ModHE: Modular Homomorphic Encryption Using Module Lattices

Potentials and Limitations

Anisha Mukherjee[1], Aikata Aikata[1], Ahmet Can Mert[1], Yongwoo Lee[2], Sunmin Kwon[2], Maxim Deryabin[2] and Sujoy Sinha Roy[1]

[1]Graz University of Technology, Graz, Austria,
[2]Samsung Advanced Institute of Technology, Suwon, Republic of Korea

Abstract.
The promising field of homomorphic encryption enables functions to be evaluated on encrypted data and produce results that mimic the same computations done on plaintexts. It, therefore, comes as no surprise that many ventures at constructing homomorphic encryption schemes have come into the limelight in recent years. Most popular are those that rely on the hard lattice problem, called the Ring Learning with Errors problem (RLWE). One major limitation of these homomorphic encryption schemes is that in order to securely increase the maximum multiplicative depth, they need to increase the polynomial-size thereby also increasing the complexity of the design. We aim to bridge this gap by proposing a homomorphic encryption (HE) scheme based on the Module Learning with Errors problem (MLWE), ModHE that allows us to break the big computations into smaller ones. Given the popularity of module lattice-based post-quantum schemes, it is an evidently interesting research endeavor to also formulate module lattice-based homomorphic encryption schemes. While our proposed scheme is general, as a case study, we port the well-known RLWE-based CKKS scheme to the MLWE setting. The module version of the scheme completely stops the polynomial-size blowups when aiming for a greater circuit depth. Additionally, it presents greater opportunities for designing flexible, reusable, and parallelizable hardware architecture. A hardware implementation is provided to support our claims. We also acknowledge that as we try to decrease the complexity of computations, the amount of computations (such as relinearizations) increases. We hope that the potential and limitations of using such a hardware-friendly scheme will spark further research.

Keywords: Homomorphic encryption, module lattice, hardware reusability

1 Introduction

The digital world is quite asymmetric: devices like cell phones are compact but computationally challenged, which is why we often need access to large off-site 'cloud' servers with greater computing power. However, trust is a major issue with such outsourced computations. For example, a hospital wants statistical analysis on the percentage of patients with certain dominating illnesses and hence wants to send its patients' medical records to a research facility. Can the hospital be assured of the complete privacy and protection of its data? One solution is that the hospital signs a privacy contract, and relies on the honesty of the research facility. A better solution would be that the hospital encrypts all its data beforehand and not rely on anybody for its own privacy. However for it to be meaningful the research facility must be able to analyze the encrypted data

without any need for decryption and the hospital would still obtain the same analytical results as the plain data would provide them. This is the very essence of Homomorphic Encryption (HE).

In 1978, Rivest, Adleman, and Dertouzos [RAD78] conjectured that computations could be performed effectively on homomorphically encrypted data without compromising its security. For the next three decades however, the world only knew of partially homomorphic schemes (like RSA [RSA78] and ElGamal [Elg85]) that could support certain fixed types of operations on the ciphertext. The breakthrough came in 2009 when Gentry [Gen09] introduced the first Fully Homomorphic Encryption (FHE) scheme that could perform arbitrary operations on homomorphically encrypted data. He showed that a somewhat homomorphic scheme could be made fully homomorphic with a process he named 'bootstrapping' through which ciphertexts can be 'refreshed', a way to homomorphically evaluate the decryption circuit. His construction, although quite complex, sparked active research and led to the introduction of other simpler homomorphic schemes. Brakerski *et al.* [BV11a] proposed an FHE scheme whose security assumption was based on the classical hardness of solving standard lattice problems in the worst-case, which is more well-known as the Learning with Errors (LWE) problem. The dimension of the lattice determines the scheme's extent of security. Then, in 2011, [BV11b] and in 2012 [FV12] (BFV) ported Brakerski's scheme from standard LWE setting to LWE over algebraic rings with the Ring Learning with Errors problem (RLWE). The BGV scheme[BGV11] is another popular RLWE-HE candidate. The TFHE scheme [CGGI20] also uses LWE and the RLWE over a torus but adopts a different bootstrapping procedure. CKKS/HEAAN [CKKS17] and its residue number system (RNS) variant, [CHK+18b], are recent RLWE-based schemes that incorporate homomorphic computation of data over the real and complex fields. In spite of a series of significant advancements in the theoretical aspects of homomorphic encryption, present-day homomorphic encryption schemes introduce a huge computation overhead ranging from 10^4 to 10^5 compared to plaintext calculations. As a consequence, software implementations of homomorphic encryption in general-purpose computers are far from usable in the privacy-preserving outsourcing of computation.

To speed up homomorphic encryption significantly, multiple attempts to develop customized hardware accelerators have surfaced in the last few years. These works range from real acceleration works, for example, the GPU and FPGA-based accelerators [JKA+21, BHM+20, MAK+23, AdCY+23, RLPD20, RJV+18, TRG+20, TRV20], to futuristic ASIC designs [FSK+21, GVBP+22, KKK+22, SFK+22, KLK+22]. It becomes clear from their impressive speedup records, that ASIC or FPGA-based hardware accelerators will be fundamental to making homomorphic encryption usable for real-life privacy-preserving computation. Studying the above-mentioned hardware acceleration works, we see that they start with the mathematical representation of a given homomorphic encryption scheme and make hardware-based building blocks to speed up the mathematical steps of the scheme. Contrasting the typical hardware accelerator development cycle, our approach takes the opposite direction and tries to answer the question *Could homomorphic encryption schemes be designed in a way that they become hardware-friendly by construction?* Our approach takes inspiration from the ongoing NIST post-quantum cryptography standardization project. The post-quantum schemes that were designed based on the MLWE problem, offer superior performance (in general) and flexibility than their RLWE-based counterparts.

The following two paragraphs present the motivation behind designing a hardware-friendly homomorphic encryption scheme. Well-known homomorphic schemes like BGV, BFV, or CKKS make use of RLWE for all scheme arithmetic. This means that applications requiring complex operations would demand working with a bigger ciphertext modulus and a corresponding polynomial degree. As an example, while polynomials of degree 2^{12} and a 180-bit ciphertext modulus would suffice for homomorphic evaluation of a simple quadratic function, an application performing logistic regression would require almost 4 times larger

ring and ciphertext modulus sizes. The security of the scheme goes hand-in-hand with the size of the ciphertext modulus and the degree of the ring: a need for higher multiplicative depth equates to choosing a bigger ciphertext modulus but to keep the level of security intact a proportional increase in the ring size is equally imperative. These inevitable variations in the parameter sets make optimizing the performance of a hardware accelerator circuit considerably challenging when different applications have to be supported.

In post-quantum public-key cryptography, module lattice-based schemes such as Saber [DKR+21], Kyber [SAB+21], Dilithium [BDK+21], etc., have been successful in mitigating similar problems with varying parameter sets. With a fixed polynomial degree, these schemes only have to switch between dimensions of the vector or the matrix of such polynomials to incorporate changing security levels. Taking inspiration from that, this paper gives the sketch of a module lattice-based homomorphic encryption scheme and discusses its advantages and limitations compared to the state-of-the-art ideal lattice-based schemes.

1.1 Our contributions

In this work, we present a leveled MLWE-based homomorphic encryption scheme and as an instance, propose a module variant of CKKS [CKKS17] which we call, `ModCKKS`. We analyze, in detail, varied perspectives that could favor the use of this scheme as well as those that pose challenges. Then we take a step closer to our main goal of achieving hardware reusability and flexibility within dynamic notions of cryptographic security.

Algorithmically, we try to retain the properties of the ring variant while adapting them for modules, thereby realizing our motivation without having to make any drastic changes to the heuristics of the original scheme. We investigate the consequences of choosing a fixed base ring and then building upon the rank of the associated module depending on the desired parameters for circuit depth and security.

We provide detailed algorithmic descriptions for the readers to understand the design decisions and challenges. Through ModHE we reduce the computation complexity, however, the amount of computations and storage required increases. We acknowledge this as a trade-off for offering hardware reusability because the availability of physical resources cannot be at par with the changing security caliber. With every such change, having to replace a machine's hardware or buying new ones would not be cheap. It is therefore highly significant to note that our work addresses this issue and is able to present a way of mitigating the expense.

The design methodology utilizes the multi-dimensional parallel processing opportunities offered by ModHE and presents them in the context of RNS-CKKS. ModHE allows us to continue processing small polynomials even when we require a higher depth. This is because the number of polynomials packed in a module increases, but their size does not increase. We show how these different components can be utilized mostly in parallel and design the hardware accordingly. To the best of our knowledge, no literature exists that discusses the implications of an MLWE-based homomorphic scheme in hardware. To support our motivation, we provide area and performance results for a hardware accelerator architecture targeting the Xilinx FPGA Alveo U280 card.

A proof-of-concept Sage implementation[1] of the module-lattice-based leveled CKKS homomorphic encryption scheme is provided.

Organisation: In Section 2, we provide mathematical details that will be useful to build concepts in the rest of the paper. We also give a brief description of the important sub-routines of the RNS-CKKS scheme. In Section 3, we present algorithmic details of `ModCKKS`, the error bounds of important sub-routines, and also discuss the RNS represen-

[1] https://github.com/anonymous-sub-ches/Mod-CKKS

tation. In section 4, we give the potentials and limitations related to a module-based HE construction. The hardware design is proposed in Section 5 and Section 6 provides food for thought toward future possibilities and modifications of ModHE constructions. Section 7 concludes the paper.

2 Mathematical background

2.1 Notation

Let $N \in \mathbb{N}$ be a power of two. For a number field $\mathbb{Q}[X]/(\phi_{2N}(X))$ we denote $\mathcal{R} = \mathbb{Z}[X]/(\phi_{2N}(X))$ as its ring of integers consisting of polynomials modulo the $2N$-th cyclotomic polynomial, $\phi_{2N}(X) = X^N + 1$. 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 denote the module of rank r over $\mathcal{R}_q$ as $\mathcal{R}_q^r$.

Unless stated explicitly, we will use q to denote a ciphertext modulus. When discussing the RNS version we will shift to a notation Q for the large ciphertext modulus which is a product of the small primes q_i.

Elements named in usual lowercase letters will denote single polynomials unless otherwise explicitly specified to be integers; bold lowercase letters will represent a vector of polynomials (except in sec 2.2 where we use this notation for a vector of integers) and bold uppercase letters will denote multi-dimensional matrices. The Number Theoretic Transform (NTT) of a polynomial a is represented by $\tilde{a}$.

2.2 The Learning with Errors problem and its algebraic variants

The Learning with Errors, LWE problem was introduced by Regev [Reg05] in 2005. The LWE problem is parameterized by two integers $n \geq 1$ and $q \geq 2$, and an error distribution χ_{err}. It has two variants: the search and the decision variant. The decision variant is usually preferred for defining cryptographic primitives.

Decision Learning with Errors, LWE: The decision $\text{LWE}_{n,q,\chi_{err}}$ problem states that for a secret vector, $\mathbf{s}$ in some distribution $\chi_{key}(\mathbb{Z}_q^n)$, random vector $\mathbf{a}$ sampled from a uniform distribution $\mathcal{U}(\mathbb{Z}_q^n)$ and e sampled from χ_{err}, it is infeasible to distinguish uniformly random samples $(\mathbf{a}, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ from samples of the form $(\mathbf{a}, b = \langle \mathbf{a}^T s \rangle + e \mod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.

The Ring Learning With Errors, RLWE was introduced by Lyubashevsky *et al.* [LPR10] for speeding up cryptographic constructions based on LWE.
Decision Ring Learning with Errors, RLWE: The decision $\text{RLWE}_{N,q,\chi_{err}}$ problem states that for a secret, s in some distribution $\chi_{key}(\mathcal{R}_q)$, a sampled from a uniform distribution $\mathcal{U}(\mathcal{R}_q)$ and e sampled from χ_{err}, it is infeasible to distinguish uniformly random samples $(a, b) \in \mathcal{R}_q \times \mathcal{R}_q$ from samples of the form $(a, b = \langle a^T s \rangle + e \mod q) \in \mathcal{R}_q \times \mathcal{R}_q$.

While the LWE problem is known to be as hard as worst-case problems on Euclidean lattices, RLWE is considered as hard as the problems are restricted over special classes of ideal lattices. The Module Learning with Errors was introduced as a bridge between the general LWE and RLWE and was discussed in detail by Langlois and Stehlé [LS15].

Decision Module Learning with Errors, MLWE: The decision $\text{MLWE}_{N,r,q,\chi_{err}}$ problem states that for a secret, $\mathbf{s}$ in some distribution $\chi_{key}(\mathcal{R}_q^r)$, $\mathbf{a}$ sampled from

a uniform distribution $\mathcal{U}(\mathbb{R}_q^r)$ and e sampled from χ_{err} given uniformly random samples $(\mathbf{a}, b) \in \mathcal{R}_q^r \times \mathcal{R}_q$, it is hard to distinguish them from LWE samples of the form, $(\mathbf{a}, b = \langle \mathbf{a}^T \mathbf{s} \rangle + e \mod q) \in \mathcal{R}_q^r \times \mathcal{R}_q$.

For a fixed number of samples, the MLWE problem can also be viewed in terms of linear algebra by considering a matrix $\mathbf{A}$ whose rows consist of all the sampled $\mathbf{a_i}$'s. Interestingly, for a commutative ring $\mathcal{R}$, an $\mathcal{R}$-algebra is an $\mathcal{R}$-module which is also a ring. The set, $M_n(\mathcal{R})$ of all square $n \times n$ matrices over a commutative ring $\mathcal{R}$ with entries in $\mathcal{R}$ is an $\mathcal{R}$-algebra. While a multiplication between two module elements is not intrinsic, a multiplication between two elements of $M_n(\mathcal{R})$ will still make sense when perceived as elements in a ring.

2.3 CKKS and RNS-CKKS

In a typical homomorphic encryption protocol, a client sends encrypted data to a cloud server that performs computations on it and sends the encrypted results back to the client. The client decrypts the received results locally to again obtain meaningful plaintext results. The client has his own secret key to be able to en(de)crypt messages.

We give an intuitive description of an RLWE-based homomorphic encryption scheme in the following part. Let a client's secret-key be $\mathtt{sk} = (1, s) \in \mathcal{R}_q^2$ and the corresponding public-key be $\mathtt{pk} = (b, a) \in \mathcal{R}_q^2$. Client encrypts a message m using $\mathtt{pk}$ and obtains the ciphertext $\mathtt{ct} \leftarrow (c_0 = v \cdot b + e_0 + m, c_1 = v \cdot a + e_1) \in \mathcal{R}_q^2$ where e_i is a Gaussian distributed error-polynomial and v is a uniformly random polynomial. Assume that a cloud contains two ciphertexts $\mathtt{ct} = (c_0, c_1)$ and $\mathtt{ct'} = (c_0', c_1') \in \mathcal{R}_q^2$ of the client with respect to messages m and m' respectively. The cloud can compute a valid encryption of $m + m'$ simply by adding the two ciphertexts as $\mathtt{ct}_{\text{add}} \leftarrow (c_0 + c_0', c_1 + c_1') \in \mathcal{R}_q^2$. Computing encryption of $m \cdot m'$ is relatively complex and often differs based on the scheme. The basic idea is that the multiplication of two ciphertexts will result in their respective components being multiplied with each other, like, $\mathtt{ct}_{\text{mult}} = (c_0 \cdot c_0', c_0 \cdot c_1' + c_1 \cdot c_0', c_1 \cdot c_1') \in \mathcal{R}_q^3$. This intermediate result has three polynomial components and can be decrypted using $(1, s, s^2)$ but not using $\mathtt{sk} = (1, s)$. To again allow for decryption to happen using $\mathtt{sk}$, a 'Key-Switching' operation is used to transform the three-component ciphertext $\mathtt{ct}_{\text{mult}}$ back into the usual two-component ciphertext $\mathtt{ct}_{\text{relin}}$ decryptable under $(1, s)$. In this context, key-switching is called 'relinearisation' as it produces a linear ciphertext.

Most ideal lattice-based homomorphic encryption schemes such as BGV [BGV11], BFV [FV12], CKKS [CKKS17] and RNS-CKKS [CHK+18b] share a similar underlying protocol structure. We give a brief description of RNS-CKKS [CHK+18b] as we will build our MLWE construction upon it.

Residue Number System (RNS)
The Residue Number System makes use of the Chinese Remainder Theorem to represent an integer as a vector of its *residues* modulo a basis of pairwise co-prime integers. The same can also be applied to polynomials in rings. If a is a polynomial in the cyclotomic ring $\mathcal{R}_Q$ and $\mathcal{C} = \{q_0, \cdots, q_{k-1}\}$ is a basis such that $Q = \prod_{i=0}^{k-1} q_i$ then, there is a ring isomorphism from $a \in \mathcal{R}_Q$ to its representation $(a^{(0)}, a^{(1)}, \cdots, a^{(k-1)}) \in \prod_{i=0}^{k-1} \mathcal{R}_{q_i}$ being applied coefficient-wise. In compact form, the RNS representation of a can be denoted by $[a]_{\mathcal{C}}$. RNS proves useful for implementing HE schemes on hardware and software platforms as it enables the parallelization of computations. Each polynomial involved in the scheme's routines has shares with respect to each small moduli in the basis. In other words, polynomial arithmetic is done among smaller decomposed polynomials instead of big polynomials modulo a big modulus. HE schemes using RNS choose prime moduli q_i's so that they can utilise the advantage of Number Theoretic Transform (NTT) for fast multiplications.

RNS-CKKS
The original CKKS scheme could not support a double-CRT representation since the rounding operation involved in the approximate arithmetic dictated the use of a power-of-two ciphertext modulus. An RNS variant of CKKS appeared in [CHK+18b] by constructing algorithms that could incorporate double-CRT without compromising on the benefits of CKKS. We give a brief description of the important sub-routines of RNS-CKKS. For better legibility, we will use only CKKS to denote them. Here, Q_l is a product of $q_i's$ up to an arbitrary level l.

- $\texttt{CKKS.KeyGen}(\texttt{s})$: The secret s is sampled from a secret key distribution χ_{key} such that the secret key is $\texttt{sk} = (1, s)$. The public key is, $\texttt{pk} = (-a \cdot s + e, a) \in \mathcal{R}^2_{Q_l}$, where the polynomial a is sampled from a uniform distribution $\mathcal{U}$ and the error e from an error distribution χ_{err}.

- $\texttt{CKKS.Enc}(\texttt{m})$: It encrypts a message m into a ciphertext $\texttt{ct} = (c_0, c_1) = v \cdot \texttt{pk} + (m + e_0, e_1) \in \mathcal{R}^2_{Q_l}$, with v sampled from a uniform distribution χ_{enc}.

- $\texttt{CKKS.Dec}(\texttt{ct})$: It returns an approximation of the message m, $\langle \texttt{ct}, \texttt{sk} \rangle$.

- $\texttt{CKKS.Add}(\texttt{ct}, \texttt{ct}')$: It adds the polynomial components of the two ciphertexts $\texttt{ct} = (c_0, c_1) \in \mathcal{R}^2_{Q_l}$ and $\texttt{ct}' = (c'_0, c'_1) \in \mathcal{R}^2_{Q_l}$, and computes $\texttt{ct}_{\text{add}} = (d_0, d_1)$ where $d_0 = c_0 + c'_0 \in \mathcal{R}^2_{Q_l}$ and $d_1 = c_1 + c'_1 \in \mathcal{R}^2_{Q_l}$.

- $\texttt{CKKS.Mult}(\texttt{ct}, \texttt{ct}')$: It multiplies two input ciphertexts $\texttt{ct} = (c_0, c_1) \in \mathcal{R}^2_{Q_l}$ and $\texttt{ct}' = (c'_0, c'_1) \in \mathcal{R}^2_{Q_l}$, and computes $d_0 = c_0 \cdot c'_0 \in \mathcal{R}_{Q_l}$, $d_1 = c_0 \cdot c'_1 + c_1 \cdot c'_0 \in \mathcal{R}_{Q_l}$, and $d_2 = c_1 \cdot c'_1 \in \mathcal{R}_{Q_l}$. The output is the non-linear ciphertext $\texttt{d} = (d_0, d_1, d_2) \in \mathcal{R}^3_{Q_l}$.

- $\texttt{CKKS.Relin}(\texttt{d}, \texttt{evk})$: It relinearises the result of $\texttt{CKKS.Mult}$ and produces a ciphertext with two polynomial components so that it is decryptable under the secret key. Let $d'_2[i] = \left[d_2\right]_{q_i}$ for $0 \leq i < l$. Now one needs to compute $\texttt{ct}'' = (c''_0, c''_1)$ where $c''_0 = \sum_{i=0}^{l-1} d'_2[i] \cdot \texttt{evk}_0[i] \in \mathcal{R}_{pQ_l}$ and $c''_1 = \sum_{i=0}^{l-1} d'_2[i] \cdot \texttt{evk}_1[i] \in \mathcal{R}_{pQ_l}$. The final output is the relinearized ciphertext $\texttt{ct}_{\text{relin}} = (d_0, d_1) + (\texttt{CKKS.ModDown}(c''_0), \texttt{CKKS.ModDown}(c''_1)) \pmod{Q_l}$. The $\texttt{CKKS.ModDown}$ operation is used to reduce the coefficient modulus from pQ_l to Q_l.

- $\texttt{CKKS.Rescale}(\texttt{ct})$: It takes a ciphertext $\boldsymbol{c} \in \mathcal{R}_{Q_l}$ with level l and produces a ciphertext element with level $l-1$. Let $\texttt{ct}' \in \mathcal{R}_{Q_{l-1}}$ such that $\texttt{ct}'[i] = \texttt{ct}[l] \pmod{q_i}$ for $0 \leq i \leq l-1$. Then, compute $\texttt{ct}'' = \texttt{ct} - \texttt{ct}' \in \mathcal{R}_{Q_{l-1}}$. Finally, output the rescaled ciphertext element $\texttt{ct}'' = q_l^{-1} \cdot \texttt{ct}'' \in \mathcal{R}_{Q_{l-1}}$.

- $\texttt{CKKS.Rotate}(\texttt{ct}, \texttt{rtk})$: The slot rotation operation takes a ciphertext $\texttt{ct} = (c_0, c_1) \in \mathcal{R}^2_{Q_l}$ and rotation key $\texttt{rtk}$ and performs an automorphism and a key-switch of the ciphertext polynomial coefficients. The ciphertext is then encrypted under a rotated secret key.

The ciphertexts and keys are stored in NTT format to make the sub-routines described above more efficient (polynomial multiplication becomes $\mathrm{O}(N \log(N))$). To provide, an algorithmic overview of this we also describe the Algorithm 1 $\texttt{CKKS.Add}$, Algorithm 2 $\texttt{CKKS.Mult}$, Algorithm 3 $\texttt{CKKS.ModDown}$, and Algorithm 4 $\texttt{CKKS.Relin}$.

Algorithm 1 `CKKS.Add` [CHK+18b]

In: $\mathtt{c} = (\tilde{c}_0, \tilde{c}_1)$, $\mathtt{c}' = (\tilde{c}'_0, \tilde{c}'_1) \in R^2_{Q_l}$
Out: $\mathtt{d} = (\tilde{d}_0, \tilde{d}_1) \in R^2_{Q_l}$

1: $\tilde{d}_0 \leftarrow \tilde{c}_0 + \tilde{c}'_0$
2: $\tilde{d}_1 \leftarrow \tilde{c}_1 + \tilde{c}'_1$

Algorithm 2 `CKKS.Mult` [CHK+18b]

In: $\mathtt{ct} = (\tilde{c}_0, \tilde{c}_1)$, $\mathtt{ct}' = (\tilde{c}'_0, \tilde{c}'_1) \in R^2_{Q_l}$
Out: $\mathtt{d} = (\tilde{d}_0, \tilde{d}_1, \tilde{d}_2) \in R^3_{Q_l}$

1: $\tilde{d}_0 \leftarrow \tilde{c}_0 \star \tilde{c}'_0$, $\tilde{d}_2 \leftarrow \tilde{c}_1 \star \tilde{c}'_1$
2: $\tilde{d}_1 \leftarrow \tilde{c}_0 \star \tilde{c}'_1 + \tilde{c}_1 \star \tilde{c}'_0$

Algorithm 3 `CKKS.ModDown` [CHK+18b]

In: $\tilde{d} \in R_{pQ_l}$
Out: $\tilde{d}' \in R_{Q_l}$

1: $t \leftarrow \mathtt{INTT}(\tilde{d}[l])$
2: **for** $i = 0$ to $l - 1$ **do**
3: $\quad \tilde{t} \leftarrow \mathtt{NTT}(\left[t\right]_{q_i})$ ▷ in $\mathbb{Z}_{q_i}$
4: $\quad \tilde{d}'[i] \leftarrow \left[p^{-1} \cdot (\tilde{d}[i] - \tilde{t})\right]_{q_i}$
5: **end for**

Algorithm 4 `CKKS.Relin` [CHK+18b]

In: $\mathtt{d} = (\tilde{d}_0, \tilde{d}_1, \tilde{d}_2) \in R^3_{Q_l}$, $\mathtt{Evk}_0 \in R^l_{pQ_l}, \mathtt{Evk}_1 \in R^l_{pQ_l}$
Out: $\mathtt{d}' = (\tilde{d}'_0, \tilde{d}'_1) \in R^2_{Q_l}$

1: **for** $j = 0$ to $l - 1$ **do**
2: $\quad d_2[j] \leftarrow \mathtt{INTT}(\tilde{d}_2[j])$ ▷ in $\mathbb{Z}_{q_j}$
3: **end for**
4: **for** $j = 0$ to l **do** ▷ Here q_l is used to represent special prime p
5: $\quad (\tilde{c}''_0[j], \tilde{c}''_1[j]) \leftarrow 0$
6: $\quad$ **for** $i = 0$ to $l - 1$ **do**
7: $\quad\quad \tilde{r} \leftarrow \mathtt{NTT}(\left[d_2[i]\right]_{q_j})$ ▷ in $\mathbb{Z}_{q_j}$
8: $\quad\quad \tilde{c}''_0[j] \leftarrow \left[\tilde{c}''_0[j] + \mathtt{Evk}_0[i][j] \star \tilde{r}\right]_{q_j}$, $\tilde{c}''_1[j] \leftarrow \left[\tilde{c}''_1[j] + \mathtt{Evk}_1[i][j] \star \tilde{r}\right]_{q_j}$
9: $\quad$ **end for**
10: **end for**
11: $\tilde{d}'_0 \leftarrow \tilde{d}_0 + \mathtt{CKKS.ModDown}(\tilde{c}''_0)$, $\tilde{d}'_1 \leftarrow \tilde{d}_1 + \mathtt{CKKS.ModDown}(\tilde{c}''_1)$

2.4 Encoding using complex embeddings

Since we work with an underlying polynomial ring structure, we would also expect messages to be in the form of polynomials. But more often than not data comes in the form of vectors of plaintexts. [CKKS17] discusses a method to 'pack' multiple messages in one ciphertext.

Let $\mathcal{K}$ be the cyclotomic field such that $\mathcal{R} \subseteq \mathcal{K}$. The complex canonical embeddings are the ring homomorphisms $\tau_j : \mathcal{K} \to \mathbb{C}$ such that for a $2N$-th root of unity ξ, $\tau_j : \xi \mapsto \xi_j$ where $j \in \mathbb{Z}^*_{2N}$. The canonical embedding $\tau : \mathcal{K} \to \mathbb{C}^N$ can be defined as, $\tau(\mathbf{z}) = (\tau_j(\mathbf{z}))_{j \in \mathbb{Z}^*_{2N}}$, $\mathbf{z} = (z_j)_{j \in \mathbb{Z}^*_{2N}}$ where addition and multiplication in $\mathbb{C}^N$ are component-wise. Since $\tau_{-j} = \bar{\tau}_j$, so there can exist a natural projection $\pi : \mathbb{H} \to \mathbb{C}^{N/2}$ where, $\mathbb{H} = \{(z_j)_{j \in \mathbb{Z}^*_{2N}} : z_{-j} = \bar{z_j}, j \in \mathbb{Z}^*_{2N}\}$. The encoding process thus involves transforming a vector $\mathbf{z}$ under the inverse of π and then using the inverse of the canonical embedding with some rounding operation to finally obtain a polynomial in $\mathcal{R}$. Hence, a plaintext is the polynomial $m(X) = \tau^{-1} \circ \pi^{-1}(\mathbf{z}) \in \mathcal{R}$. The decoding procedure is simply, $\mathbf{z} = \pi \circ \tau(m(X)) \in \mathbb{C}^{N/2}$.

3 Proposed MLWE-HE scheme: ModCKKS

Let λ be a security parameter that governs the dimension of the underlying ring N, the rank of the module r, the circuit depth L and the ciphertext modulus $q > 0$ and an integer $P > 0$. For realizing a module lattice-based homomorphic encryption scheme, we port the ring-based CKKS and RNS-CKKS to the MLWE setting. We name it ModCKKS. Let, $\mathbf{A} \in \mathcal{R}_q^{r\times r}$ consist of polynomials a sampled uniformly from $\mathcal{U}(\mathcal{R}_q)$, $\mathbf{s}$ be the secret whose components are chosen randomly from r copies of the set $\mathcal{HWT}(h)$ of signed binary polynomials $\{0, \pm 1\}^N$ with the Hamming weight h and $\mathbf{e}$ be the error each of whose polynomial coefficients is sampled independently from a discrete Gaussian distribution $\mathcal{DG}(\sigma^2)$, $\sigma > 0$. We also consider another distribution $\mathcal{ZO}(\rho)$ with $0 \leq \rho \leq 1$ which samples each entry in the vector also from $\{0, \pm 1\}^N$, but with probability $\rho/2$ for each of $-1, +1$, and $1 - \rho$ for zero. Let any message $m \in \mathcal{R}$ be encoded under some encoding scheme, like the one described in section 2.

- $\texttt{ModCKKS.KeyGen}(1^\lambda)$: Generate a secret key $\texttt{sk} = (1, \mathbf{s})$ with $s_i \leftarrow \mathcal{HWT}(h)$. Sample $a_i \leftarrow \mathcal{U}(\mathcal{R}_q)$ for the random matrix $\mathbf{A}$ and $e_i \leftarrow \mathcal{DG}(\sigma^2)$ for the error vector $\mathbf{e}$. Generate public key $\texttt{pk} = (\mathbf{b}, \mathbf{A}) = (-\mathbf{A} \cdot \mathbf{s} + \mathbf{e}, \mathbf{A}) \pmod q \in \mathcal{R}_q^r \times \mathcal{R}_q^{r\times r}$. Generate the switching key $\texttt{swk} = (\mathbf{b}_{swk}, \mathbf{A}_{swk}) = (-\mathbf{A}\cdot\mathbf{s}+\mathbf{e}+P\cdot\mathbf{s}', \mathbf{A}) \pmod q \in \mathcal{R}_q^r \times \mathcal{R}_q^{r\times r}$ and similarly, the evaluation key $\texttt{evk} = (\mathbf{b}_{evk}, \mathbf{A}_{evk}) = (-\mathbf{A} \cdot \mathbf{s} + \mathbf{e} + P \cdot \mathbf{s}'', \mathbf{A}) \pmod q \in \mathcal{R}_q^r \times \mathcal{R}_q^{r\times r}$, where the explicit forms of $\mathbf{s}''$ will be discussed later.
- $\texttt{ModCKKS.Enc}(\texttt{m})$: We obtain a ciphertext encrypting a message m, $\texttt{ct} = (c_0, \mathbf{c_1}) = (\texttt{pk} \cdot \mathbf{v} + (m + e, \mathbf{e}')) \pmod q \in \mathcal{R}_q \times \mathcal{R}_q^r$, $\mathbf{v} \in \mathcal{R}_q^r$ where the polynomials v_i of vector $\mathbf{v}$ are sampled as $v_i \to \mathcal{ZO}(0.5)$.
- $\texttt{ModCKKS.Dec}(\texttt{ct})$: We obtain an approximation of the message after decryption under $\texttt{sk}$, $(c_o + \mathbf{c_1} \cdot \mathbf{s}) \pmod q \approx m$.
- $\texttt{ModCKKS.Add}(\texttt{ct}, \texttt{ct}')$: Given two ciphertexts $\texttt{ct}$ and $\texttt{ct}'$, their sum is a sum of their corresponding components.

$$\texttt{ct}_{\texttt{add}} = \texttt{ct} + \texttt{ct}' = (c_0 + c_0', \mathbf{c_1} + \mathbf{c_1'}) \pmod q \in \mathcal{R}_q \times \mathcal{R}_q^r.$$

- $\texttt{ModCKKS.Mult}(\texttt{ct}, \texttt{ct}')$: For the multiplication operation $'*'$ between two ciphertexts $\texttt{ct}$ and $\texttt{ct}'$ to be homomorphic we would like to have,

$$\begin{aligned} m \cdot m' &\approx \texttt{ModCKKS.Dec}((c_0, \mathbf{c_1}) * (c_0', \mathbf{c_1'})) \\ &\approx \texttt{ModCKKS.Dec}(c_0, \mathbf{c_1}) * \texttt{ModCKKS.Dec}(c_0', \mathbf{c_1'}) \\ &\approx ((c_0 + \mathbf{c_1} \cdot \mathbf{s}) \cdot (c_0' + \mathbf{c_1'} \cdot \mathbf{s})) \pmod q \\ &\approx c_0 c_0' + c_0 \sum_{i=0}^{r-1} c_{1i}' s_i + c_0' \sum_{i=0}^{r-1} c_{1i} s_i + \sum_{i=0}^{r-1} \sum_{j=0}^{r-1} c_{1i} c_{1j}' s_i s_j. \end{aligned}$$

  We write the resultant ciphertext of the multiplication as, $\texttt{ct}_{\texttt{mult}} = \texttt{d} = (d_0, \mathbf{d_1}, \mathbf{d_2}, \mathbf{d_3}) \in \mathcal{R}_q \times \mathcal{R}_q^r \times \mathcal{R}_q^r \times \mathcal{R}_q^{r(r-1)/2}$ where:

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

  with all the above arithmetic done modulo q.

  The ciphertext thus obtained is 'extended' in the sense that it now has more components than a usual ciphertext. For example, in the case of a module of rank 2, the extended decryption equation would look like,

$$m \cdot m' \approx (d_0 + \mathbf{d_1} \cdot \mathbf{s} + \mathbf{d_2} \cdot \mathbf{s}^2 + d_3 \cdot s_0 s_1) \pmod q. \tag{1}$$

So just after one multiplication, there is a growth in the number of ciphertext components from 2 to 4. These changes are not desirable in the design and we would want to find a way to get $\texttt{ct}_{\texttt{mult}}$ to mimic a usual ciphertext so that it can be decrypted by the usual secret key components, $\texttt{sk}$. Like, for eqn. 1 it would mean that we would want to look for expressions d'_0 and $\mathbf{d'_1}$ such that it can be rewritten approximately as, $d'_0 + \mathbf{d'_1} \cdot \mathbf{s} \approx d_0 + \mathbf{d_1} \cdot \mathbf{s} + \mathbf{d_2} \cdot \mathbf{s}^2 + d_3 \cdot s_0 s_1$.

The concepts of relinearization and rescaling come in as a method for converting a degree 2 ciphertext again into a degree 1 ciphertext that can be decrypted under the secret key $\texttt{sk}$.

- $\texttt{ModCKKS.Relin}(\texttt{ct}_{\texttt{mult}})$: Let $P > q$ be an integer that will be used to control the increase in error after multiplication.
  Continuing with our example of module rank 2, in order to do away with the non-linear terms (in $\mathbf{s}$) from eqn. (1) with respect to $\mathbf{d_2}$ and d_3, we take help of the following evaluation keys:

$$\begin{aligned}\texttt{evk}_{\texttt{d}_2} &= (-\mathbf{A}_{d_2} \cdot \mathbf{s} + \mathbf{e} + P \cdot \mathbf{s}^2, \mathbf{A}_{d_2}) \pmod{P \cdot q} \\ \texttt{evk}_{\texttt{d}_3} &= (-\mathbf{A}_{d_3}[0] \cdot \mathbf{s} + e + P \cdot s_0 s_1, \mathbf{A}_{d_3}[0]) \pmod{P \cdot q}\end{aligned}$$

  where for a rank of two $\mathbf{A}_{d_i}$ is sampled uniformly from $\mathcal{R}^{2\times 2}_{P\cdot q}$ and $\mathbf{A}_{d_3}[0]$ is a vector of uniformly random polynomials.

  We can now define the relinearization components as,

$$\begin{aligned}d'_0 &= (d_0 + P^{-1} \cdot \mathbf{d_2} \cdot \texttt{evk}_{\texttt{d}_2}[0] + P^{-1} \cdot d_3 \cdot \texttt{evk}_{\texttt{d}_3}[0]) \pmod{q} \\ \mathbf{d'_1} &= (\mathbf{d_1} + P^{-1} \cdot \mathbf{d_2} \cdot \texttt{evk}_{\texttt{d}_2}[1] + P^{-1} \cdot d_3 \cdot \texttt{evk}_{\texttt{d}_3}[1]) \pmod{q}\end{aligned}$$

  where components of $\mathbf{d_2}$ and d_3 have been assumed to have been lifted (*modulus switch*) from residue ring modulo q to residue ring modulo $P{\cdot}q$ as is done in [CKKS17]. Hence, the central idea here is that by using evaluation keys we adjust the terms that correspond to secret key components of the form $\sum_{i=0}^{r-1}\sum_{j=0}^{r-1} s_i s_j$, that is, relinearization keys are encryptions of these secret key components.

  For a general rank r module, these keys have the form, $\texttt{evk} = (-\mathbf{A} \cdot \mathbf{s} + \mathbf{e} + P \cdot \sum_i \sum_{j\geq i} s_i s_j, \mathbf{A}) \pmod{P \cdot q}$, where, $0 \leq i, j < r$ and, $\mathbf{A} \in \mathcal{R}^{r\times r}_{P\cdot q}$. So, $\texttt{ct}_{\texttt{relin}} = (d'_0, \mathbf{d'_1}) = \big(d_0 + P^{-1}(\mathbf{d_2}\cdot\texttt{evk}_{d_2}[0] + \mathbf{d_3}\cdot\texttt{evk}_{d_3}[0]), \mathbf{d_1} + P^{-1}(\mathbf{d_2}\cdot\texttt{evk}_{d_2}[1] + \mathbf{d_3}\cdot\texttt{evk}_{d_3}[1])\big) \pmod{q} \in \mathcal{R}_q \times \mathcal{R}^r_q$.

- $\texttt{ModCKKS.Rescale}(\texttt{ct})$: Since every message has an inherent scaling factor, say $\Delta \geq 1$ to preserve a certain degree of precision, multiplications between ciphertexts also result in an exponential increase in the scaling factor size. Let, at a level $l, 0 < l \leq L$, $q_l = \Delta^l \cdot q_0$. Therefore, to keep the scale constant and also to reduce the noise, we can define the rescaling operation of a ciphertext $\texttt{ct}$ in a level l to a level $l-1$ as, $\texttt{ct}' = \left\lfloor \frac{q_{l-1}}{q_l} \cdot \texttt{ct} \right\rceil \pmod{q_{l-1}}$.

Additionally, we also mention the two subroutines of CKKS called rotation and conjugation. It is intuitive to understand how messages in the same i-th plaintext slot of a ciphertext could be added or multiplied. However, to operate between messages in two different plaintext slots, there should be a way to permute data easily across slots of the ciphertext. In this regard, permutations and rather automorphisms of the associated algebraic ring can be efficiently used for the process of rotation of slots. Like, for a polynomial a in a power-of-two cyclotomic ring $\mathcal{R}$ replacing it with $a^{(k)}(X) = a(X^k) \pmod{\phi_{2N}(X)}$, $k \in \mathbb{Z}^*_{2N}, k > 1$ would result only in a permutation of the coefficients. In particular, $\kappa_k : a(X) \to a(X^k) \pmod{\phi_{2N}(X)}$ would serve as the suitable rotation map. So a ciphertext $\texttt{ct}$ encrypting a

message m with respect to a secret key $\mathbf{s}$ would have a 'rotated' ciphertext $\kappa_k(\mathtt{ct})$ which is a valid encryption of $\kappa_k(m)$ with secret key $\kappa_k(\mathbf{s})$. This operation is equivalent to a key-switching technique similar to the ring setting and requires a switching key $\mathtt{swk}$. Notice that one of the generating sets of the group of units $\mathbb{Z}_{2N}^*$ is $\{5, -1\}$ as the integer 5 has an order $N/2$ in $\mathbb{Z}_{2N}^*$. If $k = 5^{i-j} \pmod{2N}$ for $0 \leq i, j < N/2$, then $m(\xi_j^{5^{i-j}}) = m(\xi_i)$, which resembles a permutation of the plaintext slots.

- $\mathtt{ModCKKS.Rotate}(\mathtt{ct}, \mathtt{rtk})$: The associated rotation key can be written as, $\mathtt{rtk} = (-\mathbf{A}_{rot} \cdot \mathbf{s} + \mathbf{e} + P \cdot \kappa_k(\mathbf{s}), \mathbf{A}_{rot}) \pmod{P \cdot q}$. The rotated ciphertext is $\mathtt{ct}' = (c_0, 0) + (P^{-1} \cdot \mathbf{c_1} \cdot \mathtt{rtk}) \pmod{q} \in \mathcal{R}_q \times \mathcal{R}_q^r$.

- $\mathtt{ModCKKS.Conjugate}(\mathtt{ct}, \mathtt{cjk})$: Another property to note here is that for $2N$-th primitive roots of unity ξ_j, we know $\bar{\xi_j} = \xi_j^{-1}$. Then for a polynomial a in $\mathcal{R}$, $\overline{a(\xi_j)} = a(\bar{\xi_j}) = a(\xi_j^{-1})$. Considering $k = -1$ for the mapping κ_k, the ciphertext $\bar{\mathtt{ct}}$ can be seen as the encryption of the conjugate $\bar{\mathbf{z}}$ of a vector $\mathbf{z}$. The conjugate key is $\mathtt{cjk} = (-\mathbf{A}_{cj} \cdot \mathbf{s} + \mathbf{e} + P \cdot \kappa_{-1}(\mathbf{s}), \mathbf{A}_{cj}) \pmod{P \cdot q}$ and $\bar{\mathtt{ct}} = (c_0, 0) + (P^{-1} \cdot \mathbf{c_1} \cdot \mathtt{cjk}) \pmod{q} \in \mathcal{R}_q \times \mathcal{R}_q^r$.

3.1 Noise estimations

Note that in a ring setting, the error bound grows in proportion to the size of the ring. Instead, in a module setting, the size of the ring remains fixed and the change in the error bound follows the rank of the module. For a similar security level, the lattice dimension $N \cdot r$ of an MLWE instance with the rank r and ring dimension N will be roughly equal to the ring dimension N' of the RLWE instance. Therefore the error growth in an MLWE instance with lattice dimension $N \cdot r$ will be similar to an RLWE instance with ring size N'.

We discuss the following error estimations along the lines of [CKKS17], [CHK+18b], and [CS16]. First, we mention a few facts 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 1. *The error induced during encryption is bounded by* $B_{enc} = 16r\sigma(N/\sqrt{2} + \sqrt{hN}) + 6\sigma\sqrt{N}$.

Proof. Consider the decryption equation of a ciphertext $\mathtt{ct}$ encrypting a message m.

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

Let, $E = (\sum_{i=0}^{r-1} e_{i_{pk}} \cdot v_i + e + \sum_{i=0}^{r-1} e_i' \cdot s_i)$. The upper bound can then be established

with the following inequality:

$$\begin{aligned}\|E\|_{\infty}^{can} &\leq r \cdot \frac{16 \cdot \sigma \cdot N}{\sqrt{2}} + 6\sigma\sqrt{N} + r \cdot 16\sigma\sqrt{hN} \\ &= 16r\sigma(N/\sqrt{2} + \sqrt{hN}) + 6\sigma\sqrt{N}\end{aligned}$$

□

Lemma 2. *Intuitively, the error bound after one addition is the sum of the error bounds corresponding to the individual ciphertexts.*

Lemma 3. *The error induced in the rescaling step is bounded by* $B_{res} = 6\sqrt{N/12} + r \cdot 16\sqrt{hN/12}$.

Proof. The error induced during rescaling is because of the fact that we try to approximate a ciphertext $\mathtt{ct}$ with $\mathtt{ct}'$ using $\frac{q_{l-1}}{q_l}\mathtt{ct}$. Thus an error bound E_{res} can be found on the error vector using the expression,

$$\mathtt{ct}' - \frac{q_{l-1}}{q_l}\mathtt{ct} \pmod{q_{l-1}} = (\epsilon_0, \boldsymbol{\epsilon_1})$$

Assuming that each coefficient of the polynomials in the error vector has an approximate variance of $1/12$, we write the error bound during the decryption of this vector by the following inequality:

$$\begin{aligned}((\epsilon_0, \boldsymbol{\epsilon_1}), \mathbf{s}) &= (\epsilon_0 + \sum_{i=0}^{r-1} \epsilon_{1i} \cdot s_i) \\ &\leq \|\epsilon_0\|_{\infty}^{can} + \sum_{i=0}^{r-1} \|\epsilon_{1i} \cdot s_i\|_{\infty}^{can} \\ &\leq 6\sqrt{N/12} + r \cdot 16\sqrt{hN/12}.\end{aligned}$$

□

Lemma 4. *The cumulative error bound after a homomorphic multiplication is the sum of the bounds of* $B_{res} + B_{mult} + B_{relin}$ *where* B_{mult} *and* B_{relin} *are the upper bounds of errors induced during the actual multiplication steps and relinearization respectively.*

Proof. Homomorphic multiplication involves a series of operations, starting with actual multiplications, relinearization, and rescaling. Each of these steps contributes to the error growth. First, note that the multiplication $m \cdot m'$ is approximated by $(c_o + \mathbf{c_1} \cdot \mathbf{s})(c'_o + \mathbf{c'_1} \cdot \mathbf{s}) \pmod{q}$ with respect to the two ciphertexts $\mathtt{ct}$ and $\mathtt{ct}'$. Let $\langle \mathtt{ct}, \mathbf{s} \rangle = m + E \pmod{q}$ and $\langle \mathtt{ct}', \mathbf{s} \rangle = m' + E' \pmod{q}$ such that $\|E\|_{\infty}^{can}$ and $\|E'\|_{\infty}^{can}$ have error bounds B and B' respectively, then we may write the error expression corresponding to multiplication as:

$$\begin{aligned}(m + E)(m' + E') \pmod{q} &= (mm' + mE' + m'E + EE') \pmod{q} \\ \|E_{mult}\|_{\infty}^{can} &\leq \|mE' + m'E + EE'\|_{\infty}^{can} \\ &= \mu B' + \mu' B + BB'\end{aligned}$$

where, μ and μ' are respectively the upper bounds of the message space of m and m'. Next, some error is also induced during relinearization when $\mathbf{d_2}$ and $\mathbf{d_3}$ are multiplied with their respective evaluation keys. More precisely we can write the error during relinearization with the following inequality:

$$\begin{aligned}E_{relin} &= P^{-1} \cdot ((\mathbf{d_2} \cdot \mathbf{e_{d_2}} + \mathbf{d_3} \cdot \mathbf{e_{d_3}}) \pmod{P \cdot q}) \\ \|E_{relin}\|_{\infty}^{can} &\leq P^{-1} \cdot r(r+1)/2 \cdot 16\sqrt{\frac{Nq^2}{12}}\sigma\sqrt{N}\end{aligned}$$

□

3.2 RNS representation for ease of implementation: ModRNS.CKKS

In section 2.3, we described how the RNS representation can improve the efficiency of HE operations. A natural extension would be to also define an RNS variant of ModCKKS. We can define the module isomorphism between $\mathcal{R}_Q^r$ and the module $\prod_{i=0}^{k-1}\mathcal{R}_{q_i}^r$ which is the direct product of the modules $\mathcal{R}_{q_i}^r$, $i \in \{0, \cdots, k-1\}$ such that,

$$\mathbf{a} = (a_t)_{t=0}^{r-1} \mapsto ([a_0]_{\mathcal{C}}, [a_1]_{\mathcal{C}}, \cdots, [a_{r-1}]_{\mathcal{C}}) = [\mathbf{a}]_{\mathcal{C}}$$

and $[a_t]_{\mathcal{C}} = (a_t^{(0)}, a_t^{(1)}, \cdots, a_t^{(k-1)}) \in \prod_{i=0}^{k-1}\mathcal{R}_{q_i}$. For simplicity, we will use either the notation $\mathbf{a}^{(j)}$ or $\mathbf{A}^{(j)}$ as per context to specify that each component of the module element has its corresponding j residue polynomials. Following are the RNS variants of all ModCKKS algorithms. We denote them by ModRNS.

- $\texttt{ModRNS.Setup}(q, L, \eta; 1^\lambda)$: For security parameter λ we choose an integer q which determines the approximate basis, levels L as before, a bit precision η, ensuring $q_j/q \in (1-2^{-\eta}, 1+2^{-\eta})$. We choose a prime p and the basis $\mathcal{C} = \{q_0, \cdots, q_L\}$ such that at a certain level $0 \le l \le L$, $\mathcal{C}_l = \{q_0, \cdots, q_l\}$.

- $\texttt{ModRNS.KeyGen}(1^\lambda)$: Generate a secret key $\texttt{sk} = (1, \mathbf{s})$, $s_i \leftarrow \mathcal{HWT}(h)$.
  Sample $\mathbf{a} = [\mathbf{a}]_{\mathcal{C}}$'s such that the coefficients of each such residue polynomial are sampled uniformly from $\mathcal{U}$ over $\prod_{j=0}^{L}\mathcal{R}_{q_j}^r$. Generate public key $\texttt{pk} = \left(\texttt{pk}^{(j)} = (\mathbf{b}^{(j)}, \mathbf{A}^{(j)}) \in \mathcal{R}_{q_j}^r \times \mathcal{R}_{q_j}^{r\times r}\right)_{0\le j\le L}$ and evaluation key, $\texttt{evk} = \left(\texttt{evk}^{(j)} = (\mathbf{b}_{evk}^{(j)}, \mathbf{A}_{evk}^{(j)}) \in \mathcal{R}_{pq_j}^r \times \mathcal{R}_{pq_j}^{r\times r}\right)_{0\le j\le L+1}$, where $\mathbf{b}$ and $\mathbf{b}_{evk}$ are defined as in ModCKKS.KeyGen.

- $\texttt{ModRNS.Enc}(\texttt{m})$: We obtain a ciphertext, $\texttt{ct} = (\texttt{ct}^{(j)})_{0\le j\le L}$ such that each $\texttt{ct}^{(j)} = (\texttt{pk}^{(j)} \cdot \mathbf{v} + (m + e, \mathbf{e}')) \pmod{q_j} \in \mathcal{R}_{q_j} \times \mathcal{R}_{q_j}^r$.

- $\texttt{ModRNS.Dec}(\texttt{ct})$: For a ciphertext $\texttt{ct} = (\texttt{ct}^{(j)})_{0\le j\le L}$ retrieve an approximation of the message m under the secret key $\texttt{sk}$, $\langle \texttt{ct}^{(0)}, \texttt{sk}\rangle \pmod{q_0} \approx m$.

- $\texttt{ModRNS.Add}(\texttt{ct}, \texttt{ct}')$: Given two ciphertexts $\texttt{ct} = (\texttt{ct}^{(j)})_{0\le j\le l}$ and $\texttt{ct}' = (\texttt{ct}'^{(j)})_{0\le j\le l}$ both in $\prod_{j=0}^{l}\mathcal{R}_{q_j}^{r+1}$ at some arbitrary level l, their sum is given by, $\texttt{ct}_{\texttt{add}} = (\texttt{ct}_{\texttt{add}}{}^{(j)})_{0\le j\le l} \in \prod_{j=0}^{l}\mathcal{R}_{q_j}^{r+1}$ where $\texttt{ct}_{\texttt{add}}{}^{(j)} = \texttt{ct}^{(j)} + \texttt{ct}'^{(j)} \pmod{q_j}, 0 \le j \le l$ following the algorithm ModCKKS.Add.

- $\texttt{ModRNS.Mult}(\texttt{ct}, \texttt{ct}')$: For ciphertexts $\texttt{ct} = \left(\texttt{ct}^{(j)} = c_0^{(j)}, \mathbf{c_1}^{(j)}\right)_{0\le j\le l}$ and $\texttt{ct}' = \left(\texttt{ct}'^{(j)} = c_0'^{(j)}, \mathbf{c_1'}^{(j)}\right)_{0\le j\le l}$, we write the result $\texttt{ct}_{\texttt{mult}} = \texttt{d}$ of ModCKKS.Mult in the RNS form as $(d_0^{(j)}, \mathbf{d_1}^{(j)}, \mathbf{d_2}^{(j)}, \mathbf{d_3}^{(j)})$,

  $$\begin{aligned} d_0^{(j)} &= c_0^{(j)} \cdot c_0'^{(j)} & d_{1t}^{(j)} &= c_0^{(j)} \cdot c_{1t}'^{(j)} + c_0'^{(j)} \cdot c_{1t}^{(j)} \\ d_{2t}^{(j)} &= c_{1t}^{(j)} \cdot c_{1t}'^{(j)} & d_{3_{tt'}} &= c_{1t}^{(j)} \cdot c_{1t'}'^{(j)} + c_{1t}'^{(j)} \cdot c_{1t'}^{(j)},\ t < t', 0 \le t, t' < r \end{aligned}$$

  with all arithmetic done $\pmod{q_j}$, $0 \le j \le l$.

- $\texttt{ModRNS.Relin}(\texttt{ct}_{\texttt{mult}})$: We need to relinearize the module components $(\mathbf{d_2}^{(j)})_{0\le j\le l}$ and $(\mathbf{d_3}^{(j)})_{0\le j\le l}$. As in CKKS.Relin and in ModCKKS.Relin we perform an equivalent of the 'modulus up' operation of each $\mathbf{d_2}^{(j)}$ and $\mathbf{d_3}^{(j)}$ intrinsically using a prime p. Also, recall from ModCKKS.Relin that the evaluation key, evk is in the modulo domain of $P \cdot q$. In RNS representation we denote each of its residues $\pmod{pq_j}$

by $\texttt{evk}^{(j)} = (\mathbf{b}_{evk}^{(j)}, \mathbf{A}_{evk}^{(j)}) \in \mathcal{R}_{pq_j}^r \times \mathcal{R}_{pq_j}^{r\times r}$. Let, $(d_0'')^{(j)} = \big(\mathbf{d_2}^{(j)} \cdot \texttt{evk}_{\texttt{d}_2}{}^{(j)}[0] + \mathbf{d_3}^{(j)} \cdot \texttt{evk}_{\texttt{d}_3}{}^{(j)}[0]\big) \in \mathcal{R}_{pq_j}$ and $(\mathbf{d_1}'')^{(j)} = \big(\mathbf{d_2}^{(j)} \cdot \texttt{evk}_{\texttt{d}_2}{}^{(j)}[1] + \mathbf{d_3}^{(j)} \cdot \texttt{evk}_{\texttt{d}_3}{}^{(j)}[1]\big) \in \mathcal{R}_{pq_j}^r$. The relinearized ciphertext is then $\texttt{ct}_{\texttt{relin}}$ such that each of its residue shares is given by, $(\texttt{ct}_{\texttt{relin}})^{(j)} = (d_0'^{(j)}, \mathbf{d_1}'^{(j)})_{0\le j\le l}$ such that $d_0'^{(j)} = (d_0^{(j)} + \texttt{CKKS.ModDown}\big((d_0'')^{(j)}\big)) \pmod{q_j} \in \mathcal{R}_{q_j}$ and $\mathbf{d_1}'^{(j)} = (\mathbf{d_1}^{(j)} + \texttt{CKKS.ModDown}\big((\mathbf{d_1}'')^{(j)}\big)) \pmod{q_j} \in \mathcal{R}_{q_j}^r$.

- $\texttt{ModRNS.Rescale}(\texttt{ct})$: A ciphertext $\texttt{ct} = (\texttt{ct}^{(j)})_{0\le j\le l} \in \prod_{j=0}^{l} \mathcal{R}_{q_j} \times \prod_{j=0}^{l} \mathcal{R}_{q_j}^r$ is changed into a ciphertext $\texttt{ct}' = (\texttt{ct}'^{(j)})_{0\le j\le (l-1)} = \big(q_l^{-1}\cdot(c_0^{(j)} - c_0^{(l)}), q_l^{-1}\cdot(\mathbf{c_1}^{(j)} - \mathbf{c_1}^{(l)}) \pmod{q_j}\big)_{0\le j\le (l-1)} \in \prod_{j=0}^{l-1} \mathcal{R}_{q_j} \times \prod_{j=0}^{l-1} \mathcal{R}_{q_j}^r$.

We present pseudo-codes of the major ModRNS algorithms- Algorithm 5, 6, 7, 8. We use the representation $\prod_{j=0}^{l} \mathcal{R}_{q_j}^r = \mathcal{R}_{Q_l}^r$ in the pseudo-codes.

Algorithm 5 ModRNS.Add Algorithm

In: $\boldsymbol{ct} = (\tilde{c}_0, \tilde{\mathbf{c}}_1)$, $\boldsymbol{ct'} = (\tilde{c}'_0, \tilde{\mathbf{c}}'_1) \in R_{Q_L}^{r+1}$
Out: $\boldsymbol{d} = (\tilde{d}_0, \tilde{\boldsymbol{d}}_1) \in R_{Q_L}^{r+1}$
1: $\tilde{d}_0 \leftarrow \tilde{c}_0 + \tilde{c}'_0$
2: **for** $i = 0$ to $r-1$ **do**
3: $\quad \tilde{d}_1[i] \leftarrow \tilde{c}_1[i] + \tilde{c}'_1[i]$
4: **end for**

Algorithm 6 ModRNS.Mult Algorithm

In: $\boldsymbol{ct} = (\tilde{c}_0, \tilde{\mathbf{c}}_1)$, $\boldsymbol{ct'} = (\tilde{c}'_0, \tilde{\mathbf{c}}'_1) \in R_{Q_L}^{r+1}$
Out: $\boldsymbol{d} = (\tilde{d}_0, \tilde{\boldsymbol{d}}_1, \tilde{\boldsymbol{d}}_2, \tilde{\boldsymbol{d}}_3) \in R_{Q_l} \times R_{Q_l}^r \times R_{Q_l}^r \times R_{Q_l}^{r(r-1)/2}$
1: $\tilde{d}_0 \leftarrow \tilde{c}_0 \star \tilde{c}'_0$
2: **for** $i = 0$ to $r-1$ **do**
3: $\quad \tilde{d}_1[i] \leftarrow \tilde{c}_0[i] \star \tilde{c}'_1[i] + \tilde{c}_1[i] \star \tilde{c}'_0[i]$
4: $\quad \tilde{d}_2[i] \leftarrow \tilde{c}_1[i] \star \tilde{c}'_1[i]$
5: $\quad$ **for** $j = 1$ to $r-1$ **do**
6: $\quad\quad$ **if** $i < j$ **then**
7: $\quad\quad\quad \tilde{d}_3[i,j] \leftarrow \tilde{c}_1[i] \star \tilde{c}'_1[j] + \tilde{c}_1[j] \star \tilde{c}'_1[i]$
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: **end for**

4 Potentials and limitations of MLWE-based homomorphic encryption

Although the shift from rings to modules looks almost seamless, there are a few factors we need to analyze when deciding the module rank and the security of the scheme.

4.1 Hardware-reusability and fixed optimization target

RLWE-based homomorphic encryption schemes set their ring dimensions based on the desired level of security and the multiplicative depth. For example, CKKS [CKKS17] sets the degree of polynomial to 2^{14} or 2^{15} for the multiplicative depths 6 and 10 respectively. Note that, an increase in the multiplicative depth increases the ciphertext modulus size

Algorithm 7 `ModRNS.SubRelin` Algorithm

In: $\acute{\boldsymbol{d}} \in R^r_{Q_l}$
In: $\tilde{\texttt{Evk}}_0 \in R^{r\cdot L}_{pQ_L}, \tilde{\texttt{Evk}}_1 \in R^{t\cdot r\cdot L}_{pQ_L}$
Out: $\boldsymbol{d} = (\acute{d}_0, \acute{\boldsymbol{d}}_1) \in R^{r+1}_{Q_L}$

for $k = 0$ to $r-1$ **do**
  for $j = 0$ to l **do** ▷ Here q_l is used to represent special prime p
    for $i = 0$ to $l-1$ **do**
      $\tilde{\boldsymbol{u}} \leftarrow \texttt{NTT}(\texttt{INTT}(\big[\acute{\boldsymbol{d}}[i][k]\big])_{q_j})$ ▷ in $\mathbb{Z}_{q_j}$
      $\acute{d}_0 \leftarrow \acute{d}_0 + \tilde{\texttt{Evk}}_0[i][j][k] \star \tilde{\boldsymbol{u}}_{q_j}$
      $\acute{d}_1[j,k] \leftarrow \sum_{t=0}^{r-1} \tilde{\texttt{Evk}}_1[i][j][k][t] \star \tilde{\boldsymbol{u}}_{q_j}$
    end for
  end for
end for

Algorithm 8 `ModRNS.Relin` Algorithm

In: $\boldsymbol{d} = (\tilde{d}_0, \tilde{\boldsymbol{d}}_1, \tilde{\boldsymbol{d}}_2, \tilde{\boldsymbol{d}}_3) \in R_{Q_l} \times R^r_{Q_l} \times R^r_{Q_l} \times R^{r(r-1)/2}_{Q_l}$
In: $\tilde{\texttt{Evk}}_0 \in R^{(r\cdot(r+1)\cdot L)/2}_{pQ_L}, \tilde{\texttt{Evk}}_1 \in R^{(r\cdot r\cdot(r+1)\cdot L)/2}_{pQ_L}$
Out: $\boldsymbol{d}' = (\tilde{d}'_0, \tilde{\boldsymbol{d}}'_1) \in R^{r+1}_{Q_L}$

$\boldsymbol{t_0}, \boldsymbol{t_1} \leftarrow 0$ ▷ $(\boldsymbol{t_0}, \boldsymbol{t_1}) \in R^{((r+1)\cdot L)/2}_{pQ_L} \times R^{(r\cdot(r+1)\cdot L)/2}_{pQ_L}$
$(t_0[0], \boldsymbol{t_1}[0]) \leftarrow \texttt{ModCKKS.SubRelin}(\tilde{\boldsymbol{d}}_2, \tilde{\texttt{Evk}}_0[0], \tilde{\texttt{Evk}}_1[1])$
for $j = 1$ to $(r+1)/2$ **do**
  $(t_0[j], \boldsymbol{t_1}[j]) \leftarrow \texttt{ModCKKS.SubRelin}(\tilde{\boldsymbol{d}}_3[j-1], \tilde{\texttt{Evk}}_0[j], \tilde{\texttt{Evk}}_1[j])$
end for
$\tilde{\boldsymbol{d}}'_0 \leftarrow \sum_{j=0}^{(r+1)/2} t_0[j] + \texttt{CKKS.ModDown}(\tilde{\boldsymbol{c}}''_0)$, $\tilde{\boldsymbol{d}}'_1 \leftarrow \sum_{j=0}^{(r+1)/2} \boldsymbol{t}_1 + \texttt{CKKS.ModDown}(\tilde{\boldsymbol{c}}''_1)$

which results in diminished security. To compensate for the security loss, the dimension of the lattice is increased by increasing the polynomial degree. The polynomial size changes by a factor of two for different multiplicative depths when cyclotomic rings are used.

Such variations in the polynomial degree in the RLWE-based homomorphic encryption scheme make hardware implementation of a parameter-flexible architecture challenging. Furthermore, when the polynomial degree is 2^{13} or larger, implementation of the large accelerator circuit in hardware becomes very challenging due to low resource utilization, placement, routing, slow clock frequency, and limited on-chip memory, as discussed in [RLPD20, MAK+23].

MLWE-based homomorphic encryption scheme avoids these problems due to its modular nature. This is explained as follows. When working with modules, we can choose a base ring of some suitable (and small) dimension, say $N = 2^k$, and then we can adjust the rank r of the module as per our requirements. This is a good way to restrain the degree of the polynomial from increasing every time we want more depth.

Furthermore, it becomes easier to optimize the ASIC/FPGA implementations as the ring size is fixed and small in MLWE. For a given and fixed ring size in MLWE, hardware architecture designers could now focus on optimizing the implementation (with fixed parameters) to achieve better clock frequency, memory utilization, area optimization, placement, routing, etc.

4.2 Increased scope for parallel computations

In our `ModRNS` construction, mod-switching and key-switching parts of the relinearization step are performed on independent module components, $(\boldsymbol{d}_2, \boldsymbol{d}_3) \in R^r_{Q_l} \times R^{r(r-1)/2}_{Q_l}$ in

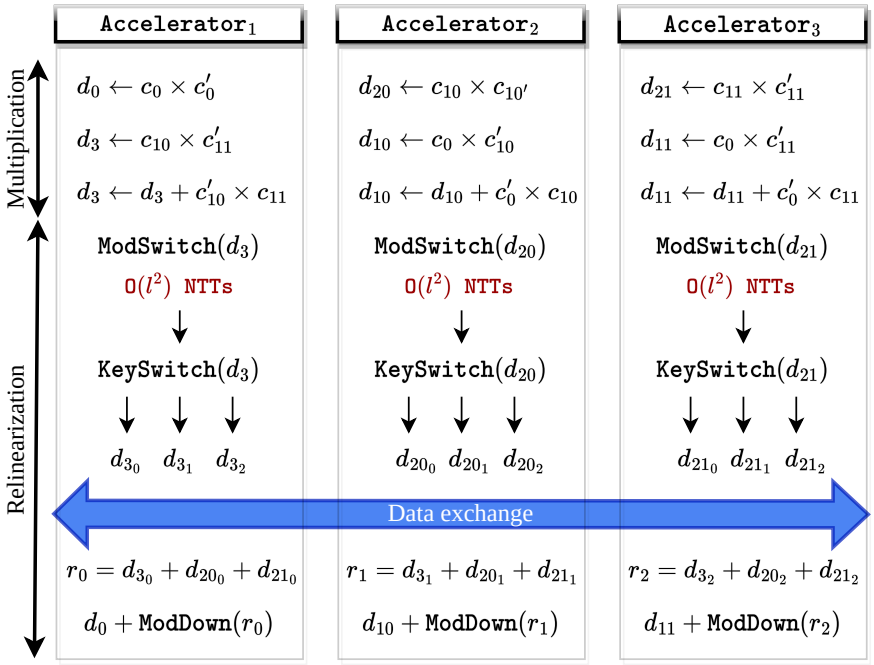

Figure 1: An operation schedule of the relinearization step demonstrating massive parallelism and limited communication for module-dimension $r = 2$

parallel. At the end of it, an accumulation is performed to combine all the results. For the $r = 2$ setting discussed above, $\boldsymbol{d}_2$ has two ring elements, and $\boldsymbol{d}_3$ has one ring element. Therefore, in total, we need to perform three relinearizations. All of these components are data and computation-independent of each other through the relinearization until the very end when accumulation is performed. Thus, in a multi-accelerator setting (common in clouds), dedicated accelerators can be used to compute these three operations in parallel. An example with three such accelerators is shown in Figure 1. We observe that until the very end of the multiplication and key-switching, there was no communication required between the accelerators. Note that frequent inter-accelerator data exchanges due to data-dependent steps are problematic for parallel processing. The same applies to multi-threaded or multi-processor software systems.

4.3 Ciphertext compression due to module rank reduction

Module-based homomorphic encryption scheme offers additional flexibility in determining the dimension of the lattice problem to work with. This also let us adapt the module rank depending on the size of the ciphertext modulus. In a leveled homomorphic encryption scheme, the ciphertext modulus is initially the largest and then it gradually reduces with the multiplicative depth. Therefore, we may scale down the lattice problem from the initial dimension $N \cdot r$ to a smaller $N \cdot r'$ where $r > r'$ while keeping the security of the scheme in place. Smaller ciphertexts would also mean a reduced decryption complexity, lower computation overhead, and smaller key size.

4.4 Message packing and Key size

As previously mentioned, the packing of plaintext messages does not increase with an increase in the module dimension. So, applications that only need sparse packing can still benefit from the added depth that ModHE provides. Note that, in view of the canonical

embedding map $\tau : \mathcal{K} \to \mathbb{C}^N$, the map $(\tau, \tau, \cdots, \tau)$ will be an embedding from $\mathcal{K}^d \to \mathbb{C}^{Nr}$, $r \leq d$. For an $\mathcal{R}$-module $M \subseteq \mathcal{K}^d$ (if M is full rank then $r = d$) let us then denote this embedding as τ_M. The set $\tau_M(M)$ is then a module lattice of dimension Nr. This map could help to extend packing along the rank of a module and its use for applications that necessarily require full packing of plaintexts could be explored.

Another limitation of using MLWE is that the number of (small) ring-level operations as well as the number of polynomials in the keys increases quadratically with the module dimension r. More generally, with the module rank r, there will be $O(r^2)$ polynomial multiplications where each polynomial is of size N/r coefficients. In comparison, homomorphic multiplication in RLWE requires only 4 polynomial multiplications of size N coefficients. Therefore, the proposed MLWE-based scheme performs more polynomial operations than the corresponding RLWE-based scheme for the same parameter set. Although the number of polynomials is higher in MLWE than in RLWE, the polynomials in MLWE are smaller by r times than those in RLWE. Therefore, choosing a too-small ring size and a very large module rank will not be an ideal design decision for a homomorphic encryption scheme.

Next, to understand how the rank r of the module affects the total number of keys, we recall that for the secret vector $\mathbf{s} = \{s_0, \cdots, s_{r-1}\}$, we require relinearization keys containing the product of every two of its components, that is, $\sum_{i,j=0}^{r-1} s_i s_j$ but with terms of the form $s_i s_j$ and $s_j s_i$ for a pair of (i, j) absorbed into the same relinearization key. Thus the total number of keys would be $\frac{r(r+1)}{2}$ relinearization keys with respect to each distinct quadratic secret component along with the usual r decryption keys. Choice of optimal parameters thus depends mainly on the level of security we desire, the amount of packing and multiplicative circuit depth an application demands, as well as polynomial degree-relinearization key size tradeoffs based on computational feasibility.

5 Hardware architecture for `ModRNS` and its evaluation

In the literature, several ASIC- and FPGA-based hardware accelerators for homomorphic encryption have been proposed. Present-day ASIC implementation papers [FSK+21, KLK+22] typically use RTL-based behavioral models of their accelerators for obtaining performance estimates. Several FPGA-based accelerators e.g., [MAK+23, AdCY+23, RLPD20], etc., present performance results after implementing the full-stack system.

In this paper, along with a sage implementation, we define a proof-of-concept hardware architecture for the proposed RNS-based and module lattice-based homomorphic encryption scheme, write its RTL design, and evaluate its area and performance using the Xilinx design tool for the Xilinx Alveo U280 accelerator card. We do not target a full system implementation, for example, prototyping in actual hardware, as we believe the new module lattice-based scheme is in its early stage and needs further algorithmic and mathematical advancements before becoming ready for engineering explorations. We provide area and performance estimates obtained after compiling the RTL of our accelerator using the Xilinx toolchain.

5.1 Hardware architecture

Intuitively, the proposed MLWE-based homomorphic encryption scheme `ModRNS` adds an additional layer of abstraction with respect to the RLWE-based RNS-CKKS scheme – a module element is a collection of ring elements. Hence, RLWE-based and MLWE-based homomorphic encryption schemes share very similar homomorphic routines and arithmetic operations. For example, the relinearization operation of RNS-CKKS (Algorithm 4) is very similar to the relinearization subroutine of `ModRNS` (Algorithm 7). Both operations include NTT/INTT and multiplication with evaluation keys. The `ModRNS` performs the ring-relinearization subroutine several times to perform the module-relinearization operation

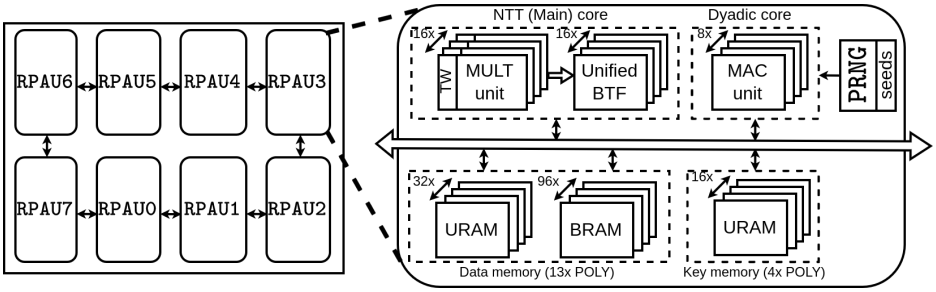

Figure 2: High-level architecture of the proposed design which follows the ring-based interconnect proposed in [MAK+23]. Each RPAU has a 16-core NTT coupled with a 16-core twiddle factor generator, an 8-core dyadic unit, and on-chip BRAM/URAMs.

as shown in Algorithm 8. Thus, a hardware architecture designed for an RLWE-based homomorphic encryption scheme can be re-used for an MLWE-based scheme with very few changes. For example, any accelerator architecture for the RNS-CKKS scheme can be adapted to accelerate the homomorphic operations of the `ModRNS`. To that end, we followed the design approach of an existing instruction-based accelerator architecture, Medha [MAK+23], and showcase how with minimal changes it can efficiently support the `ModRNS` scheme. As a proof of concept, we selected polynomial degree N as 2^{14}, module dimension r as 2, and the number of small RNS primes as 15.

Medha's [MAK+23] arithmetic units are designed for a polynomial size of $N = 2^{14}$. Medha designs and implements residue polynomial arithmetic units (RPAU) for each small RNS prime used in the RNS-CKKS scheme where each RPAU has an NTT (main) unit, a dyadic unit, and on-chip memory for data and key storage. Medha can execute NTT and dyadic operations in parallel and it tailored its computational resources accordingly for implementing the relinearization operation of RNS-CKKS efficiently. For example, the relinearization operation of RNS-CKKS requires one NTT and two evaluation key multiplications (steps 7,8 of Algorithm 4). Thus, Medha used a 16-core NTT unit and a 4-core dyadic unit. In `ModRNS`, on the other hand, based on the module dimension r, the relinearization operation needs $r(r+1)/2$ evaluation key multiplications. For example, for $r = 2$, it requires three multiplications. Thus, we used an 8-core dyadic unit instead of a 4-core dyadic unit in each RPAU.

Medha uses the Xilinx Alveo U250 card as the target platform and uses only on-chip BRAMs and URAMs to store evaluation keys. As explained in Section 4, one limitation of `ModRNS` is that the key size can increase with large module dimensions. Thus, using only on-chip memory resources for storing the evaluation keys can become more challenging for large module dimensions. To that end, for our proof-of-concept design, we selected the Xilinx Alveo U280 card, equipped with a high bandwidth memory (HBM) providing 460GB/s, as our target platform. This high-speed data transfer eliminates excessive use of on-chip memory, and it also enables storing evaluation keys in onboard HBM and reading them during homomorphic operations without degrading the performance. The increased key size of `ModRNS` does not have an impact on the performance as we utilize HBM's bandwidth to load the keys when needed. We employ 8 RPAUs where each RPAU can support two small RNS primes by employing reconfigurable modular reduction units. For the rest of the architecture, we followed the same approach as Medha.

We synthesized our RTL for the Xilinx Alveo U280 accelerator card using Vivado 2022.2. The synthesis results show that our design consumes only 700,773 LUTs, 463,293 FFs, 3,200 DSPs, 1,034 BRAMs, and 305 URAMs. It could achieve a clock frequency of 120MHz. A very high-level architecture diagram of the proposed accelerator is shown in Fig. 2.

5.2 Performance benchmarking

Estimated performance results for `ModRNS.Add`, `ModRNS.Mult` and `ModRNS.Relin` operations are presented in Table 1. It should be noted that the proposed accelerator can be used in a multi-FPGA setting where each FPGA implements the accelerator of Fig. 2 to improve the performance as shown in Fig. 1. Cloud providers such as Amazon AWS use FPGA stacks for accelerating computation. Our base ring dimension (when $r = 1$) is 2^{14}, with the packing of 2^{13}, and for higher multiplicative depth, we increase the $r = 2$. Note that at this point, we have more multiplicative depth; however, the maximum available packing stays fixed at 2^{13}.

Table 1: Estimated performance figures

N	r	**Depth** (L)	**Homomorphic Operation**	**Perf. (Cycle)** (1 accelerator)	**Perf. (Cycle)** ($\frac{r(r+1)}{2}$ accelerators)
2^{14}	1	7	Add.	1,152	1,152
2^{14}	1	7	Mult.	2,560	2,560
2^{14}	1	7	Mult.+Relin.	99,448	99,448
2^{14}	2	15	Add.	3,456	1,152
2^{14}	2	15	Mult.	13,824	4,608
2^{14}	2	15	Mult.+Relin.	1,193,376	397,792

The fixed amount of packing is one limitation of the `ModRNS` scheme, as we go higher up in the module dimension, the amount of packing does not increase, unlike RLWE-based schemes. However, large packing is only important for applications that require full packing, for example, machine learning training. Most machine learning inference applications do not require full packing and, on the other hand, might require more depth. Thus, using `ModRNS` for the later cases is likely to give comparable run time to the RLWE-based constructions and ensure hardware reusability.

We also provide an estimated benchmark for the logistic regression inference. For computing the logistic function $g(x) = 1/(1 + e^x)$ homomorphically, we use Taylor's polynomial approximation up to polynomial degree-9 i.e., $g(x) \approx 1/2 + 1/4x - 1/48x^3 + 1/480x^5 - 17/80640x^7 + 31/1451520x^9$. The proposed `ModRNS` hardware architecture consumes 3.8M clock cycles and requires 31.7 ms using a three accelerator setting (see Fig. 1) when the FPGA-based accelerator runs at 120 MHz. The same application consumes 1.8 seconds using the SEAL library with polynomial degree $N = 2^{15}$ on one Intel(R) Core(TM) i5-10210U CPU running at 1.60GHz. Although we only provide estimates for logistic regression, similar estimations can be extended to other function approximations, for example, sine, cosine, etc.

6 Future scopes

6.1 Bootstrapping

The next step would be to analyze the bootstrapping procedure of `ModCKKS`. [CHK+18a] presents a series of procedures for bootstrapping in the CKKS/HEAAN scheme. It aims at refreshing or recrypting a ciphertext in a low level of modulus q to produce an equivalent ciphertext with a larger modulus $Q \gg q$ such that both of these ciphertexts decrypt approximately to the same message over their respective moduli. This approximate decryption would also involve a modular reduction resulting in a polynomial of a substantial degree and evaluating it during refreshing would again consume a lot of levels. To avoid this problem, [CHK+18a] instead proposed approximating the modular reduction function with a scaled sine function, both possessing a periodic behavior in a given suitable

interval and ultimately exploiting its relation with the complex exponential function to arrive at a desired polynomial approximation. The sequence of high-level routines involved in CKKS bootstrapping consists of `ModRaise`, `CoeffToSlot`, `EvalExp`, `ImgExt` and `SlotToCoeff` respectively. Since these routines are part of a homomorphic recryption, they are built upon combinations of fundamental homomorphic functions. For example, `CoeffToSlot` and `SlotToCoeff` involve linear transformations of vectors over plaintext slots, which in turn involve homomorphic multiplication and rotation functions. Likewise, conjugation is required for extracting the imaginary part of the recrypted ciphertext in `ImgExt`. Therefore, bootstrapping in the module variant of CKKS could follow along these lines. It serves as an interesting future scope for a complete analysis of the process as well as its feasible implementation.

6.2 Other HE schemes and MLWE

At the beginning of the paper we listed quite a few popular HE schemes such as the BFV/FV [FV12] and the BGV [BGV11] schemes that rely on the RLWE problem. Although these schemes differ in their concrete noise management techniques, they share a similar underlying structure for their arithmetic circuits. For example, the polynomials involved in the en(de)cryption routines of the FV [FV12] HE scheme could be replaced by module elements such that the ciphertext `ct` now becomes a vector of polynomials contained in the module. The decryption is still valid as all modulo operations with respect to the ciphertext modulus or the plaintext modulus can be performed component-wise over the resulting module elements. In fact, homomorphic functions like homomorphic addition or multiplication could be extended to operate over module elements just like we showcased in our MLWE construction. One has to however apply the relinearization techniques with some caution so as to maintain the correctness of the decryption. A similar intuition can also be applied to the BGV scheme [BGV11] as well. Therefore, it is not just CKKS but also the other RLWE-based HE schemes that could be ported to the module setting.

6.3 Memory-centric platforms

Various memory-centric platforms like processing-in-memory (PIM) and computing-in-memory (CIM) are emerging as the future paradigms of computing. PIM [AMY+23] platforms are ideal when there are several small computation tasks in an algorithm. Operands for the small tasks could be fetched fast from the near-memory, then processed in the processing elements of the platform, and finally written back in low latency to the near-memory. CIM platforms [VJV+19] offer computation in memory so data movement is minimal and is within the memory. The smaller the computation requirement, the easier it is to perform it in this setting.

The proposed `ModRNS` is a suitable algorithm for such platforms, owing to its ability to process smaller rings for homomorphic operations instead of one big ring. The NTT/INTT transforms, and the remaining homomorphic operations can be more efficiently evaluated using the `ModRNS` setting. Expensive operations such as relinearization are done independently on each non-linear module element. Even within a module element that consists of a component equal to the rank of the module, each of the module components is operated upon separately. Once these operations are done, a data share is indeed required to add the result to the remaining elements. The proposed hardware architecture is optimized for module components and features parallel dyadic units that can utilize all three relinearization key components in parallel.

In the emerging memory landscape, Micron Technology Inc. introduced 3D stacked memory [Sch], called Hybrid Memory Cube (HMC) that offers an aggregate bandwidth of 480 GBps at much lower power and a smaller form factor. More recently AMD designed the 3D Chiplets [JP] that can offer up to 2 terabytes of bandwidth. Such Chiplets feature

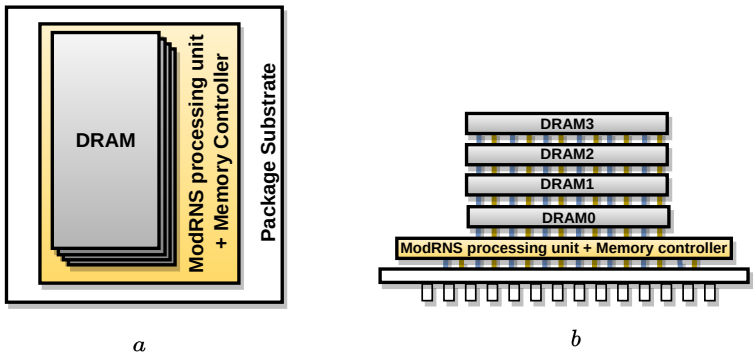

Figure 3: (a) shows the side-view of the 3D stack where the bottom layer after the substrate is the logic layer and the upper layers are memories, and (b) is a cross-sectional view to show the TSV interconnects connecting multiple different layers.

multiple DRAM layers stacked on top of each other and connected through high bandwidth through silicon vias (TSVs) as shown in Figure 3. The bottom layer of the 3D stack is the logic layer which contains the necessary *computation circuits* for the memory controller. As the logic layer could not be arbitrarily large, a small `ModRNS` processing unit could be placed in the logic layer. All the above memory layers can be used to store all the required keys and ciphertexts. Thus, the data can be fetched fast via TSVs (i.e., high bandwidth), processed fast in the `ModRNS` processing unit, and then stored back fast into the memory layers. In this way, data will never leave the memory cube and thus the classical memory-processor data transfer bottlenecks will be avoided.

7 Conclusions and future directions

In this paper we discussed the use of module lattices to instantiate homomorphic encryption schemes and as a concrete construction present `ModCKKS`, a module version of the original ring-based CKKS scheme. We then proposed an RNS variant of the scheme to further improve the efficiency of HE operations. The original scheme's sub-routines were adapted in a way that requires minimum deviations from its fundamental approaches. In addition to reaping the benefits of the ring-based CKKS, our scheme also aids hardware reusability and long-lived parallel computation opportunities. It eliminates the need to increase the polynomial degree every time we want our scheme to support a higher level of security. We also made a few important observations about module-based HE constructions: while ModHE can flourish in environments that need greater circuit depth, selecting appropriate parameter trade-offs is vital for achieving its full potential. It faces shortcomings in the context of an increased load on memory and the amount of computations. We discussed a hardware implementation that is consistent with our primary goal of reusability and flexibility. We then provided results for homomorphic operations in the module setting.

MLWE-based HE schemes are fairly unexplored: there is immense scope for investigation of its properties and designing concrete hardware prototypes. Efficient bootstrapping for ModHE and detailed analysis of MLWE instances of other popular ring-based HE schemes are the major future direction of research in this area. We believe that experimenting with different parameter sets and improving functionalities of module-based homomorphic encryption in order to overcome its limitations would indeed be worthwhile.

8 Acknowledgement

This work was supported in part by the Samsung Electronics co. ltd., Samsung Advanced Institute of Technology and the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

References

[AdCY+23] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Tugce Yazicigil, Anantha P. Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. FAB: an fpga-based accelerator for bootstrappable fully homomorphic encryption. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023*, pages 882–895. IEEE, 2023.

[AMY+23] Kazi Asifuzzaman, Narasinga Rao Miniskar, Aaron R. Young, Frank Liu, and Jeffrey S. Vetter. A survey on processing-in-memory techniques: Advances and challenges. *Memories - Materials, Devices, Circuits and Systems*, 4:100022, 2023.

[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.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electron. Colloquium Comput. Complex.*, page 111, 2011.

[BHM+20] Ahmad Al Badawi, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. Privft: Private and fast text classification with homomorphic encryption. *IEEE Access*, 8:226544–226556, 2020.

[BV11a] 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.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 505–524, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[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.

[CHK+18a] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 360–384, Cham, 2018. Springer International Publishing.

[CHK+18b] 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.

[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.

[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.

[DKR+21] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Proposal to NIST PQC Standardization, Round3, 2021. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[Elg85] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FSK+21] Axel Feldmann, Nikola Samardzic, Aleksandar Krastev, Srini Devadas, Ron Dreslinski, Karim Eldefrawy, Nicholas Genise, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption (extended version), 2021.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, page 144, 2012.

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

[GVBP+22] Robin Geelen, Michiel Van Beirendonck, Hilder V. L. Pereira, Brian Huffman, Tynan McAuley, Ben Selfridge, Daniel Wagner, Georgios Dimou, Ingrid Verbauwhede, Frederik Vercauteren, and David W. Archer. Basalisc: Flexible asynchronous hardware accelerator for fully homomorphic encryption, 2022.

[JKA+21] Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):114–148, Aug. 2021.

[JP] Cadence John Park, Product Management Group Director for Advanced IC Packaging. Chiplets and heterogeneous packaging are changing system design and analysis. https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/ic-package-design-analysis/chiplets-and-heterogeneous-packaging-are-changing-system-design-and-analysis.pdf accessed on 14 April, 2023.

[KKK+22] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. Bts: An accelerator for bootstrappable fully homomorphic encryption. In *Proceedings of the 49th Annual International*

Symposium on Computer Architecture, ISCA '22, page 711–725, New York, NY, USA, 2022. Association for Computing Machinery.

[KLK+22] Jongmin Kim, Gwangho Lee, Sangpyo Kim, Gina Sohn, John Kim, Minsoo Rhu, and Jung Ho Ahn. Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse, 2022.

[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.

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

[MAK+23] Ahmet Can Mert, Aikata, Sunmin Kwon, Youngsam Shin, Donghoon Yoo, Yongwoo Lee, and Sujoy Sinha Roy. Medha: Microcoded hardware accelerator for computing on encrypted data. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):463–500, 2023.

[RAD78] R L Rivest, L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

[RJV+18] S. Sinha Roy, K. Järvinen, J. Vliegen, F. Vercauteren, and I. Verbauwhede. HEPCloud: An FPGA-based multicore processor for FV somewhat homomorphic function evaluation. *IEEE Transactions on Computers*, 2018.

[RLPD20] M. Sadegh Riazi, Kim Laine, Blake Pelton, and Wei Dai. Heax: An architecture for computing on encrypted data. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 1295–1309, New York, NY, USA, 2020. Association for Computing Machinery.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.

[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.

[Sch] Andreas Schlapka. Micron announces shift in high-performance memory roadmap strategy. https://www.micron.com/about/blog/2018/august/micron-announces-shift-in-high-performance-memory-roadmap-strategy accessed on 14 April, 2023.

[SFK+22] Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. Craterlake: A hardware accelerator for efficient unbounded

computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 173–187, New York, NY, USA, 2022. Association for Computing Machinery.

[TRG+20] Jonathan Takeshita, Dayane Reis, Ting Gong, Michael Niemier, X. Sharon Hu, and Taeho Jung. Algorithmic acceleration of b/fv-like somewhat homomorphic encryption for compute-enabled ram. Cryptology ePrint Archive, Report 2020/1223, 2020. https://ia.cr/2020/1223.

[TRV20] Furkan Turan, Sujoy Sinha Roy, and Ingrid Verbauwhede. Heaws: An accelerator for homomorphic encryption on the amazon aws fpga. *IEEE Transactions on Computers*, 69(8):1185–1196, 2020.

[VJV+19] Naveen Verma, Hongyang Jia, Hossein Valavi, Yinqi Tang, Murat Ozatay, Lung-Yen Chen, Bonan Zhang, and Peter Deaville. In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine*, 11(3):43–55, 2019.