase. Moreover, the homomorphic evaluation algorithms remain unchanged as they operate on ciphertexts encrypted with the single joint public key. Such a protocol is described in [MTBH21].

Multi-key homomorphic encryption (MKHE): A *multi-key* homomorphic encryption scheme does not need a fixed access structure that is defined at the setup phase. Each party encrypts their message using their own public keys and the homomorphic evaluations on the server are adapted to perform operations between ciphertexts encrypted under different keys. The resultant of such evaluations can be considered to be a ciphertext

that is encrypted with an on-the-fly secret key with respect to any number of parties that are actively involved in the computations. Since homomorphic evaluations are performed among ciphertexts encrypted under different keys, the relinearization or key-switching algorithm in the multi-key setting has increased space and computational complexities compared to MPHE. In the following part, we provide some details about the multi-key multiplication/relinearization algorithms and the improvements proposed by [CDKS19]. Taking inspiration from the multi-key version of TFHE proposed in [CCS19], the authors of [CDKS19] construct a multi-key versions of CKKS [CKKS17, CHK+18] and BFV [FV12] with packed ciphertexts. Each party i generates a secret key s_i and a public key $b_i = (-a \cdot s_i + e_i) \in \mathcal{R}_q$ using a common public polynomial a. A ciphertext related to k different parties $\mathtt{ct} = (c_0, \cdots, c_k) \in \mathcal{R}_q^{k+1}$ is then decrypted under the concatenated secret $(1, s_1, \cdots, s_k)$ such that $m \approx c_0 + \sum_{i=1}^{k} c_i \cdot s_i$. Thus, the multiplication of two ciphertexts $\mathtt{ct}_1$ and $\mathtt{ct}_2$ result in an extended non-linear ciphertext containing encryptions $s_i \cdot s_j$, similar to the MLWE setting. One way proposed by [CDKS19] to generate the relinearization key with respect to such ciphertexts encrypting $s_i \cdot s_j$ is by multiplying the j-th public key b_j by the i-th evaluation key $\mathtt{evk}_i$. More specifically, $\mathtt{evk}_i$ is given by the triplet, $\mathtt{evk}_i = (\mathtt{evk}_{i0}, \mathtt{evk}_{i1}, \mathtt{evk}_{i2}) \in \mathcal{R}_q^3$ such that $\mathtt{evk}_{i1}$ is a uniformly randomly polynomial in $\mathcal{R}_q$, $\mathtt{evk}_{i0} = -\mathtt{evk}_{i1} \cdot s_i + e_{i1} + v_i \cdot \mathbf{g}$ (where v_i is randomly sampled from some suitable distribution governed by the scheme's security requirements) and $\mathtt{evk}_{i2} = v_i \cdot a + e_{i2} + s_i \cdot \mathbf{g}$, where $\mathbf{g}$ is a gadget vector based on a chosen gadget decomposition as in [CDKS19]. Let c'_{ij} refer to the component of the multiplied ciphertext $\mathtt{ct}'$ consisting of the encryption of the product of secret components $s_i \cdot s_j$. Now let us consider the multiplication of suitable b_j with c'_{ij} such that $c''_{ij} = \langle \mathbf{g}^{-1}(c'_{ij}), b_j \rangle$. Then the multiplication modulo q of the components of $\mathtt{evk}_i$ to c''_{ij} can be written as follows,

$$\begin{aligned} c''_{ij} \cdot (\mathtt{evk}_{i0}, \mathtt{evk}_{i1}) \cdot (1, s_i) &\approx v_i \cdot c''_{ij}, \text{and,} \\ \langle \mathbf{g}^{-1}(c'_{ij}), \mathtt{evk}_{i2} \rangle \cdot s_j &\approx \langle \mathbf{g}^{-1}(c'_{ij}), -v_i \cdot b_j + s_i \cdot s_j \cdot \mathbf{g} \rangle \\ &= -v_i \cdot c''_{ij} + c'_{ij} \cdot s_i \cdot s_j. \end{aligned}$$

Observe that adding the two equations together cancels out $v_i \cdot c''_{ij}$ giving the required term, $c'_{ij} \cdot s_i \cdot s_j$. It is intuitive to see that a similar scenario during relinearization also arises in MLWE-HE because of the existence of r polynomials in the secret key. Thus, it is a natural curiosity to apply the aforementioned improvement technique to MLWE-HE. However, this solution is not directly applicable in this case because of the different structures of the public keys in the RLWE-based multi-key setting and MLWE-HE and hence we have to deviate substantially from the relinearization key structure of [CDKS19]. We discuss our proposed improvement in Sec. 3.

Multi-group homomorphic encryption (MGHE): In [KLSW24], the authors define the concept of a *multi-group* homomorphic encryption (MGHE) setting which essentially interpolates between MPHE and MKHE. Fig. 1 gives a visual description of MGHE: it consists of multiple "groups" of parties wherein each group has joint (shared) public and evaluation keys. In the multi-group setting, we will use lowercase letters in subscript to indicate scheme components belonging to individual parties and lowercase superscript for ownership of scheme components in a group. For example, let $\{I_1, I_2, \cdots, I_k\}$ be the groups of parties such that $I = \bigcup_{1 \le j \le k} I_j$. Thus within a group I_j, parties, say $P_i^j \in I_j$ generate individual public keys $\mathtt{pk}_i$ which is accumulated to form the joint public key $\mathtt{pk}^j$ of the group which is then used to encrypt messages, thereby acting as a multi-party setting. Note that the public/evaluation keys are shared within the group, but are different across groups. Thus, there is a multi-key setting among groups wherein ciphertexts encrypted under different keys can be operated upon using multi-key homomorphic evaluation algorithms. We elaborate upon the security and correctness notions of the MGHE scheme defined

in [KLSW24] as we base proofs of security and correctness of our MLWE-based MGHE scheme on these definitions.

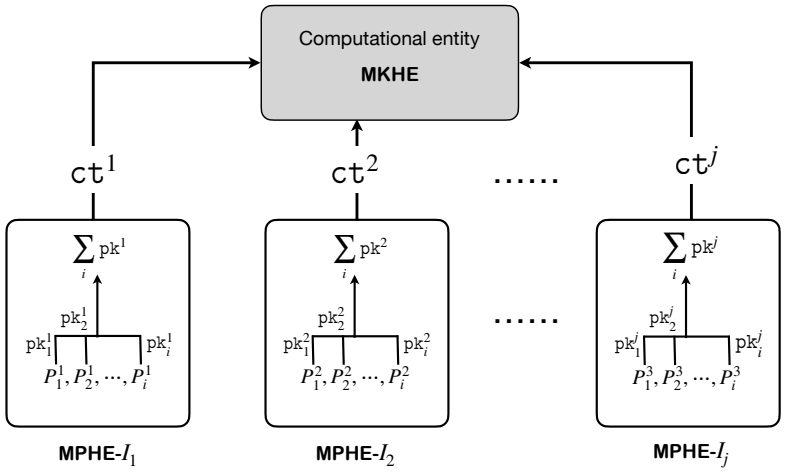

Figure 1: MGHE emulates MPHE within groups and MKHE among groups

Definition of security of MGHE: Let $\{I_1, I_2, \cdots, I_k\}$ be sets (groups) of parties such that $I = \bigcup_{1 \leq j \leq k} I_j$. Let $A \subseteq I$ denote the set of adversarial parties and $H = I \backslash A$. An MGHE scheme is said to be secure if the advantage of any PPT adversary $\mathcal{A}$ in the following game is negligible:

- The challenger generates a public parameter set $pp \leftarrow \texttt{Setup}(1^\lambda, 1^L)$ with λ being the security parameter and L being the maximum circuit evaluation level.
- The challenger executes the key generation protocol $\texttt{KeyGen}(pp, I_j)$ for all $1 \leq j \leq k$. The challenger sends the groups' public keys $\texttt{pk}^1, \cdots, \texttt{pk}^k$ (and similarly, relinearization/key-switching keys) and secret key shares of parties in A that is, $\{[\texttt{sk}_i]^j : \mathcal{A}_i \in A, 1 \leq j \leq k\}$ to the adversary.
- The adversary chooses messages m_0, m_1 in the message space $\mathcal{M}$ and picks an index j such that $I_j \not\subseteq A$, and sends them to the challenger. The challenger samples a random bit $b \in \{0, 1\}$ and using the public key $\texttt{pk}^j$ of I_j, sends $\texttt{Enc}(\texttt{pk}^j; m_b)$ back to the adversary.
- The adversary $\mathcal{A}$ then outputs a bit b'. The advantage is defined as $|Pr[b = b'] - 1/2|$.

Thus, the advantage of the adversary $\mathcal{A}$ in the above game is negligible when $\mathcal{A}$ is unable to distinguish between the distribution of the encryption of m_0 from that of m_1.
Definition of correctness of MGHE: Let $pp \leftarrow \texttt{Setup}(1^\lambda, 1^L)$ be the public parameter set generated as in the security definition before. For $1 \leq j \leq k$, let $\texttt{pk}^j \leftarrow \texttt{KeyGen}(I_j)$ be a public key generated by a set of parties $\{P_i : P_i \in I_j\}$ and $\texttt{ct}^j \leftarrow \texttt{Enc}(\texttt{pk}^j; m_j)$ be an encryption of a message m_j. An MGHE scheme is said to be correct if for any circuit $\mathcal{C} : \mathcal{M}^k \rightarrow \mathcal{M}$ with maximum evaluative level is L, the following holds with an

overwhelming probability in λ:

$$\begin{aligned}&\texttt{DistDec}\Big(\{P_i^j : I_j \in I\}; \texttt{Eval}(\texttt{pk}^1, \cdots, \texttt{pk}^k; \mathcal{C}, \texttt{ct}^1, \cdots, \texttt{ct}^k)\Big)\\ &\qquad = \mathcal{C}(m_1, \cdots, m_k).\end{aligned}$$

where `DistDec` refers to the 'distributed decryption' subroutine. Thus, the correctness of the MGHE scheme ensures that evaluation, '`Eval`' of the encrypted data yields the same result after decryption as that of the evaluation of the circuit on plaintext $m_1, \cdots, m_k$.

General MPC protocols from M(P/K/G)HE: Using the security and correctness guarantees of the underlying HE scheme, general MPC protocol constructions can perform computations in the homomorphic domain. Such a protocol can be endowed with the desired functionality by specifying the access structure of the HE scheme.

3 Overcoming the drawback: $\mathcal{O}(r)$ relinearization keys

As we note from the homomorphic multiplication subroutine provided in Sec. 2.2, ciphertext-ciphertext multiplication of the form $\texttt{ct} \cdot \texttt{ct}'$ also results in the intrinsic squaring of the secret $\mathbf{s} \otimes \mathbf{s}$. In [MAM+24], the relinearization operation required distinct relinearization keys corresponding to each multiplied secret component of the form $s_i \cdot s_j$, without considering duplicated entries of (i, j) versus (j, i). We observe that it is not necessary to generate distinct keys for all such terms. We first explain why the method discussed in Sec. 2.3 cannot be applied to MLWE-HE and then move on to explain our approach of key generation and the consequent changes in the relinearization algorithm.

The RLWE-MKHE relinearization technique from Sec. 2.3 cannot be applied to MLWE-HE because of the difference in the structures of the public keys in RLWE-MKHE and MLWE-HE. While the public key of each party b_i in RLWE-MKHE is an RLWE pair consisting of the secret key s_i of parties P_i, in MLWE-HE, the public key $\mathbf{b}$ is an MLWE pair A consisting of the entire secret $\mathbf{s} = (s_0, \cdots, s_{r-1})$. Thus, $\mathbf{b}$ cannot be used as easily as each b_i is utilized for relinearization in an RLWE-MKHE setting.

First attempt: Instead of directly adopting the method from [CDKS19], suppose we generate two different forms of relinearization keys with a function, $f : \mathcal{R}^r \to \mathcal{R}_{Pq}^r$ of the secret vector $\mathbf{s}$ such that the pair, $(\mathbf{b}_1, \mathbf{b}_2)$ consist of r elements in the ring each. First, we provide a motivation for the structure of $(\mathbf{b}_1, \mathbf{b}_2)$ and then we show the need for certain tweaks to finally obtain a relinearization key consisting of three components $\texttt{evk} = (\texttt{evk}_1, \texttt{evk}_2, \texttt{evk}_3)$.

Consider the pair, $(\mathbf{b}_1, \mathbf{b}_2) = (f(\mathbf{s}) \boxdot \mathbf{s} + \mathbf{e}_1, -f(\mathbf{s}) + \mathbf{e}_2 + P \cdot \mathbf{s})$, with the operator $\boxdot$ defined such that each ring component of the module element $\mathbf{b}_i$, $i \in \{1, 2\}$ has the form, $(b_{1i} = f(\mathbf{s}) \cdot s_i + e_{1i}, b_{2i} = -f(\mathbf{s}) + e_{2i} + P \cdot s_i)_{0 \leq i < r}$ modulo $P \cdot q$, where $P(\lambda, q) > 0$ is an auxiliary integer chosen as per specifications in [CKKS17, MAM+24]. Then, $\langle (b_{1j}, b_{2i}), (1, s_j) \rangle = (f(\mathbf{s}) \cdot s_j + e_{1j}) + (-f(\mathbf{s}) \cdot s_j + e_{2i} + P \cdot s_i \cdot s_j) \approx P \cdot s_i \cdot s_j$, provides a valid encryption of $s_i \cdot s_j$ for relinearization for $0 \leq i, j < r$.

Concerns with correctness and security: We make two observations about the structure of the keys: first, to ensure correct decryption after relinearization, the function f must satisfy *associativity* such that, $\mathbf{d}_m \cdot (f(\mathbf{s}) \boxdot \mathbf{s}) = (\mathbf{d}_m \cdot f(\mathbf{s})) \boxdot \mathbf{s}$ for $m \in \{2, 3\}$ and we can choose $f(\mathbf{s})$ to be $\mathbf{A}_{\text{evk}} \cdot \mathbf{s}$, where $\mathbf{A}_{\text{evk}} \in \mathcal{R}_{Pq}^{r \times r}$ (while taking into account the correct dimensions of the vector $\mathbf{d_2}$ and $\mathbf{d_3}$) is the randomly generated public matrix used in relinearization for associativity to hold. Second, although such a relinearization key structure solves the problem of quadratic key growth proportional to the rank of the module, the pair $(\mathbf{b}_1, \mathbf{b}_2)$ deviates from a traditional MLWE pair. Specifically, each row of

the key $\mathbf{b_1}$, that is, each b_{1i} can be seen as an RLWE encryption under the 'smaller' secret polynomial $s_i \in \mathcal{R}$ instead of an MLWE encryption under the full secret $\mathbf{s} \in \mathcal{R}^r$ and hence may suffer loss of security and be vulnerable against key recovery attacks by solving the RLWE problem for each of the smaller degree polynomials s_i.

Table 2: Sizes (in bits) of relinearization keys for RNS variants of CKKS, [MAM+24] and this work. Recall that if a single polynomial in MLWE-HE has degree N then its RLWE equivalent (of comparable security) has degree $N \cdot r$. For the seeded approach, we exclude the size of the public seed (polynomial a or matrix $\mathbf{A}$).

LWE variant/approach	Unseeded	Seeded
RLWE [CHK+18]	$2NrL(L+1)\log(q_i)/2$	$NrL(L+1)\log(q_i)/2$
MLWE [MAM+24]	$Nr(r+1)^2L(L+1)\log(q_i)/2$	$Nr(r+1)L(L+1)\log(q_i)/2$
MLWE [This work]	$2Nr^2L(L+1)\log(q_i)/2$	$2NrL(L+1)\log(q_i)/2$

Final modifications: We introduce an additional matrix $\mathbf{A'}_{\text{evk}} \in \mathcal{R}_{Pq}^{r\times r}$ which has been randomly sampled from a uniform distribution over $\mathcal{R}_{Pq}^{r\times r}$ and write the relinearization key triple modulo $P \cdot q$ as, $(\texttt{evk}_1, \texttt{evk}_2, \texttt{evk}_3) = (-\mathbf{A'}_{\text{evk}} \cdot \mathbf{s} + (\mathbf{A}_{\text{evk}} \cdot \mathbf{s}) \boxdot \mathbf{s} + \mathbf{e}_1, -\mathbf{A}_{\text{evk}} \cdot \mathbf{s} + \mathbf{e}_2 + P \cdot \mathbf{s}, \mathbf{A'}_{\text{evk}})$. Let us consider the case of $r = 2$ such that the (non-RNS) relinearized ciphertext is given by $\texttt{ct}_{\texttt{relin}} = (\mathbf{d'_0}, \mathbf{d'_1})$, corresponding to the non-relinearized ciphertext $\texttt{ct}_{\texttt{mult}} = (d_0, \mathbf{d_1}, \mathbf{d_2}, d_3) \in \mathcal{R}_q \times \mathcal{R}_q^2 \times \mathcal{R}_q^2 \times \mathcal{R}_q$ with $\mathbf{d_2}$ and d_3 being the non-linear (in $\mathbf{s}$) components, as described in Sec. 2.2 with a detailed structure description in Sec. A. Let, $(d''_0, \mathbf{d''_1})$ denote the relinearized components such that, $d''_0 = d_{20}\cdot\texttt{evk}_{10} + d_{21}\cdot\texttt{evk}_{11} + d_3\cdot\texttt{evk}_{10}$, $d''_{10} = d_{20} \cdot (\texttt{evk}_{20} + \texttt{evk}_3[0][0]) + d_3 \cdot (\texttt{evk}_{21} + \texttt{evk}_3[0][0]) + d_{21} \cdot \texttt{evk}_3[1][0]$, and, $d''_{11} = d_{20} \cdot \texttt{evk}_3[0][1] + d_{21} \cdot (\texttt{evk}_{21} + \texttt{evk}_3[1][1]) + d_3 \cdot \texttt{evk}_3[0][1]$, where $\texttt{evk}_3[i][j]$ refers to the i-th entry of the j-th column in the matrix $\mathbf{A'}_{\text{evk}}$. The decryption equation of $\texttt{ct}_{\texttt{relin}}$ with respect to $(1, \mathbf{s})$ modulo q can then be written as follows:

$$
\begin{aligned}
d'_0 \cdot 1 + \mathbf{d'_1} \cdot \mathbf{s} &= (d_0 + P^{-1} \cdot d''_0) + (\mathbf{d_1} + P^{-1} \cdot \mathbf{d''_1}) \cdot \mathbf{s} \\
&\approx d_0 + P^{-1} \cdot \big(d_{20} \cdot ((-\mathbf{A'}_{\text{evk}} \cdot \mathbf{s})_0 + (\mathbf{A}_{\text{evk}} \cdot \mathbf{s})_0 \cdot s_0) \\
&\quad + d_{21} \cdot ((-\mathbf{A'}_{\text{evk}} \cdot \mathbf{s})_1 + (\mathbf{A}_{\text{evk}} \cdot \mathbf{s})_1 \cdot s_1) + d_3 \cdot ((-\mathbf{A'}_{\text{evk}} \cdot \mathbf{s})_0 + (\mathbf{A}_{\text{evk}} \cdot \mathbf{s})_0 \cdot s_0)\big) \\
&\quad + P^{-1} \cdot \big(d_{10} + d_{20} \cdot ((-\mathbf{A}_{\text{evk}} \cdot \mathbf{s})_0 + \mathbf{A'}_{\text{evk}}[0][0] + P \cdot s_0) + d_{21} \cdot \mathbf{A'}_{\text{evk}}[1][0] \\
&\quad + d_3 \cdot ((-\mathbf{A}_{\text{evk}} \cdot \mathbf{s})_0 + \mathbf{A'}_{\text{evk}}[0][0] + P \cdot s_1)\big) \cdot s_0 \\
&\quad + P^{-1} \cdot \big(d_{11} + d_{20} \cdot \mathbf{A'}_{\text{evk}}[0][1] + d_{21} \cdot ((-\mathbf{A}_{\text{evk}} \cdot \mathbf{s})_1 + \mathbf{A'}_{\text{evk}}[1][1] + P \cdot s_1) \\
&\quad + d_3 \cdot \mathbf{A'}_{\text{evk}}[0][1]\big) \cdot s_1 \\
&\approx d_0 + d_{10} \cdot s_0 + d_{11} \cdot s_1 + d_{20} \cdot s_0^2 + d_{21} \cdot s_1^2 + d_3 \cdot s_0 \cdot s_1
\end{aligned}
$$

These relinearization keys can be generated using alg. 1. The relinearized components $(d''_0, \mathbf{d''_1})$ take a slightly different form compared to [MAM+24], as shown below:

$$
(d''_0) = \Big(\mathbf{d_2} \cdot \texttt{evk}_1 + \sum_{t=0}^{r-1} d_{3_{tt'}} \cdot \texttt{evk}_{1\texttt{t}}\Big), t' > t
$$

$$
(d''_{1t'}) = \begin{cases} \Big(d_{2t'}(\texttt{evk}_{2\texttt{t}'} + \texttt{evk}_3[t'][t']) + \sum_{t=0}^{r-1} d_{3_{tt'}}(\texttt{evk}_3[t][t']) + d_{2t}(\texttt{evk}_3[t][t'])\Big), \text{if } t' > t \\ \Big(d_{2t'}(\texttt{evk}_{2\texttt{t}'} + \texttt{evk}_3[t'][t']) + \sum_{t=0}^{r-1} d_{3_{tt'}}(\texttt{evk}_{2\texttt{t}} + \texttt{evk}_3[t'][t']) + d_{2t}(\texttt{evk}_3[t][t'])\Big), \text{else} \end{cases}
$$

where $t' < r$ and $\mathbf{d_2}$ and $\mathbf{d_3}$ are assumed to have been lifted from q to $P \cdot q$ using the modulus switching technique [CKKS17, MAM+24]. For better readability, we refrain from using '$\cdot$' explicitly to denote the multiplication of $d_{2t'}$ and $d_{3_{tt'}}$ with the respective relinearization keys in the above equation. The relinearization algorithm using the above equations is given in alg. 2 and the corresponding noise analysis is given in appendix C. We observe that both $\texttt{evk}_1$ and $\texttt{evk}_2$ consists of r ring polynomials each and thus the proposed technique reduces the number of relinearization keys from $\frac{r\cdot(r+1)}{2}$ to, $r + r = 2r$. We provide the concrete sizes in Table 2.

Addressing security: We note that the relinearization key triple deviates from traditional relinearization key structure constructed in [CKKS17] or in [MAM+24]. Therefore, we provide a security argument by considering MLWE circular security for the relinearization key, $\texttt{evk} = (\texttt{evk}_1, \texttt{evk}_2, \texttt{evk}_3)$ in the following way: $\texttt{evk}_3$ is just the public matrix $\mathbf{A'}_{\text{evk}} \in \mathcal{R}_{Pq}^{r\times r}$. Next, each row of the second component $\texttt{evk}_2$ along with the relinearization matrix $\mathbf{A}_{\text{evk}} \in \mathcal{R}_{Pq}^{r\times r}$ is an MLWE instance under $\mathbf{s} \in \mathcal{R}^r$ that encrypts individual secret polynomials s_i for $0 \leq i < r$. Again, each row of $\texttt{evk}_1$ can be considered as an MLWE instance under $\mathbf{s} \in \mathcal{R}^r$ and the public matrix $\mathbf{A'}_{\text{evk}} \in \mathcal{R}_{Pq}^{r\times r}$ encrypting each row $(\mathbf{A}_{\text{evk}}\cdot\mathbf{s})_i \cdot s_i$ of the term $(\mathbf{A}_{\text{evk}} \cdot \mathbf{s}) \boxdot \mathbf{s}$ for $0 \leq i < r$. We recall that this particular term led to security loss in the construction described in *first attempt* as it lacked the additional 'mask' $\mathbf{A'}_{\text{evk}} \cdot \mathbf{s}$ required for $\texttt{evk}_1$ to be contained in the module $\mathcal{R}_{Pq}^r$.

In [TZF+24], the authors propose using a partial RNS decomposition (with the number of RNS moduli being $\mu < L$) and a temporary rank-up during relinearization wherein the relinearization key is encrypted under a 'larger' secret key $\mathbf{s}||\mathbf{s'}$ in a module of rank $(r + r')$ with a temporary bigger modulus $P' \cdot q > P \cdot q$. Their key structure is of the form $(-\mathbf{A}_{\text{evk}} \cdot (\mathbf{s}||\mathbf{s'}) + \mathbf{e}_1 + P' \cdot \mathbf{s} \otimes \mathbf{s}, \mathbf{A}_{\text{evk}}) \in \mathcal{R}_{P'q}^{(1+r')r(r+1)/2}$. This key structure still consists of the terms $s_i \cdot s_j$ and hence increases quadratically in r. A direct comparison of the bit-sizes is not straightforward due to the partial RNS decomposition step, however, using our technique could also make their sizes grow linearly in r.

In: Secret key, $\texttt{sk} = (s_0, \cdots, s_{r-1}) \in \mathcal{R}^r$, error vectors $\mathbf{e}_{\text{evk}_1}$ and $\mathbf{e}_{\text{evk}_2}$.
In: Public matrices, $\mathbf{A}_{\text{evk}}, \mathbf{A'}_{\text{evk}} \in \mathcal{R}_{Pq}^{r\times r}$
Out: $\texttt{evk} = (\texttt{evk}_1, \texttt{evk}_2, \texttt{evk}_3 = \mathbf{A'}_{\text{evk}}) \in \mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^{r\times r}$

```
temp1 ← −A'_evk · s
temp2 ← A_evk · s
for k = 0 to r − 1 do
    evk_1k ← temp1_k + temp2_k · s_k + e_evk_1k (mod Pq)
    evk_2k ← −temp2_k + e_evk_2k + P · s_k (mod Pq)
end for
```

Algorithm 1: `EvkGen` Algorithm

In: Non-relinearized ciphertext, $\texttt{ct}_{\texttt{mult}} = (d_0, \mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}) \in \mathcal{R}_q \times \mathcal{R}_q^r \times \mathcal{R}_q^r \times \mathcal{R}_q^{r(r-1)/2}$
In: $(d_0'', \mathbf{d_1''}) \in \mathcal{R}_q \times \mathcal{R}_q^r$ derived from equations aforementioned in the text.
Out: $\texttt{ct}_{\texttt{relin}} = (c_0', \mathbf{c_1''}) \in \mathcal{R}_q \times \mathcal{R}_q^r$

```
c'_0 ← (d_0 + P^{-1} · d''_0) (mod q)
for k = 0 to r − 1 do
    c'_1k ← (d_1k + P^{-1} · d''_1k) (mod q)
end for
```

Algorithm 2: `Relinearization` Algorithm

4 Multi-group homomorphic encryption from module lattices: MLWE-MGHE

In this section, we take a step forward and explore an MLWE-based MGHE scheme. Before going onto the mathematical nuances of the scheme, we list the main features of such an approach as a result of applying homomorphic encryption techniques and the underlying MLWE problem:

- *Non-interactive key generation:* By virtue of the underlying homomorphic encryption scheme, joint public keys for groups can be obtained without requiring individual parties to interact among each other. Moreover, apart from requiring the joint public key of the group, as explored by works such as [CDKS19, KLSW24], no interaction is required corresponding to the relinearization keys for the multi-key scheme working among groups.
- *Flexibility in the choice of parameters:* As explained in Sec. 1, an MLWE-based HE scheme allows for a flexible choice of the module rank by fixing a base ring degree. Moreover, it also allows dynamic parameter management during homomorphic evaluations through rank reduction which is also inherited in MLWE-MGHE.
- *Flexibility in the number of participating entities:* In the subsequent part, we also show how to extend the functionality of rank reduction to offer a dynamic access structure within groups using a sub-routine similar to rank reduction, thereby proposing a fulfillment of requirement-2 (Sec. 1).
- *Security assurances:* As mentioned in works proposing MLWE-based cryptographic constructions such as [MAM+24, SAB+21, BDK+21], our multi-group homomorphic encryption scheme also inherits better security reliability compared to its RLWE-based counterparts.

4.1 A concrete instantiation

Now, we provide details of the MLWE-based MGHE construction. Let us first consider a groups of parties $\bigcup_{1\leq j\leq k} I_j$ such that party $P_i^j \in I_j$. MGHE simulates an MPHE-like environment within each I_j and an MKHE-like environment among the different groups of parties, I_j. For the sake of brevity of notations, we will denote the public matrices used by all parties for generating either the public keys or the relinearization/key-switching keys as $\mathbf{A}$ and the error terms mostly with $\mathbf{e}_i$ (in practice, errors are freshly generated for each key generation). Discussions on correctness of important subroutines are given in appendix B.

- **Setup($1^\lambda, 1^L$):** Given the security parameter λ, generate the public parameter set pp. The set pp consists of the fixed ring degree N, the rank of the module r, the maximum multiplicative level L, the ciphertext modulus q, the auxiliary integer P and the various parameters for secret and error distributions that include certain fixed hamming weight h, variance σ^2 and probability parameter ρ.
- **Key generation:** Public key generation requires common public matrix $\mathbf{A} \in \mathcal{R}_q^{r\times r}$ that is used by all parties, as per the CRS (common random string) model. Individual parties P_i^j belonging to group index j generate their share of secret key $[\mathbf{s}_i]^j$ via `KeyGen.sk()` and use it to also generate respective public keys $[\mathtt{pk}_i]^j$ according to the `KeyGen.pk()` algorithm. Each party generates its relinearization key triple $[(\mathtt{evk}_{i1}, \mathtt{evk}_{i2}, \mathtt{evk}_{i3})]^j$ and key-switching key tuple $[(\mathtt{ksk}_{i1}, \mathtt{ksk}_{i2})]^j$ following `KeyGen.evk()` and `KeyGen.ksk()` respectively. The key-switching key generation algorithm is used to facilitate additional operations such as rotation and conjugation.

Joint keys of a group are generated in `KeyGen.jk()` by summing up individual public keys of all the participating parties in the group. All the key generation sub-routines are provided in Fig. 2.

- **Encryption:** In a group, a message m is encrypted using the joint public key given by `KeyGen.jk()` following the MLWE-based encryption algorithm $\texttt{MLWE.Enc}_{\texttt{pk}}\texttt{()}$. A multi-group ciphertext encrypting messages of multiple groups belonging to the union $\bigcup_j I_j$ of ordered sets $\{I_1, \cdots, I_j\}_{1 \leq j \leq k}$ would contain information about the joint public keys of the 'multi-group' set and would accordingly be decrypted by the joint secret key of the set. The encryption algorithm is given in `MGHE.Enc()` in Fig. 3. We use the notation $\bigcup_j$ in the superscript to specifically denote that the number of groups in the union is strictly greater than one. We demonstrate this further in Section B.

- **Ideal decryption and distributed decryption:** An ideal decryption algorithm assumes a global joint secret key for $\bigcup_j I_j$ and follows the `MLWE.Dec()` procedure A. Distributed decryption is a more feasible real-world scenario where it is assumed that parties do not communicate to obtain the joint secret key. Each party $P_i^j \in I_j$ carries out a partial decryption and sends it to the next before adding a fresh noise with certain bounds on the variance σ'^2 of the noise distribution. This technique known as *smudging* has been discussed in [AJLA+12, MTBH21]. The final decrypted result is an accumulation of the partial decryptions. This method is given in `MGHE.Dec()` in Fig. 3.

  Multi-group ciphertexts in homomorphic operations: Before formulating the algorithms for homomorphic addition and multiplication operations between multi-group ciphertexts, we align them according to the strategy followed by [KLSW24]. Consider homomorphic operations between a multi-group ciphertext $\texttt{ct}$ encrypting the group secret keys $\{\mathbf{s}^1, \cdots, \mathbf{s}^{j'}\}$ corresponding to the ordered sets $\{I_1, \cdots, I_{j'}\}$ and another ciphertext $\texttt{ct}'$ encrypting the group secret keys $\{\mathbf{s}^1, \cdots, \mathbf{s}^{j''}\}$ corresponding to the ordered sets $\{I_1, \cdots, I_{j''}\}$. Then, we extend both ciphertexts by padding zeroes and rearranging the position of their components if required, so that both these ciphertexts are decryptable under the group secret keys $\{\mathbf{s}^1, \cdots, \mathbf{s}^{j'}, \cdots, \mathbf{s}^{j''}\}$ corresponding to the union of the the two previous ordered sets, that is, $\{I_1, \cdots, I_{j'}\} \bigcup \{I_1, \cdots, I_{j''}\}$ with $1 \leq j', j'' \leq k$. In the following algorithms for homomorphic addition and multiplication, we assume that the ciphertexts have undergone this step already.

- **Homomorphic addition:** Since addition is a linear operation, it easily extends to the MGHE setting such that addition of two ciphertexts $\texttt{ct}$ and $\texttt{ct}'$ results in the ciphertext $\texttt{ct}_{\texttt{add}} = \texttt{ct} + \texttt{ct}'$, as shown in `MGHE.Add()`, Fig. 4.

- **Homomorphic multiplication/evaluation:** The underlying algorithmic flow of homomorphic multiplication in an MGHE setting is similar to that of `MLWE.Mult()`. The differences arise in the complexity of the actual individual component-level computations based on the number of participating groups. We discuss this subroutine in `MGHE.Mult()`, Fig. 4.

- **Key-switching and relinearization procedures:** As explained in the key-switching and relinearization key generation procedures, in the MGHE setting, different parties generate their individual key-switching and relinearization keys, and the linear sum of these individual keys result in joint keys. Since the key-switching algorithm in MGHE follows `MLWE.swk()` given in appendix A, we do not reiterate it again in this section. Instead, we describe the changes to the relinearization algorithm.

Key generation for MLWE-based MGHE

KeyGen: Each party $P_i^j \in I_j$ generates $(\mathbf{s}_i,\ \mathtt{pk}_i, \mathtt{evk}_i, \mathtt{ksk}_i)$ using the following algorithms with arithmetic done modulo q, the key-switching key and relinearization key generation done modulo $P \cdot q$.

- `KeyGen.sk()`:
    1. Sample a secret key share $\mathbf{s}_i \in \mathcal{R}^r$ from a secret distribution similar to `MLWE.KeyGen.sk()`.
- `KeyGen.pk()`:
    1. **In:** Error vector $\mathbf{e}_i$ with each error polynomial $e_i \leftarrow \mathcal{DG}(\sigma^2)$, own secret key $\mathbf{s}_i$ and a common public matrix $\mathbf{A}$ shared between all groups.
    2. **Out:** Individual public keys, $\mathtt{pk}_i \leftarrow \mathtt{MLWE.KeyGen.pk}(\mathbf{A}, \mathbf{s}_i) \in \mathcal{R}_q^r \times \mathcal{R}_q^{r\times r}$. Joint public key of the form $\mathtt{pk}^j$ is a simple sum of the individual keys. We define this sum formally in `KeyGen.jk()`.
- `KeyGen.evk()`:
    1. **In:** Error vector $\mathbf{e}_i$ with each error polynomial $e_i \leftarrow \mathcal{DG}(\sigma^2)$, own secret key $\mathbf{s}_i$ and $\mathbf{u}_i \in \mathcal{HWT}(h)$ and common public matrices $\mathbf{A}$, $\mathbf{A}' \in \mathcal{R}_{Pq}^{r\times r}$.
    2. **Out:** Individual relinearization key $\mathtt{evk}_i$ which is a triplet of the form, $(\mathtt{evk}_{i1}, \mathtt{evk}_{i2}, \mathtt{evk}_{i3}) = \big((-\mathbf{A}\cdot\mathbf{s}_i) + \mathbf{e}_{i1}, -\mathbf{A}\cdot\mathbf{u}_i + \mathbf{e}_{i2} + P\cdot\mathbf{s}_i, -\mathbf{A}'\cdot\mathbf{s}_i + \mathbf{e}_{i3} - P\cdot\mathbf{u}_i\big) \in \mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^r$.
- `KeyGen.ksk()`:
    1. **In:** Error vector $\mathbf{e}_i$ with each error polynomial $e_i \leftarrow \mathcal{DG}(\sigma^2)$, own secret key $\mathbf{s}_i$ and a common public matrix $\mathbf{A}$.
    2. **Out:** Individual key-switching key $\mathtt{ksk}_i$ which is a pair of the form, $(\mathtt{ksk}_{i1}, \mathtt{ksk}_{i2}) = \big(-\mathbf{A}\cdot\mathbf{s}_i + \mathbf{e}_{1i} + P\cdot\phi(\mathbf{s}_i), \mathbf{A}\big) \in \mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^{r\times r}$.
- `KeyGen.jk()`:
    1. **In:** Individual public keys, relinearization keys and key-switching keys of parties, $(\mathtt{pk}_i, (\mathtt{evk}_{i1}, \mathtt{evk}_{i2}, \mathtt{evk}_{i3}), (\mathtt{ksk}_{i1}, \mathtt{ksk}_{i2}))$.
    2. **Out:** Joint public and key-switching keys of groups with the form, $(\mathtt{pk}^j, (\mathtt{evk}_1, \mathtt{evk}_2, \mathtt{evk}_3)^j, (\mathtt{ksk}_1, \mathtt{ksk}_2)^j)$ $= (\sum_i \mathtt{pk}_i, (\sum_i \mathtt{evk}_{i1}, \sum_i \mathtt{evk}_{i2}, \sum_i \mathtt{evk}_{i3}), (\sum_i \mathtt{ksk}_{i1}, \sum_i \mathtt{ksk}_{i2}))$.

Figure 2: Key generation algorithms for MLWE-MGHE.

Encryption and decryption for MLWE-based MGHE

Enc-Dec: Using joint keys generated from the above-mentioned algorithms, one or more groups can encrypt message m into a ciphertext (and recover m later) following the en(de)cryption algorithms given below:

- `MGHE.Enc()`:

  if $\bigcup_j I_j = I_j$ for some j:

  1. **In:** Joint public key of a group $\mathtt{pk}^j$, a message m and error vector $\mathbf{e}$ with each error polynomial $e_i \leftarrow \mathcal{DG}(\sigma^2)$.
  2. **Out:** A ciphertext of the group encrypting message m given by, $\mathtt{ct} = (c_0, \mathbf{c_1})^j \in \mathcal{R}_q \times \mathcal{R}_q^r$ using $\mathtt{MLWE.Enc}_{\mathtt{pk}^j}()$

  else:

  1. **In:** Joint public key $\mathtt{pk}^j$ of each group in a 'multi-group' ordered set $\{I_1, \cdots I_j\}$, $1 < j \leq k$, a message m and error vector $\mathbf{e}$ with each error polynomial $e_i \leftarrow \mathcal{DG}(\sigma^2)$.
  2. **Out:** A ciphertext of the group encrypting message m given by, $\mathtt{ct} = (c_0, \mathbf{c_1})^{\cup j} \in \mathcal{R}_q \times \mathcal{R}_q^{r \times j}$ using $\mathtt{MLWE.Enc}_{\mathtt{pk}^j}()$.

- `MGHE.Dec()`:

  1. **In:** A general ciphertext, $\mathtt{ct} = (c_0, \mathbf{c_1})^{\cup j} \in \mathcal{R}_q \times \mathcal{R}_q^r$, secret key $\mathbf{s}_i$ of each party $P_i \in I = \bigcup_j I_j$, smudging error e_i' sampled from a distribution $\mathcal{DG}(\sigma'^2)$.
  2. **Intermediate Out:** Partial decryptions, $\mu_i = (\sum_{1 \leq j \leq k} \mathbf{c_1}^{\cup j}) \cdot \mathbf{s}_i + e_i'$.
  3. **Out:** Merged final decryption, $m \approx c_0 + \sum_{i \in I} \mu_i$.

Figure 3: En(de)cryption algorithms for MLWE-MGHE

Synergies and differences with the relinearization key structure in Sec. 3: Using concepts from relinearization key generation techniques discussed in Sec. 3, we reformulate a relinearization procedure for a multi-group setting. However, we list some distinct challenges that we encounter within the MLWE-MGHE framework and discuss their work-arounds.

The first observation is that, unlike in the case of single-party (single-key) MLWE-HE where b_{1i} in our *first attempt* was encrypted only under the partial secret polynomial $s_i \in \mathcal{R}$, since each party P_i^j in the multi-group case holds a complete secret key $\mathbf{s}_i \in \mathcal{R}^r$, hence we can use the idea from our *first attempt* pair in Sec. 3 to generate a relinearization key in the multi-group setting. However, in our proposed relinearization technique (both *first* and *final*) for single-keyed MLWE-HE, the operation $f(\mathbf{s}) \boxdot \mathbf{s}$ is not additively homomorphic over all parties, that is, $f(\mathbf{s}_i) \boxdot \mathbf{s}_i + f(\mathbf{s}_{i'}) \boxdot \mathbf{s}_{i'} \neq f(\mathbf{s}_i + \mathbf{s}_{i'}) \boxdot (\mathbf{s}_i + \mathbf{s}_{i'})$. To allow additive aggregation, we change the structure of the keys in the multi-group scenario. Along with $\mathbf{s}_i$ we generate $\mathtt{evk}_i$ using another vector $\mathbf{u}_i \in \mathcal{R}^r$ sampled from the secret distribution by following [KLSW24] in order to introduce additional randomness and prevent key-recovery attacks from a linear combination of the relinearization key components.

Second, we note that a homomorphic ciphertext multiplication involves intrinsic multiplications between public keys with different secret components with respect to the parties and the groups involved. For instance, in a setting with two groups of

two parties each, $\mathtt{pk}^1 \cdot \mathtt{pk}^2$ will involve the product of the secret keys of group 1 with those of group 2. More specifically, using the pre-processing strategy, the common secret key will contain the union of the secret keys of group 1 and group 2, that is, $\mathtt{sk} = (1, (\mathbf{s}_0 + \mathbf{s}_1), (\mathbf{s'}_0 + \mathbf{s'}_1)) = (1, \mathbf{s}^1, \mathbf{s}^2)$ and so homomorphic multiplication results in products of the form $(\mathbf{s}_0 + \mathbf{s}_1) \cdot (\mathbf{s'}_0 + \mathbf{s'}_1)$. Thus, there must be relinearization keys that are capable of relinearizing components of the multiplication, $\mathbf{s}_i \cdot \mathbf{s'}_{i'}$. At first glance, this gives the impression that the size of the relinearization keys is proportional to $\mathcal{O}(k^2Nr)$, where k denotes the total number of groups. However, since this problem again boils down to handling terms of the product $\mathtt{sk} \otimes \mathtt{sk}$, thus it can be circumvented by using the proposed relinearization key structure in Fig. 2. Notice that the joint relinearization keys $(\mathtt{evk}_1, \mathtt{evk}_2, \mathtt{evk}_3)^1$ and $(\mathtt{evk}_1, \mathtt{evk}_2, \mathtt{evk}_3)^2$ of group 1 and group 2 will contain encryptions of $(\mathbf{s}_0 + \mathbf{s}_1)$ and $(\mathbf{s'}_0 + \mathbf{s'}_1)$ respectively such that relinearization of $\mathtt{ct}_{\mathtt{mult}}$ can be carried out using $(\mathtt{evk}_1, \mathtt{evk}_2, \mathtt{evk}_3)^1$ and $(\mathtt{evk}_1, \mathtt{evk}_2, \mathtt{evk}_3)^2$ with sizes proportional to $\mathcal{O}(kNr)$. Security of the keys are discussed in Lemma 1. The number of multiplications, however, remain proportional to $\mathcal{O}(k^2)$. We consider the following relinearization components for `MGHE.Relin()`.

$$d''_0 = \sum_{1 \le t \le k} \left(\mathbf{d_{2_t}} \cdot {\mathtt{evk}_1}^t \cdot {\mathtt{evk}_3}^t \right) + \sum_{t=1}^{k-1} \mathbf{d_{3_{tt'}}} \cdot {\mathtt{evk}_1}^{t'} \cdot {\mathtt{evk}_3}^t,\ t' > t$$

$$\mathbf{d''_{1t'}} = \begin{cases} \left(\mathbf{d_{2_{t'}}} \cdot ({\mathtt{evk}_1}^{t'} \cdot \mathbf{A'} + {\mathtt{evk}_2}^{t'}) + \sum_{t=1}^{k-1} \mathbf{d_{3_{tt'}}} \cdot ({\mathtt{evk}_1}^{t'} \cdot \mathbf{A'} + {\mathtt{evk}_2}^{t}) \right),\ t' > t \\ \left(\mathbf{d_{2_{t'}}} \cdot ({\mathtt{evk}_1}^{t'} \cdot \mathbf{A'} + {\mathtt{evk}_2}^{t'}) \right), \text{otherwise} \end{cases}$$

where $1 \le t, t' < k$, with k being the total number of participating groups and $\mathbf{d_2}$, $\mathbf{d_3}$ are assumed to have been lifted from q to $P \cdot q$ using modulus switching technique. Note here that the variables t and t' denote the participating groups. Algorithms for multi-group addition, multiplication and relinearization are given in Fig. 4. A demonstration of correctness for the case when $k = 2$ is given in Sec. B.

4.2 Key homomorphic property of joint keys

A multi-client setting using an HE scheme benefits from the additively homomorphic property of public and evaluation keys. This is because the individual public keys correctly add up to the group's joint key only if they are additively homomorphic. We demonstrate this property with respect to the structure of MLWE keys used in our proposed MGHE construction for a group I_j consisting of two parties, which can be trivially extended to any number of parties.

Theorem 1. *Let* $\mathbf{s}_1, \mathbf{s}_2$ *be two secret keys,* $\mathbf{A}$ *be the common public matrix and* $\mathbf{e}_1, \mathbf{e}_2$ *two noise values. Also, let* $\mathtt{pk}_1 = (\mathbf{A}, \mathbf{b}_1) = \mathtt{KeyGen.pk}(\mathbf{A}, \mathbf{s}_1; \mathbf{e}_1)$ *and* $\mathtt{pk}_2 = (\mathbf{A}, \mathbf{b}_2) = \mathtt{KeyGen.pk}(\mathbf{A}, \mathbf{s}_2; \mathbf{e}_2)$*. Then,* $\mathtt{KeyGen.jk}(\mathbf{A}, \mathbf{b}_1 + \mathbf{b}_2) = \mathtt{KeyGen.pk}(\mathbf{A}, \mathbf{s}_1 + \mathbf{s}_2; \mathbf{e}_1 + \mathbf{e}_2)$*.*

Proof. We derive the following equalities to prove our claim:

$$\begin{aligned} \mathtt{KeyGen.jk}(\mathbf{A}, \mathbf{b}_1 + \mathbf{b}_2) &= \big(\mathbf{A}, \mathtt{KeyGen.pk}(\mathbf{A}, \mathbf{s}_1; \mathbf{e}_1) + \mathtt{KeyGen.pk}(\mathbf{A}, \mathbf{s}_2; \mathbf{e}_2)\big) \\ &= \big(\mathbf{A}, (-\mathbf{A} \cdot \mathbf{s}_1 + \mathbf{e}_1) + (-\mathbf{A} \cdot \mathbf{s}_2 + \mathbf{e}_2)\big) \\ &= \big(\mathbf{A}, -\mathbf{A} \cdot (\mathbf{s}_1 + \mathbf{s}_2) + (\mathbf{e}_1 + \mathbf{e}_2)\big) \\ &= \mathtt{KeyGen.pk}(\mathbf{A}, \mathbf{s}_1 + \mathbf{s}_2; \mathbf{e}_1 + \mathbf{e}_2) \end{aligned}$$

□

Homomorphic subroutines for MLWE-based MGHE

The computationally powerful entity (server) can perform the following homomorphic operations, `MGHE.Eval()` on the groups' encrypted data. All arithmetic is done modulo q.

- `MGHE.Add()`:
  1. **In:** Ciphertexts `ct` and `ct'`.
  2. **Out:** Added ciphertext, $\texttt{ct}_{\texttt{add}} = \texttt{ct} + \texttt{ct}' = (c_0 + c_0', \mathbf{c_1} + \mathbf{c_1'})^{\cup j}$ following `MLWE.Add()`.
- `MGHE.Mult()`:
  1. **In:** Ciphertexts `ct` and `ct'`.
  2. **Out:** Multiplied ciphertext, $\texttt{ct}_{\texttt{mult}} = (d_0, \mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3})^{\cup j}$ following `MLWE.Mult()`.
- `MGHE.Relin()`:
  1. **In:** Multiplied ciphertext $\texttt{ct}_{\texttt{mult}} = (d_0, \mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3})^{\cup j}$, joint relinearization key $(\texttt{evk}_1, \texttt{evk}_2, \texttt{evk}_3)^{\cup j} \leftarrow$ `Keygen.jk()` such that for each party P_i, $(\texttt{evk}_{i1}, \texttt{evk}_{i2}, \texttt{evk}_{i3}) \leftarrow$ `KeyGen.evk()`.
  2. **Out:** Relinearized ciphertext, $\texttt{ct}_{\texttt{relin}} = (d_0', \mathbf{d_1'})^{\cup j}$ such that $(d_0' = d_0 + P^{-1} \cdot d_0'', \mathbf{d_1'}^{\cup j} = \mathbf{d_1} + P^{-1} \cdot \mathbf{d_{1t'}''})$ with $(d_0'', \mathbf{d_{1t'}''})$ as given in the equations for d_0'' and $\mathbf{d_{1t'}''}$ aforementioned in the text.

Figure 4: Homomorphic operations for MLWE-MGHE

Theorem 2. *Let* $\mathbf{s}_1, \mathbf{s}_2$ *be two secret keys,* A, A' *be the common public matrices, and* $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ *be the noise values. Also, let* $(\texttt{evk}_{11}, \texttt{evk}_{12}, \texttt{evk}_{13}) = \texttt{KeyGen.evk}(\mathbf{u}_1, \mathbf{s}_1; (\mathbf{e}_{11}, \mathbf{e}_{12}, \mathbf{e}_{13}))$ *and* $(\texttt{evk}_{21}, \texttt{evk}_{22}, \texttt{evk}_{23}) = \texttt{KeyGen.evk}(\mathbf{u}_2, \mathbf{s}_2; (\mathbf{e}_{21}, \mathbf{e}_{22}, \mathbf{e}_{23}))$. *Then,* $\texttt{KeyGen.jk}((\texttt{evk}_{11}, \texttt{evk}_{12}, \texttt{evk}_{13}) + (\texttt{evk}_{21}, \texttt{evk}_{22}, \texttt{evk}_{23})) = \texttt{KeyGen.evk}(\mathbf{u}_1 + \mathbf{u}_2, \mathbf{s}_1 + \mathbf{s}_2; (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3))$.

Proof. Let the parties belong to the group I_j. Then, we can derive the following equalities to prove our claim:

$$
\begin{aligned}
\texttt{KeyGen.jk}((\texttt{evk}_{11}, \texttt{evk}_{12}, \texttt{evk}_{13}) + (\texttt{evk}_{21}, \texttt{evk}_{22}, \texttt{evk}_{23})) &= \big(\texttt{KeyGen.evk}(\mathbf{u}_1, \mathbf{s}_1; (\mathbf{e}_{11}, \mathbf{e}_{12}, \mathbf{e}_{13})) \\
&\quad + \texttt{KeyGen.evk}(\mathbf{u}_2, \mathbf{s}_2; (\mathbf{e}_{21}, \mathbf{e}_{22}, \mathbf{e}_{23}))\big) \\
&= \big((-\mathbf{A} \cdot \mathbf{s}_1) + \mathbf{e}_{11} + ((-\mathbf{A} \cdot \mathbf{s}_2) + \mathbf{e}_{21}), \\
&\quad (-\mathbf{A} \cdot \mathbf{u}_1 + \mathbf{e}_{12} + P \cdot \mathbf{s}_1) + (-\mathbf{A} \cdot \mathbf{u}_2 + \mathbf{e}_{22} + P \cdot \mathbf{s}_2)\big) \\
&\quad (-\mathbf{A}' \cdot \mathbf{s}_1 + \mathbf{e}_{13} - P \cdot \mathbf{u}_1) + (-\mathbf{A}' \cdot \mathbf{s}_2 + \mathbf{e}_{23} - P \cdot \mathbf{u}_2)\big) \\
&= \big(((-\mathbf{A} \cdot (\mathbf{s}_1 + \mathbf{s}_2) + \mathbf{e}_{11} + \mathbf{e}_{12})), \\
&\quad (-\mathbf{A} \cdot (\mathbf{u}_1 + \mathbf{u}_2) + (\mathbf{e}_{21} + \mathbf{e}_{22}) + P \cdot (\mathbf{s}_1 + \mathbf{s}_2))\big), \\
&\quad (-\mathbf{A}' \cdot (\mathbf{s}_1 + \mathbf{s}_2) + (\mathbf{e}_{31} + \mathbf{e}_{32}) - P \cdot (\mathbf{u}_1 + \mathbf{u}_2))\big) \\
&= \texttt{KeyGen.evk}(\mathbf{u}_1 + \mathbf{u}_2, \mathbf{s}_1 + \mathbf{s}_2; (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3))
\end{aligned}
$$

□

4.3 Security of the proposed MLWE-MGHE scheme

Let, I_j be sets ('groups' in MGHE) such that $I = \bigcup_{1 \leq j \leq k} I_j$ and $H = I \backslash A$ for any set $A \subset I$. To prove the security of the proposed MGHE scheme based on the definition given in Sec. 2.3, we consider the following three hybrid games as given in [KLSW24].

- **Game 0:** This is a real world execution of the security game defined in Sec. 2.3.
- **Game 1:** It is similar to **Game 0**, but the challenger samples public keys $\mathtt{pk}_i$ uniformly at random from $\mathcal{R}_q^r \times \mathcal{R}_q^{r \times r}$, the evaluation keys $(\mathtt{evk}_{i1}, \mathtt{evk}_{i2}, \mathtt{evk}_{i3})$ uniformly at random from $\mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^r$ and the key-switching keys $(\mathtt{ksk}_{i1}, \mathtt{ksk}_{i2})$ uniformly at random from $\mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^{r \times r}$ for $P_i \in H$.
- **Game 2:** It is similar to **Game 1**, but the challenger encrypts 0 instead of a message m_b.

Lemma 1. *With respect to the three games described above, the proposed MGHE-MLWE scheme is semantically secure under the MLWE security assumption with suitable parameters (N, r, q, χ), where χ refers to error and secret distributions considered in the scheme.*

Proof. Sketch: Let $\mathtt{pk}_i$ and $(\mathtt{evk}_{i1}, \mathtt{evk}_{i2}, \mathtt{evk}_{i3})$ be the public and relinearization keys of party $P_i \in H$. Since $\mathtt{pk}_i$ and $\mathtt{evk}_{i1}$ follow the MLWE distribution, so by the definition of MLWE security, these keys are indistinguishable from a uniform distribution over $\mathcal{R}_q^r \times \mathcal{R}_q^{r \times r}$ and $\mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^{r \times r}$. For the structures of $\mathtt{evk}_{i2}$ and $\mathtt{evk}_{i3}$, note that they can be considered as MLWE encryptions of one secret key $\mathbf{s}_i$ under another secret $\mathbf{u}_i$ and vice versa, hence we rely on the circular security assumption [ACPS09, BGK11, BHHI10, MTY11], which is a commonly used notion to assess the security of RLWE or MLWE-based HE schemes. Similarly, the key-switching keys $\mathtt{ksk}_{i1}$ is an MLWE encryption of $\phi(\mathbf{s}_i)$ under the secret key $\mathbf{s}_i$ with the random matrix $\mathtt{ksk}_{i2}$. Thus under the above-mentioned assumptions, **Game 0** and **Game 1** are computationally indistinguishable.

For the case of **Game 1** and **Game 2**, the adversary sends an index j corresponding to a group I_j to the challenger, such that the encryption key being used has the form, $\mathtt{pk}^{\text{game}} = \sum_{P_i \in I_j \cap A} \mathtt{pk}_i + \sum_{P_i \in I_j \cap H} \mathtt{pk}_i$. Since $I_j \cap H$ is non-empty and each public key $\mathtt{pk}_i$ is uniformly sampled from $\mathcal{R}_q^r$ for all $P_i \in H$, hence the sum of the multiple such public keys also resembles a uniform distribution over $\mathcal{R}_q^r$. Again, under the MLWE assumption, this implies that either 0 or m_b encrypted using such a key would be computationally indistinguishable from each other. Thus, the advantage for an adversary between **Game 1** and **Game 2** would be negligible. Hence, finally, the advantage of an adversary in a real world execution of the MGHE scheme, which is **Game 0** will be negligible. Thus, the proposed MLWE-MGHE scheme possesses semantic security according to the definition in Sec. 2.3 against semi-malicious corruptions.

□

5 Flexibility features with different levels of granularity

We recall that in this work we aimed to explore the design of an MLWE-based multi-group HE scheme that integrates the flexibility of both MLWE-HE and multi-group HE. In this section, we present a detailed analysis of how the proposed MLWE-MGHE scheme meets these flexibility requirements, addressing the two objectives outlined at the outset of this work.

5.1 Towards requirement-1: flexibility in scheme parameters

A recent work [Sma23] applies hybrid homomorphic encryption (HHE) in the multi-party setting to show further reduction in the client's communication costs when compared to plain HE. Using HHE requires a larger parameter set because before the server can operate on the encrypted data, it has to spend certain multiplicative levels to perform the homomorphic decryption of the symmetric circuit. For single-keyed MLWE-HE, [MAM+24] discussed that dynamic ciphertext compression using rank reduction could be beneficial in HHE-based applications as the server can drop the module rank and work with smaller ciphertexts after the homomorphic symmetric decryption. Hence, we extend this functionality to the multi-group setting in Fig. 5 to complement multi-client HHE. We demonstrate this in the `MGHE.RankRed()` algorithm. Note that, for correctness of evaluations following a rank reduction, each group participating in the computation must drop the required rank from their ciphertexts.

Parameter flexibility for MLWE-based MGHE

The rank reduction procedure in the multi-group setting, `MGHE.RankRed()` takes a group ciphertext $\texttt{ct} \in \mathcal{R}_q \times \mathcal{R}_q^r$ and transforms it into a ciphertext of a reduced rank, $\texttt{ct}_{red} \in \mathcal{R}_q \times \mathcal{R}_q^{r'}$ such that $r' < r$. Each party segregates secret key components into $\mathbf{s'}_i = (s_0, \cdots, s_{r'-1})_i \in \mathcal{R}^{r'}$ and $\mathbf{s''}_i = (s_{r'}, \cdots, s_{r-1})_i \in \mathcal{R}^{r-r'}$ and then generates their rank reduction keys using the following method:

`MGHE.KeyGen.RedKey():`

1. **In:** Error vector $\mathbf{e}_i$ with each error polynomials sampled from $\mathcal{DG}(\sigma^2)$, own secret key $\mathbf{s}_i$.
2. **Out:** Individual rank reduction keys $(\texttt{redk}_{i1}, \texttt{redk}_{i2})$ using `MLWE.KeyGen.RedKey()`, where $\mathbf{A}_{\text{red}}$ is also common to all parties according to the CRS model.

`MGHE.RankRed():`

1. **In:** Group ciphertext $\texttt{ct} \in \mathcal{R}_q \times \mathcal{R}_q^r$ and joint rank reduction key of the group $\texttt{redk}^j = (\texttt{redk}_1, \texttt{redk}_2)^j \leftarrow$ `MGHE.KeyGen.jk()` where individual rank reduction keys are generated using `MGHE.KeyGen.RedKey()`.
2. **Out:** Rank reduced ciphertext, $\texttt{ct}_{red} \leftarrow \texttt{MLWE.RankRed}(\texttt{ct}, \texttt{redk}^j)$.

Figure 5: Rank reduction algorithm for MLWE-MGHE

5.2 Towards requirement-2: flexibility in access structure

An MGHE scheme allows flexible participation among groups due to the underlying multi-key access structure. However, within a group, a multi-party access structure requires participation from a fixed number of parties. We propose a key-switching-like procedure to also allow additional flexibility in the access structure among parties contained in a group. As discussed in the previous section, the rank reduction subroutine enables to decrypt the ciphertext with a secret key of a lower module rank instead of the original key. This is facilitated via a key-switching-like operation, where the server is given an additional rank-reduction key using which it can 'remove' the desired secret key components without requiring access to the actual secret key of the client. In a similar way, a flexible access structure also corresponds to an update or modification of the secret key of the group.

Thus, we propose the following modifications to the rank reduction algorithm to allow a flexible access structure among parties. Let $\mathbf{s}'' = (\mathbf{s}_{t+1} + \cdots + \mathbf{s}_k) \in \mathcal{R}^r$ such that $(\mathbf{s}_{t+1}, \cdots, \mathbf{s}_k)$ are the secret keys of the parties that want to leave the group. Note that this procedure requires an additional interaction round through *confidential channels*. We define the following formal sub-routine called `FlexStruct` in Fig. 6 to demonstrate this functionality.

Access structure flexibility for MLWE-based MGHE$_{confidential}$

The flexible access structure procedure in the multi-group setting, takes a group ciphertext $\mathtt{ct} \in \mathcal{R}_q \times \mathcal{R}_q^r$ involving secret key of k parties and transforms it into a ciphertext, $\mathtt{ct}_{flex} \in \mathcal{R}_q \times \mathcal{R}_q^r$ involving encryptions of t parties such that $t < k$. It requires secret keys segregated into $\mathbf{s}' = (\mathbf{s}_1 + \cdots + \mathbf{s}_t) \in \mathcal{R}^r$ and $\mathbf{s}'' = (\mathbf{s}_{t+1} + \cdots + \mathbf{s}_k) \in \mathcal{R}^r$ through confidential channels.

`MGHE.FlexKey():`

1. **In:** Error vector $\mathbf{e}_i$ with each error polynomial sampled from $\mathcal{DG}(\sigma^2)$, own secret key $\mathbf{s}_i$.
2. **Out:** The flex key, $(\mathtt{flexk}_1, \mathtt{flexk}_2)^j$ using `MGHE.KeyGen.jk()` and `MLWE.KeyGen.RedKey()` with $\mathbf{A}_{\text{red}} \in \mathcal{R}_{Pq}^{r \times r}$.

`MGHE.FlexStruct():`

1. **In:** Group ciphertext $\mathtt{ct} \in \mathcal{R}_q \times \mathcal{R}_q^r$ and flex key $(\mathtt{flexk}_1, \mathtt{flexk}_2)^j$.
2. **Out:** New ciphertext, $\mathtt{ct}_{\mathtt{flex}} \leftarrow \mathtt{MLWE.RankRed}\big((\mathtt{flexk}_1, \mathtt{flexk}_2)^j\big)$.

Figure 6: Flexible access structure algorithms with confidential channels

In [MTBH21], the authors discuss a method to dynamically adjust the number of participating entities without using confidential channels. We adopt their method in the following algorithm `PublicFlexStruct` in Fig. 7 to also enable flexible access structure using public components and no interaction.

Implementation: We used the currently available SageMath [S+24] implementation of the MLWE-HE scheme to instantiate a proof-of-concept relinearization with the proposed approach for single-party setting as well as a multi-group setting with the MGHE sub-routines discussed in the paper. We ran our implementation[2] on a MacBook Air with Apple M1 chip and macOS Sonoma version 14.6.1. Without considering the time for key-generation, the modified relinearization technique (for the 1-party 1-group case) has an increased run-time when compared with [MAM+24]'s relinearization due to the fact that the number of key-ciphertext multiplications increase, e.g., 6-multiplications in [MAM+24] versus 9-multiplications in our case for $r = 2$. We provide some run-time numbers in the repository. However, note that we refrain from formally mentioning implementation timing results of our proof-of-concept implementation in this paper as these numbers cannot be used for reasonable comparison with the highly optimized RLWE-based HE or multi-party implementations [SEA21, Ins23]. An optimized library that incorporates the single and multi-client MLWE-HE setting remains an ongoing work.

[2] https://github.com/anishamukh/MLWE-MGHE.git

Access structure flexibility for MLWE-based MGHE$_{public}$

Like `FlexStruct`, this procedure also takes a group ciphertext $\mathtt{ct} \in \mathcal{R}_q \times \mathcal{R}_q^r$ involving secret key of k parties and transforms it into a ciphertext, $\mathtt{ct}_{flex} \in \mathcal{R}_q \times \mathcal{R}_q^r$ involving encryptions of t parties such that $t < k$. The only difference is that it requires the joint public key $\mathtt{pk}'$ encrypting the secret vector $\mathbf{s}' = (\mathbf{s}_1, \cdots, \mathbf{s}_t)$ for remaining set of parties $P' = \{P_1, \cdots, P_t\}^j$ in some group I_j. Also let, $\mathbf{s}'' = (\mathbf{s}_{t+1}, \cdots, \mathbf{s}_k)$ be the secret keys corresponding to the leaving set of parties $P'' = \{P_{t+1}, \cdots, P_k\}^j$ in the same group I_j.

`MGHE.PublicFlexStruct():`

1. **In:** Error terms e_{0i} and $\mathbf{e}_{1i}$ with each polynomial sampled from $\mathcal{DG}(\sigma^2)$, public key $\mathtt{pk}'^j$ of the set P' and vector $\mathbf{v}_i$ with each of its polynomials sampled from $\mathcal{ZO}(\rho)$ as in `MLWE.Enc()`.
2. **In:** Using `MGHE.jk()` and secret key $\mathbf{s}_i$, $t+1 \leq i \leq k$ of parties $P_i \in P''$, the following is computed: $(c_0'', \mathbf{c}_1'') = \mathtt{MGHE.jk}(\mathbf{s_i} \cdot \mathbf{c_1} + \mathbf{v}_i \cdot \mathtt{pk}'^j[0] + e_{0i}, \mathbf{v}_i \cdot \mathtt{pk}'^j[1] + \mathbf{e}_{1i})_{t+1 \leq i \leq k} = (\mathbf{s}'' \cdot \mathbf{c_1} + \sum_i \mathbf{v}_i \cdot \mathtt{pk}'^j[0] + \sum_i e_{0i}, \sum_i \mathbf{v}_i \cdot \mathtt{pk}'^j[1] + \sum_i \mathbf{e}_{1i})$.
3. **In:** Group ciphertext $\mathtt{ct} = (c_0, \mathbf{c_1}) \in \mathcal{R}_q \times \mathcal{R}_q^r$.
4. **Out:** New ciphertext, $\mathtt{ct_{flex}} = (c_0 + c_0'', \mathbf{c}_1 + \mathbf{c}_1'') \in \mathcal{R}_q \times \mathcal{R}_q^r$.

Figure 7: Flexible access structure algorithms without confidential channels

6 MPC from MGHE-MLWE

While MPHE and MKHE are both viable options for building an MPC protocol, each have limitations that restrict their usefulness in certain applications. For example, MPHE-based MPC protocols require fixed set of parties to generate a shared key. On the other hand, MKHE schemes are more time and space intensive than MPHE because ciphertexts expand as they interact with other ciphertexts under different keys. Thus, an MGHE scheme that integrates the strengths of both these schemes can be used to construct MPC protocols bestowed with different levels of flexibilities. In the following part, we discuss the construction of an MPC protocol $\pi_{\mathcal{C}}$ given in Fig. 8 for a polynomial-time deterministic circuit $\mathcal{C}$ based on our MGHE construction along the lines of [KLSW24]. Note that, while the procedure for generating joint public and evaluation keys differ slightly from [KLSW24], we do not claim novelty in the protocol structure itself as it closely follows that of [KLSW24].

In HE-based MPC protocols, the evaluation step in `Phase III` is non-interactive and does not require any private input from the parties. Thus, it eliminates the need for non-collusion assumptions required in traditional non-HE MPC applications. As such, the role of the computing party can either be taken up by a semi-honest cloud server (like we assumed in the protocol above) or by any one or more of the designated parties in I.

6.1 Security of MPC from semi-malicious setting

We provide a security proof for our proposed MPC protocol in the semi-malicious setting according to the definitions discussed in [AJLA+12, KLSW24]. We refrain from reiterating the proof of security of the distributed decryption as it can be directly inferred from [KLSW24].

$\pi_{\mathcal{C}}$: MPC protocol from MGHE

`Setup:` In this stage, a common public parameter set pp to be used in the key generation phase is setup based on the security parameter λ of the scheme.
`Input:` A circuit $\mathcal{C} : \mathcal{M}^d \rightarrow \mathcal{M}$ that evaluates over the input vector $\mathbf{x} = \{x_1, \cdots, x_d\}$ in the encrypted domain.

`Phase I:` In the first phase of the protocol, all involved parties $P_i \in I = \bigcup_{1 \leq j \leq k} I_j$ generate their individual secret and public keys, in the form of $(\mathbf{s}_i, \mathtt{pk}_i) \leftarrow$ (`KeyGen.sk()`, `KeyGen.pk()`), and broadcast their public keys according to the agreed-upon security parameters of the scheme. This phase can be considered to be independent from the rest of the protocol because it is carried out only once, and can be done offline.

`Phase II:` After all parties have communicated their individual public keys, corresponding joint keys of the groups I_j can be obtained by adding all individual keys, that is, $\mathtt{pk}^j \leftarrow$ `KeyGen.jk()`. A party P_i can use the joint public key of the group to encrypt the corresponding input x_i into a ciphertext $\mathtt{ct}_t \leftarrow$ `MGHE.Enc`$(x_t, \mathtt{pk}^j)$ for $1 \leq t \leq d$, $1 \leq j \leq k$ and broadcasts it to a cloud server or any third party for homomorphic computation of the circuit. Additionally, each party generates its individual relinearization keys $(\mathtt{evk}_{i1}, \mathtt{evk}_{i2}, \mathtt{evk}_{i3}) \leftarrow$ `KeyGen.evk()` and key-switching keys $(\mathtt{ksk}_{i1}, \mathtt{ksk}_{i2})$ such that the joint keys computed as, $((\mathtt{evk}_1, \mathtt{evk}_2, \mathtt{evk}_3), (\mathtt{ksk}_1, \mathtt{ksk}_2))^j \leftarrow$ `Keygen.jk()`, is then available to the computational entity.

`Phase III:` This is the evaluation phase where there is no interaction involved from the parties as the required key-switching/relinearization components can be generated by the server using the public material already provided by the parties in the aforementioned phases. The homomorphic computations take place according to $\mathtt{ct}_{\mathtt{eval}} \leftarrow$ `MGHE.Eval`$(\mathcal{C}, \mathtt{ct}_1, \cdots, \mathtt{ct}_\mathtt{d}; \mathtt{pk}^j, \mathtt{evk}^j, \mathtt{ksk}^j)$. After that, the computing party or the cloud server sends the output of the homomorphic evaluation back to the parties for them to decrypt it via an interactive distributed decryption protocol.

`Final Output:` The final output of the protocol is the decrypted result of the homomorphic evaluation given by, $m \leftarrow$ `MGHE.Dec`$(\mathtt{I}, \sigma'; \mathtt{ct}_{\mathtt{eval}})$ performed in `Phase III`.

Figure 8: A multi-party protocol from an MLWE-based multi-group homomorphic scheme

Theorem 3. ***Security against semi-malicious adversary:*** *Let $\mathcal{C}$ be any deterministic poly-time function with d inputs and single output. Let the parameters of the scheme be chosen in a way that they satisfy the general MGHE-MPC constraints and consider the case that the corresponding MLWE assumptions holds. Then the protocol $\pi_{\mathcal{C}}$ securely UC-realizes*[3] *the ideal functionality $\mathcal{F}_{\mathcal{C}}$ in the presence of a static semi-malicious adversary corrupting $n-1$ parties.*

Proof. Consider a set I_c of corrupt parties such that $|I_c| \leq n-1$. Let, $|I_c| = n-1$ so that there is only one honest party P_h. We describe a simulator S against a static semi-malicious adversary $\mathcal{A}$ as follows:
The simulator, S: Let the simulator sample public key, relinearization and key-switching keys each from uniform distribution over $\mathcal{R}_q^r$ instead of using the `MGHE.KeyGen.pk()`, `MGHE. KeyGen.evk()` and `MGHE.KeyGen.ksk()` subroutines for the honest party P_h in the `KeyGen()` phase. Then in the encryption phase, it encrypts 0 instead of a real messages m from P_h. As the simulator has access to the inputs and secret keys of all parties except P_h from the witness tape, the simulator can evaluate the circuit $\mathcal{C}$ on ciphertexts $\mathtt{ct}_1, \cdots, \mathtt{ct}_d$ and obtain the resulting ciphertext $\mathtt{ct}_{\mathtt{eval}}$. In addition, it also receives the output message m from the ideal functionality. Finally, the simulator computes the partial decryption for the honest party P_h with the assumption that simulated and real partial decryption are indistinguishable. Next, we define the following hybrid games to prove indistinguishability of the real and ideal world executions of the protocol.

- $\mathsf{REAL}_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$: This is exactly an execution of the protocol $\pi_{\mathcal{C}}$ in the real world with environment Z and semi-malicious adversary $\mathcal{A}$.
- $\mathsf{HYB}^1_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$: This corresponds to the real world game $\mathsf{REAL}_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$ except for the fact that it publishes the simulated partial decryption [KLSW24].
- $\mathsf{HYB}^2_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$: This game is similar to $\mathsf{HYB}^1_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$, except for the fact that the honest party P_h encrypts 0.
- $\mathsf{IDEAL}_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$: This is the ideal-world execution with simulator S and environment Z.

On the basis of the games described above, we make the following claims.

- Claim 3.1: $\mathsf{REAL}_{(\pi_{\mathcal{C}},\mathcal{A},Z)} \equiv \mathsf{HYB}^1_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$.

  Proof. Since the simulated partial decryption is statistically indistinguishable from real partial decryption [KLSW24], thus the adversary $\mathcal{A}$ learns no extra information than with a real partial decryption. Hence, the two games mentioned above are also statistically indistinguishable. □

- Claim 3.2: $\mathsf{HYB}^1_{(\pi_{\mathcal{C}},\mathcal{A},Z)} \equiv \mathsf{HYB}^2_{(\pi_{\mathcal{C}},A,Z)}$.

  Proof. Note that while $\mathsf{HYB}^1_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$ corresponds to Game 0, $\mathsf{HYB}^2_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$ corresponds to Game 2 in Sec. 4.3. Thus, they are indistinguishable based on the arguments presented in the section for indistinguishability of Game 0 and Game 2. □

- Claim 3.3: $\mathsf{HYB}^2_{(\pi_{\mathcal{C}},\mathcal{A},Z)} \equiv \mathsf{IDEAL}_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$.

  Proof. Again, note here that $\mathsf{IDEAL}_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$ corresponds to Game 1 in Sec. 4.3. Thus, our claims holds due to the proof of indistinguishability of Game 1 and Game 2 in Sec. 4.3. □

[3]We work in the standard universal composability framework of Canetti[Can01]

- Finally, based on Claim 3.1, 3.2 and 3.3, the equivalence of the real world execution $\mathsf{REAL}_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$ and ideal world execution $\mathsf{IDEAL}_{(\pi_{\mathcal{C}},\mathcal{A},Z)}$ of the protocol holds.

□

7 Discussion on current constraints of MLWE-MGHE

In the previous sections we highlighted various benefits and flexibility factors of the MLWE-based MGHE scheme. To keep this discussion comprehensive and fair, in this section, we also consider the following constraints with respect to our proposed scheme. We note that these limitations do not arise from our construction, but are either inherited from the underlying MLWE-HE primitive or are based on MPC-specific scenarios.

- *More computations than RLWE-based multi-client proposals:* While using an MLWE-based HE primitive allows to perform computations with 'smaller' polynomials, the number of such computations are higher during homomorphic multiplication than in RLWE-HE. Thus, even in the multi-client setting, the computational entity (server) has to perform more polynomial multiplications during a ciphertext-ciphertext multiplication for MLWE-MGHE when compared to RLWE-based multi-client proposals. This computational limitation would, however, not pose a challenge in applications requiring only an additively homomorphic scheme (such as secure aggregation in encrypted federated learning) and an MLWE-MGHE scheme can then be used without any substantial performance degradation. We mention here a similar but more acute observation in this context by the author of [Sma23] which emphasizes that network improvements are inherently limited by physical constraints, while computational performance continues to grow rapidly. Consequently, FHE-based MPC, which relies more on computation than communication, is expected to become more useful than traditional communication-heavy MPC approaches in the long term. In such a scenario, the varying degrees of flexibility provided by MLWE-MGHE could be particularly useful.
- *Considering more varied threat models:* Zero-knowledge proof systems have been proposed as a way of extending the security model from semi-malicious attackers to fully malicious or active adversaries as discussed in [KLSW24, RST+22] but their real-life feasibility needs further research. Works such as [YAZ+19, BLS19, GNS23] that explore zero-knowledge proofs for lattice-based schemes could significantly contribute to broaden the scope of study of adversarial models in HE-based MPC protocols.

8 Conclusion and future work

In this work, we first set out to simplify the relinearization key-generation algorithm for the MLWE-based variant of the CKKS scheme [MAM+24]. We demonstrated that the number of relinearization keys can be reduced from $\mathcal{O}(r^2)$ to $\mathcal{O}(r)$ without increasing the computational complexity of the relinearization algorithm. We then took the first step to apply the MLWE-HE primitive to the multi-group setting. We extended the functionalities of the single-keyed MLWE-based CKKS variant and proposed multi-group sub-routines inspired from a multi-party-like access structure within a group of parties and multi-key-like access structure among groups. Finally, we presented two levels of flexibility features of our proposed construction and also discussed a few of its current constraints. As homomorphic encryption continues to gain traction in emerging privacy-preserving MPC applications, we hope that this work serves as a motivation for further investigation of MLWE-based schemes in this context.

As future work, an in-depth analysis to find the right balance between the module rank and the error distribution is required to ensure better security bounds without penalizing efficiency [BJRW23]. This is important because the error propagation and the extent of the efficiency bottleneck of an MLWE-HE construction are directly linked to the increased number of computations during homomorphic evaluation. This in turn is dependent on the rank of the module being chosen during the setup phase. The authors in [MAM+24] also report observing more precision loss due to increased error propagation during experiments. Furthermore, in applications that do not require many multiplicative levels and largely rely on additively homomorphic evaluations (such as secure aggregation in federated learning discussed in Sec. 1), it would be interesting to measure concrete performance metrics for MLWE-HE-based constructions versus its RLWE-based counterparts. Since these applications would not have a large multiplicative depth, MLWE-HE primitives could benefit from effective parallel processing and smaller individual computations compared to its RLWE-HE counterparts.

References

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 595–618, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-03356-8_35.

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 483–501, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-29011-4_29.

[BDK+21] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium. Proposal to NIST PQC Standardization, Round3, 2021. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[BGK11] Zvika Brakerski, Shafi Goldwasser, and Yael Tauman Kalai. Black-box circular-secure encryption beyond affine functions. In Yuval Ishai, editor, *Theory of Cryptography*, pages 201–218, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-19571-6_13.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electron. Colloquium Comput. Complex.*, page 111, 2011. URL: https://eccc.weizmann.ac.il/report/2011/111.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 423–444, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-13190-5_22.

[BJRW23] Katharina Boudgoust, Corentin Jeudy, Adeline Roux-Langlois, and Weiqiang Wen. On the hardness of module learning with errors with short distributions. *J. Cryptol.*, 36(1):1, 2023. URL: https://doi.org/10.1007/s00145-022-09441-3, doi:10.1007/S00145-022-09441-3.

[BLS19] Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 176–202. Springer, 2019. doi:10.1007/978-3-030-26948-7_7.

[BS23] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 371–404. Springer, 2023. doi:10.1007/978-981-99-8721-4_12.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, 2011. doi:10.1109/FOCS.2011.12.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959888.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from TFHE. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 446–472. Springer, 2019. doi:10.1007/978-3-030-34621-8_16.

[CDKS19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 395–412, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3319535.3363207.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–299. Springer, 2001. doi:10.1007/3-540-44987-6_18.

[CDW17] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-svp. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 324–348, Cham, 2017. Springer International Publishing. doi:10.1007/978-3-319-56620-7_12.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020. doi:10.1007/S00145-019-09319-X.

[CHK+18] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In Carlos Cid and Michael J. Jacobson Jr., editors, *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, volume 11349 of *Lecture Notes in Computer Science*, pages 347–368. Springer, 2018. doi:10.1007/978-3-030-10970-7_16.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 409–437. Springer, 2017. doi:10.1007/978-3-319-70694-8_15.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast Private Set Intersection from Homomorphic Encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1243–1255, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3133956.3134061.

[CS16] Ana Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016*, pages 325–340, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-29485-8_19.

[Fer23] Ramsès Fernàndez-València. Verifiable encodings in multigroup fully homomorphic encryption. *CoRR*, abs/2303.08432, 2023. URL: https://doi.org/10.48550/arXiv.2303.08432, arXiv:2303.08432, doi:10.48550/ARXIV.2303.08432.

[FH96] Matthew K. Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. *J. Cryptol.*, 9(4):217–232, 1996. doi:10.1007/BF00189261.

[FTSH20] David Froelicher, Juan Ramón Troncoso-Pastoriza, Joao Sa Sousa, and Jean-Pierre Hubaux. Drynx: Decentralized, secure, verifiable system for statistical queries and machine learning on distributed datasets. *IEEE Trans. Inf. Forensics Secur.*, 15:3035–3050, 2020. doi:10.1109/TIFS.2020.2976612.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012. URL: http://eprint.iacr.org/2012/144.

[Gen09] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009.

[GNS23] Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: Snarks for ring arithmetic. *J. Cryptol.*, 36(4):41, 2023. URL: https://doi.org/10.1007/s00145-023-09481-3, doi:10.1007/S00145-023-09481-3.

[Ins23] Tune Insight. Lattigo v5. Online: https://github.com/tuneinsight/lattigo, November 2023. EPFL-LDS, Tune Insight SA.

[KLSW24] Hyesun Kwak, Dongwon Lee, Yongsoo Song, and Sameer Wagh. A general framework of homomorphic encryption for multiple parties with non-interactive key-aggregation. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 403–430, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-54773-7_16.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-13190-5_1.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015. doi:10.1007/s10623-014-9938-4.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1219–1234. ACM, 2012. doi:10.1145/2213977.2214086.

[MAM+24] Anisha Mukherjee, Aikata, Ahmet Can Mert, Yongwoo Lee, Sunmin Kwon, Maxim Deryabin, and Sujoy Sinha Roy. Modhe: Modular homomorphic encryption using module lattices potentials and limitations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):527–562, 2024. URL: https://doi.org/10.46586/tches.v2024.i1.527-562, doi:10.46586/TCHES.V2024.I1.527-562.

[MTBH21] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *Proc. Priv. Enhancing Technol.*, 2021(4):291–311, 2021. URL: https://doi.org/10.2478/popets-2021-0071, doi:10.2478/POPETS-2021-0071.

[MTY11] Tal Malkin, Isamu Teranishi, and Moti Yung. Efficient circuit-size independent public key encryption with kdm security. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 507–526, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-20465-4_28.

[PLZ24] Jeongeun Park, Barry Van Leeuwen, and Oliver Zajonc. FINALLY: A multikey FHE scheme based on NTRU and LWE. *IACR Commun. Cryptol.*, 1(3):15, 2024. URL: https://doi.org/10.62056/aebn-4c2h, doi:10.62056/AEBN-4C2H.

[RST+22] Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for SPDZ. *J. Cryptol.*, 35(1):5, 2022. URL: https://doi.org/10.1007/s00145-021-09416-w, doi:10.1007/S00145-021-09416-W.

[S+24] W. A. Stein et al. *Sage Mathematics Software (Version 10.3)*. The Sage Development Team, 2024. http://www.sagemath.org.

[SAB+21] Peter Schwabe, Roberto Avanzi, Joppe Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehle. CRYSTALS-KYBER. Proposal to NIST PQC Standardization, 2021.

https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[SEA21] Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL, September 2021. Microsoft Research, Redmond, WA.

[Sma23] Nigel P. Smart. Practical and efficient fhe-based MPC. In Elizabeth A. Quaglia, editor, *Cryptography and Coding - 19th IMA International Conference, IMACC 2023, London, UK, December 12-14, 2023, Proceedings*, volume 14421 of *Lecture Notes in Computer Science*, pages 263–283. Springer, 2023. doi:10.1007/978-3-031-47818-5_14.

[TZF+24] Wenxu Tang, Fangyu Zheng, Guang Fan, Tian Zhou, Jingqiang Lin, and Jiwu Jing. Dpad-he: Towards hardware-friendly homomorphic evaluation using 4-directional manipulation. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 2475–2489. ACM, 2024. doi:10.1145/3658644.3690280.

[YAZ+19] Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, Zuoxia Yu, and William Whyte. Efficient lattice-based zero-knowledge arguments with standard soundness: Construction and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 147–175. Springer, 2019. doi:10.1007/978-3-030-26948-7_6.

[ZPGS19] Wenting Zheng, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 724–738. IEEE, 2019. doi:10.1109/SP.2019.00045.

A MLWE-based HE sub-routines

In this section, we provide a summary of the MLWE-based HE subroutines proposed by [MAM+24]. In order for notations to be comprehensible, we only reiterate the non-RNS version of the algorithms.

Key generation and encryption in MLWE-HE

KeyGen: The `MLWE.KeyGen()` algorithm generates the triples $(\texttt{sk}, \texttt{pk}, \texttt{ksk})$ using the following sub-algorithms. We use a general notation $\mathbf{A}$ to denote the public matrix used during public key or key-switching key generations.

`KeyGen.sk()`:

1. Generate a secret key $\texttt{sk} = (1, \mathbf{s})$ such that each secret polynomial $s_i \leftarrow \mathcal{HWT}(h)$ where the $\mathcal{HWT}(h)$ is the set of signed binary polynomials $\{0, \pm 1\}^N$ with Hamming weight h.

`KeyGen.pk()`:

1. **In:** Error vector $\mathbf{e}$ with each error polynomial $e_i \leftarrow \mathcal{DG}(\sigma^2)$ which is a secret Gaussian distribution with $\sigma > 0$, secret key $\mathbf{s}$.
2. **Out:** Public key of the form, $\texttt{pk} = (-\mathbf{A} \cdot \mathbf{s} + \mathbf{e}, \mathbf{A}) \in \mathcal{R}_q^r \times \mathcal{R}_q^{r \times r}$.

`KeyGen.ksk()`:

1. **In:** Error vector $\mathbf{e}$ with each error polynomial $e_i \leftarrow \mathcal{DG}(\sigma^2)$ which is a secret Gaussian distribution with $\sigma > 0$, secret key $\mathbf{s}$ and a function $\phi(\mathbf{s}) = \mathbf{s}'$ of the initial secret $\mathbf{s}$.
2. **Out:** Key-switching key of the form, $\texttt{swk} = (\mathbf{b}, \mathbf{A}) = (-\mathbf{A} \cdot \mathbf{s} + \mathbf{e} + P \cdot \mathbf{s}', \mathbf{A}) \in \mathcal{R}_{Pq}^r \times \mathcal{R}_{Pq}^{r \times r}$.

`KeyGen.RedKey()`:

1. **In:** Error vector $\mathbf{e}$ with each error polynomial $e_i \leftarrow \mathcal{DG}(\sigma^2)$ which is a secret Gaussian distribution with $\sigma > 0$, secret key $\mathbf{s}$ segregated into two vectors $\mathbf{s}' = (s_0, \cdots s_{r'-1}) \in \mathcal{R}^{r'}$ and $\mathbf{s}'' = (s_{r'}, \cdots, s_{r-1}) \in \mathcal{R}^{r-r'}$.
2. **Out:** Rank reduction key of the form, $\texttt{redk} = (b, \mathbf{A}_{\text{red}}) = (-\mathbf{A}_{\text{red}} \cdot \mathbf{s}' + \mathbf{e}_{\text{red}} + P \cdot \mathbf{s}'', \mathbf{A}_{\text{red}}) \in \mathcal{R}_{Pq}^{r-r'} \times \mathcal{R}_{Pq}^{(r-r') \times r'}$

$\texttt{MLWE.Enc}_{\texttt{pk}}()$:

1. **In:** Public key $\texttt{pk}$, a message m, a vector $\mathbf{v} \in \mathcal{R}_q^r$ where each polynomial v_i of is sampled from $\mathcal{ZO}(0.5)$ where the distribution $\mathcal{ZO}(\rho)$ with $0 \leq \rho \leq 1$ allows sampling from $\{0, \pm 1\}^N$, but with probability $\rho/2$ for each of $-1, +1$, and $1 - \rho$ for zero. It also requires an error polynomial e and an error vector $\mathbf{e}'$ such that polynomials $e, e_i \leftarrow \mathcal{DG}(\sigma^2)$ which is a secret Gaussian distribution with $\sigma > 0$.
2. **Out:** A ciphertext of the form, $\texttt{ct} = (c_0, \mathbf{c_1}) = (\texttt{pk} \cdot \mathbf{v} + (m + e, \mathbf{e}')) \in \mathcal{R}_q \times \mathcal{R}_q^r$.

$\texttt{MLWE.Dec}_{\texttt{sk}}()$:

1. **In:** Secret key $\texttt{sk}$, a ciphertext $\texttt{ct}$.
2. **Out:** An approximation of the message, $m \approx c_0 + \mathbf{c_1} \cdot \mathbf{s} \pmod{q_0}$.

Next, we discuss the homomorphic sub-routines of the MLWE-based HE scheme. We also discuss the original relinearization procedure and key generation for a comparison with our improved key generation method given in Sec. 3.

Homomorphic subroutines of MLWE-HE

Following are the homomorphic addition and multiplication sub-routines of the MLWE-HE scheme. All operations are done modulo q with some exceptions during the relinearization procedure.

`MLWE.Add()`:

1. **In:**Two ciphertexts $\mathtt{ct}$ and $\mathtt{ct}' \in \mathcal{R}_q \times \mathcal{R}_q^r$.
2. **Out:** The ciphertext sum, which is another ciphertext of the form, $\mathtt{ct}_{\mathtt{add}} = \mathtt{ct} + \mathtt{ct}' = (c_0 + c_0', \mathbf{c_1} + \mathbf{c_1'}) \in \mathcal{R}_q \times \mathcal{R}_q^r$.

`MLWE.Mult()`:

1. **In:**Two ciphertexts $\mathtt{ct}$ and $\mathtt{ct}' \in \mathcal{R}_q \times \mathcal{R}_q^r$.
2. **Out:** The multiplied ciphertext $\mathtt{ct}_{\mathtt{mult}} = (d_0, \mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}) \in \mathcal{R}_q \times \mathcal{R}_q^r \times \mathcal{R}_q^r \times \mathcal{R}_q^{r(r-1)/2}$ of the form,

$$d_0 = c_0 \cdot c_0' \qquad d_{1i} = c_0 \cdot c_{1i}' + c_0' \cdot c_{1i}$$
$$d_{2i} = c_{1i} \cdot c_{1i}' \qquad d_{3_{ij}} = c_{1i} \cdot c_{1j}' + c_{1i}' \cdot c_{1j},\ i < j$$

   where $0 \leq i \leq r-1$.

`MLWE.Relin()`:

1. **In:** Non-relinearized ciphertext $\mathtt{ct}_{\mathtt{mult}}$ and relinearization key $(\mathtt{evk}_1, \mathtt{evk}_2) = \mathtt{MLWE.KeyGen.ksk}(\mathbf{A}_{\mathrm{evk}}; \mathbf{s}, \mathbf{s}' = \mathbf{s} \otimes \mathbf{s})$.
2. **Out:** The relinearized ciphertext $\mathtt{ct}_{\mathtt{relin}} = (d_0', \mathbf{d_1'}) = \big(d_0 + P^{-1}(\mathbf{d_2} \cdot \mathtt{evk}_1[0] + \mathbf{d_3} \cdot \mathtt{evk}_2[0]), \mathbf{d_1} + P^{-1}(\mathbf{d_2} \cdot \mathtt{evk}_1[1] + \mathbf{d_3} \cdot \mathtt{evk}_2[1])\big) \in \mathcal{R}_q \times \mathcal{R}_q^r$

`MLWE.RankRed()`:

1. **In:** Ciphertext $\mathtt{ct} \in \mathcal{R}_q \times \mathcal{R}_q^r$.
2. **Out:** Rank-reduced ciphertext $\mathtt{ct}_{\mathtt{red}} = (c_0', \mathbf{c'}_1) \in \mathcal{R}_q \times \mathcal{R}_q^{r'}$ such that, $\big(c_0' = c_0 + P^{-1} \cdot \mathbf{c}_{\mathrm{rem}} \cdot \mathtt{redk}[0]\big)$ and, $\big(\mathbf{c_1'} = \mathbf{c}_{\mathrm{red}} + P^{-1} \cdot \mathbf{c}_{\mathrm{rem}} \cdot \mathtt{redk}[1]\big)$, where, $\mathbf{c}_{\mathrm{red}} = (c_{1k})_{0 \leq k < r'}$ and $\mathbf{c}_{\mathrm{rem}} = (c_{1j})_{r' \leq j < r}$, are the 'reduced' and the 'removed' components of $\mathbf{c_1}$.

`MLWE.swk()`:

1. **In:** Ciphertext $\mathtt{ct} \in \mathcal{R}_q \times \mathcal{R}_q^r$ involving secret $\mathbf{s}$, key-switching key $\mathtt{swk}$.
2. **Out:** Ciphertext $\mathtt{ct}' = (c_0, 0) + \lfloor P^{-1} \cdot \mathbf{c_1} \cdot \mathtt{swk} \rceil \in \mathcal{R}_q \times \mathcal{R}_q^r$, involving the new secret $\phi(\mathbf{s}) = \mathbf{s}'$.

Residue Number System (RNS) in homomorphic encryption: Efficient implementations of RLWE-based homomorphic encryption schemes in the literature often make use of the Residue Number System to decompose the composite ciphertext modulus into a product of l (co)primes q_i, i.e., $q = \prod_{i=0}^{l-1} q_i$. This transforms modulo q arithmetic into modulo q_i arithmetic. Hence, polynomial arithmetic in $\mathcal{R}_q$ transforms into arithmetic of

residue polynomials in $\mathcal{R}_{q_i}$. This optimization improves efficiency by enabling parallel processing of small-integer arithmetic operations. [MAM+24] also provides an RNS version of their subroutines.

B Correctness of the proposed MLWE-MGHE scheme

In Sec. 4, we demonstrated the correctness of key aggregation. Now, we motivate the correctness of the homomorphic subroutines of MLWE-MGHE using the case where $k = 2$ groups, which can be trivially extended to a general k. Let $\mathtt{pk}^1$ and $\mathtt{pk}^2$ be the joint keys of groups I_1 and I_2. Given messages m_1 and m_2, groups I_1 and I_2 can use their respective joint encryption keys to obtain respective ciphertexts modulo q using `MGHE.Enc()`:

$$\mathtt{ct}^1 = (c_0, \mathbf{c_1})^1 = (\mathbf{v}^1 \cdot \mathtt{pk}^1 + m_1 + e, \mathbf{v}^1 \cdot \mathbf{A} + \mathbf{e})$$
$$\mathtt{ct}^2 = (c_0, \mathbf{c_1})^2 = (\mathbf{v}^2 \cdot \mathtt{pk}^2 + m_2 + e', \mathbf{v}^2 \cdot \mathbf{A} + \mathbf{e}')$$

In `MGHE.Enc()`, we discussed an alternative case when there are multiple groups involved. As mentioned in [KLSW24], a multi-group ciphertext can be considered to contain information about an ordered set of multiple groups and can be decrypted using an ordered set of the joint secret keys of the involved groups. Let, $I = I_1 \bigcup I_2$ be the multi-group set. The multi-group ciphertext, $\mathtt{ct}$ can be written as $(c_0, \mathbf{c_1})^{\cup j} = (c_0^1 + c_0^2, \mathbf{c_1}^1, \mathbf{c_1}^2)$ such that the decryption would satisfy, $c_0^{\cup j} + \mathbf{c_1}^1 \cdot \mathbf{s}^1 + \mathbf{c_1}^2 \cdot \mathbf{s}^2 \approx m_1 + m_2 \pmod q$. Now consider the output of the distributed decryption `DistDec` sub-routine:

$$\begin{aligned} c_0 + \sum_{i \in I} \mu_i &= c_0 + \sum_{i \in I} \Big(\sum_{1 \le j \le k} \mathbf{c_1}^{\cup j} \Big) \cdot \mathbf{s}_i + \sum_{i \in I} e'_i \\ &= c_0 + \big(\mathbf{c_1}^1 \cdot \mathbf{s}^1 + \mathbf{c_1}^2 \cdot \mathbf{s}^2 \big) + \sum_{i \in I} e'_i \\ &\approx m_1 + m_2 + e + e' \end{aligned}$$

where $\sum_{i \in I} e'_i = e'$.

The correctness of homomorphic addition is quite straightforward, so we discuss homomorphic multiplication and relinearization next. Let, $\mathtt{ct}_{\mathtt{mult}} = (d_0, \mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3})^{\cup j}$ be the ciphertext obtained after multiplying two multi-group ciphertexts $\mathtt{ct}$ and $\mathtt{ct}'$ encrypting messages m and m' respectively, and both of which are decryptable under $\mathtt{sk} = (1, \mathbf{s} = (\mathbf{s}^1, \mathbf{s}^2))$. For correctness of multiplication and then relinearization, the following decryption equation modulo q should hold true:

$$\begin{aligned} m \cdot m' &\approx ((c_0 + \mathbf{c_1} \cdot \mathbf{s}) \cdot (c'_0 + \mathbf{c'_1} \cdot \mathbf{s})) \\ &\approx c_0 c'_0 + c_0 \sum_{1 \le j \le k} \mathbf{c'_1}^j \mathbf{s}^j + c'_0 \sum_{1 \le j \le k} \mathbf{c_1}^j \mathbf{s}^j + \sum_{1 \le j, j' \le k} \mathbf{c_1}^j \mathbf{c'_1}^{j'} \mathbf{s}^j \mathbf{s}^{j'} \end{aligned}$$

where, $((c_0 + \mathbf{c_1} \cdot \mathbf{s}) \cdot (c'_0 + \mathbf{c'_1} \cdot \mathbf{s})) = ((c_0^1 + c_0^2 + \mathbf{c_1}^1 \cdot \mathbf{s}^1 + \mathbf{c_1}^2 \cdot \mathbf{s}^2) \cdot (c_0'^1 + c_0'^2 + \mathbf{c'_1}^1 \cdot \mathbf{s}^1 + \mathbf{c'_1}^2 \cdot \mathbf{s}^2))$ for $k = 2$. Also notice that the ciphertext components $(d_0, \mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3})$ indeed follow a similar structure to those in `MLWE.Mult()` since here, $\mathbf{d_2}_j = \mathbf{c_1}^j \cdot \mathbf{c'_1}^j$ and $\mathbf{d_3}_{jj'} = (\mathbf{c_1}^j \cdot \mathbf{c'_1}^{j'} + \mathbf{c_1}^{j'} \cdot \mathbf{c'_1}^j)$ $j \neq j'$. Now, based on the additively homomorphic property of the group relinearization keys as discussed in Sec. 4.2, we can represent the group keys modulo $P \cdot q$ in the following form: $(\mathtt{evk}_1, \mathtt{evk}_2, \mathtt{evk}_3)^1 = \big((-\mathbf{A} \cdot \mathbf{s}^1) + \mathbf{e}_1^1, -\mathbf{A} \cdot \mathbf{u}^1 + \mathbf{e}_2^1 + P \cdot \mathbf{s}^1, -\mathbf{A}' \cdot \mathbf{s}^1 + \mathbf{e}_3^1 - P \cdot \mathbf{u}^1\big)$ and $(\mathtt{evk}_1, \mathtt{evk}_2, \mathtt{evk}_3)^2 = \big((-\mathbf{A} \cdot \mathbf{s}^2) + \mathbf{e}_1^2, -\mathbf{A} \cdot \mathbf{u}^2 + \mathbf{e}_2^2 + P \cdot \mathbf{s}^2, -\mathbf{A}' \cdot \mathbf{s}^2 + \mathbf{e}_3^2 - P \cdot \mathbf{u}^2\big)$. Using the algorithm for `MGHE.Relin()` given in Sec. 4, first let us write the following

decryption equation modulo q for $\mathbf{d_2}$ when $j = 1$:

$$(\mathbf{d_{2_1}} \cdot P^{-1}(\mathtt{evk}_2^1)) \cdot \mathbf{s}^1 + (\mathbf{d_{2_1}} \cdot P^{-1}\mathtt{evk}_1^1) \cdot P^{-1}(\mathtt{evk}_3^1 + \mathbf{A}' \cdot \mathbf{s}^1) \approx \mathbf{d_{2_1}} \cdot (-\mathbf{A} \cdot \mathbf{u}^1 + P \cdot \mathbf{s}^1) \cdot \mathbf{s}^1 + \mathbf{d_{2_1}}(-\mathbf{A} \cdot \mathbf{s}^1)(-\mathbf{A}'\mathbf{s}^1 - \mathbf{u}^1 + \mathbf{A}'\mathbf{s}^1) \approx \mathbf{c_1}^1 \cdot \mathbf{c'_1}^1 \cdot \mathbf{s}^1 \cdot \mathbf{s}^1$$

Similarly, $(\mathbf{d_{2_2}} \cdot P^{-1}(\mathtt{evk}_2^2)) \cdot \mathbf{s}^2 + (\mathbf{d_{2_2}} \cdot P^{-1}\mathtt{evk}_1^2) \cdot P^{-1}(\mathtt{evk}_3^2 + \mathbf{A}' \cdot \mathbf{s}^2) \approx \mathbf{c_1}^2 \cdot \mathbf{c'_1}^2 \cdot \mathbf{s}^2 \cdot \mathbf{s}^2$ for $j = 2$. Finally, for $\mathbf{d_3}$ we have the following:

$$\begin{aligned}(\mathbf{d_{3_{12}}} \cdot P^{-1}(\mathtt{evk}_2^1)) \cdot \mathbf{s}^2 + (\mathbf{d_{3_{12}}} \cdot P^{-1}\mathtt{evk}_1^2) \cdot P^{-1}(\mathtt{evk}_3^1 + \mathbf{A}' \cdot \mathbf{s}^1) &\approx \mathbf{d_{3_{12}}} \cdot (-\mathbf{A} \cdot \mathbf{u}^1 + P \cdot \mathbf{s}^1) \cdot \mathbf{s}^2 \\ &+ \mathbf{d_{3_{12}}}(-\mathbf{A} \cdot \mathbf{s}^2)(-\mathbf{A}'\mathbf{s}^1 - \mathbf{u}^1 + \mathbf{A}'\mathbf{s}^1) \\ &\approx (\mathbf{c_1}^1 \cdot \mathbf{c'_1}^2 + \mathbf{c_1}^2 \cdot \mathbf{c'_1}^1) \cdot \mathbf{s}^1 \cdot \mathbf{s}^2\end{aligned}$$

Thus, the joint relinearization keys correctly relinearize the non-linear components $\mathbf{d_2}$ and $\mathbf{d_3}$ when multiplied by suitable secret key components $\mathbf{s}^j$. Thus, the rest of the linear decryption equation follows a usual ciphertext decryption in MLWE-HE.

The correctness of `MGHE.RankRed()` follows directly from the correctness of `MLWE.RankRed()`. Next, to show how `MGHE.FlexStruct()` works, let us consider that group I_1 consists of three parties such that parties P_1 and P_2 stay whereas P_3 wants to leave the protocol. Then, assuming that the parties interact through confidential channels and using `MGHE.FlexKey()` as well as `MGHE.KeyGen.jk()` gives the flex key as, $(\mathtt{flexk}_1, \mathtt{flexk}_2)^1 = (-\mathbf{A}_{red} \cdot (\mathbf{s}_1 + \mathbf{s}_2) + \mathbf{e}_{red} + P \cdot \mathbf{s}_2, \mathbf{A}_{red})$. If $\mathtt{ct}_{\text{flex}} \leftarrow$ `MLWE.RankRed()` then the decryption equation modulo q is the following:

$$\begin{aligned}c'_0 + \mathbf{c'}_1 \cdot (\mathbf{s}_1 + \mathbf{s}_2) &= \big(c_0 + P^{-1} \cdot \mathbf{c_1} \cdot (-\mathbf{A}_{red} \cdot (\mathbf{s}_1 + \mathbf{s}_2) + \mathbf{e}_{red} + P \cdot \mathbf{s}_1)\big) \\ &+ \big(\mathbf{c_1} + P^{-1} \cdot \mathbf{c_1} \cdot \mathbf{A}_{red}\big) \cdot (\mathbf{s}_1 + \mathbf{s}_2) \\ &\approx (\mathbf{v}^1 \cdot (-\mathbf{A} \cdot (\mathbf{s}_1 + \mathbf{s}_2 + \mathbf{s}_3)) + m_1) \\ &+ P^{-1} \cdot \mathbf{v}^1 \cdot \mathbf{A} \cdot (-\mathbf{A}_{red} \cdot (\mathbf{s}_1 + \mathbf{s}_2) + \mathbf{e}_{red} + P \cdot \mathbf{s}_1) \\ &+ (\mathbf{v}^1 \cdot (-\mathbf{A}) + P^{-1} \cdot \mathbf{v}^1 \cdot \mathbf{A} \cdot (-\mathbf{A}_{red})) \cdot (\mathbf{s}_1 + \mathbf{s}_2) \\ &\approx (\mathbf{v}^1 \cdot (-\mathbf{A} \cdot (\mathbf{s}_1 + \mathbf{s}_2)) + m_1) + (\mathbf{v}^1 \cdot (\mathbf{A} \cdot (\mathbf{s}_1 + \mathbf{s}_2)) \\ &\approx c'_0 + \mathbf{c'_1} \cdot (\mathbf{s}_1 + \mathbf{s}_2)\end{aligned}$$

which is the desired functionality because the group's joint secret key reduces to $\mathbf{s}^j = (\mathbf{s}_1 + \mathbf{s}_2)$ after P_3 leaves the group.

C Noise estimation of the new relinearization procedure in Sec. 3 and MGHE scheme in Sec. 4

We discuss the noise bounds introduced by the modified relinearization technique along the lines of [CKKS17], [CHK+18], [CS16] and [MAM+24]. First, we reiterate some information about the various distributions that the polynomials have been sampled from: let all sampled coefficients be independent and identically distributed, and let σ^2 be the variance of each such coefficient. Then, a polynomial sampled from a uniform distribution $\mathcal{U}$ over $\mathcal{R}_q$ has a variance of $q^2N/12$, a polynomial sampled from the discrete Gaussian distribution $\mathcal{DG}(\sigma^2)$ of mean centered around zero has variance $\sigma^2 N$ and a polynomial sampled from $\mathcal{ZO}(\rho)$ has variance ρN. For a distribution $\mathcal{HWT}(h)$ over signed binary integers $\{0, \pm 1\}$ the variance is just its Hamming weight, h. In the case of a multiplication of two independent random variables sampled from Gaussian distributions with variances σ_1^2 and σ_2^2, the high-probability bound is set to $16\sigma_1\sigma_2$. As a consequence of the *law of large numbers*, the high-probability bound on the (ring) canonical embedding norm is taken to be 6σ.

Lemma 2. *The error introduced by the new relinearization technique is bounded by* $B_{relin} = P^{-1} \cdot r(r+1) \cdot 16\sqrt{\frac{Nq_l^2}{12}}\sigma\sqrt{N} + 16r\sigma\sqrt{hN}$.

Proof. Recall from the expressions of d_0'' and $d_{1t'}''$ in Sec. 3 that the non-linear ciphertext components $\mathbf{d_2}$ and $\mathbf{d_3}$ are multiplied by evaluation keys $\mathtt{evk_1}$ and $\mathtt{evk_2}$ which also results in $\mathbf{d_2}$ and $\mathbf{d_3}$ being multiplied by $\mathbf{e}_{\mathrm{evk}_1}$ and $\mathbf{e}_{\mathrm{evk}_2}$. Also, during decryption, there is an additional error induced by the terms $\mathbf{s} \cdot \mathbf{e}_{\mathrm{evk}_2}$ with respect to each component of $\mathbf{d_2}$ as well as $\mathbf{d_3}$. Consequently, following the error analysis of [MAM+24], the error bound can be described as follows:

$$\|E_{relin}\|_\infty^{can} \leq P^{-1} \cdot r(r+1) \cdot 16\sqrt{\frac{Nq_l^2}{12}}\sigma\sqrt{N} + 16r\sigma\sqrt{hN}$$

□

C.1 Noise analysis of the MGHE scheme

We provide a noise analysis for the case of k groups such that $I = \bigcup_{1 \leq j \leq k} I_j$ with each group consisting of n parties.

Lemma 3. *The error induced during encryption procedure* `MGHE.Enc()` *is bounded by* $B_{MGHE}^{enc} = 16|I|r\sigma(N/\sqrt{2} + \sqrt{hN}) + 6|I|\sigma\sqrt{N}$.

Proof. The group ciphertext encrypted using the joint public key is written as,

$$\begin{aligned}\mathtt{ct} &= (\mathbf{v} \cdot \mathtt{pk}^j + m + e, \mathbf{v} \cdot \mathbf{A} + \mathbf{e}') \\ &= (\mathbf{v} \cdot \sum_{i \in I} \mathtt{pk}_i + m + e, \mathbf{v} \cdot \mathbf{A} + \mathbf{e}')\end{aligned}$$

The ideal decryption of this ciphertext can be written as,

$$\begin{aligned}c_0 + \mathbf{c_1} \cdot \mathbf{s} &= ((\mathtt{pk}^j[0] \cdot \mathbf{v} + m + e) + (\mathtt{pk}^j[1] \cdot \mathbf{v} + \mathbf{e}') \cdot \mathbf{s}^j) \pmod q \\ &= (m + \sum_{i \in I} \mathbf{e_i} \cdot \mathbf{v} + e + \sum_{i \in I} \mathbf{e_i'} \cdot \mathbf{s_i}) \pmod q \\ &= (m + |I| \cdot \sum_{i=0}^{r-1} e_{i_{pk}} \cdot v_i + e + |I| \sum_{i=0}^{r-1} e_i' \cdot s_i) \pmod q\end{aligned}$$

Let, the overall error be $E_{\mathrm{MGHE}}^{enc} = |I| \cdot \sum_{i=0}^{r-1} e_{i_{pk}} \cdot v_i + e + |I| \cdot \sum_{i=0}^{r-1} e_i' \cdot s_i$, then using the upper bound of encryption noise in MLWE-HE [MAM+24], we obtain the following expression:

$$\|E_{\mathrm{MGHE}}^{enc}\|_\infty^{can} \leq 16|I|r\sigma(N/\sqrt{2} + \sqrt{hN}) + 6|I|\sigma\sqrt{N}$$

□

The error bound after a homomorphic addition and multiplication of two group ciphertexts will be a somewhat straightforward adoption of the addition and multiplication error bounds of MLWE-HE [MAM+24]. Next, for the error introduced due to relinearization, recall from Sec. 4 and Sec. B that the non-linear components $\mathbf{d_2}$ and $\mathbf{d_3}$ are multiplied

with the error terms $\mathbf{e_2}^j$ and $\mathbf{e_1}^j \cdot \mathbf{e_3}^j$. In addition, $\mathbf{d_2}$ and $\mathbf{d_3}$ are also multiplied with $\mathbf{s}^j \cdot \mathbf{e_2}^j$ and $\mathbf{u}^j \cdot \mathbf{e_1}^j$. There are a total of $k(k+1)/2$ non-linear terms $\mathbf{d_m}$, $m = 2, 3$ where k is the number of groups, and each such non-linear term has r components. Hence we can give the relinearization error by the following lemma.

Lemma 4. *The error introduced in* `MGHE.Relin()` *by the given relinearization technique is bounded by* $B_{MGHE}^{relin} = P^{-1} \cdot \left(8r|I|(|I|+1)\sqrt{\frac{Nq_l^2}{12}}\sigma\sqrt{N} + 32r|I|\sigma\sqrt{hN}\right)$.