Hardware Acceleration for Fully Homomorphic Encryption Scheme Switching from CKKS to FHEW

Kaiyuan Zhang*, Antian Wang†,Keshab K. Parhi‡, Yingjie Lao*
*Department of Electrical and Computing Engineering, Tufts University, Medford, MA 02155, USA
{kzhang11,yingjie.lao}@tufts.edu
†Department of Electrical and Computer Engineering, Purdue University Fort Wayne, Fort Wayne, IN 46805, USA
antian.wang@pfw.edu
‡Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN 55455, USA
parhi@umn.edu

***Abstract*—Fully Homomorphic Encryption (FHE) presents a paradigm-shifting framework for performing computations on encrypted data, offering revolutionary implications for privacy-preserving technologies. This paper introduces a novel hardware implementation of scheme switching between two leading FHE schemes targeting different computational needs, i.e., arithmetic HE scheme CKKS, and Boolean HE scheme FHEW. The proposed architecture facilitates dynamic switching between the schemes with improved throughput and latency compared to the software baseline. The proposed architecture computation modules support scheme switching operations involving coefficient conversion, modular switching, and key switching. We also optimize the hardware designs for the pre-processing and post-processing blocks, involving key generation, encryption, and decryption. The effectiveness of our proposed design is verified on the Xilinx U280 Datacenter Acceleration FPGA. We demonstrate that the proposed scheme switching accelerator yields a 365× performance improvement over the software counterpart.**

***Index Terms*—Homomorphic Encryption, Scheme Switching, FPGA acceleration**

I. Introduction

The proliferation of data breaches during local devices and cloud servers has drawn the attention of many companies and researchers. This demand has led to an increased reliance on established security techniques to protect sensitive data. Fully Homomorphic Encryption (FHE) is as a highly promising approach, as it allows for secure computations on encrypted data without compromising confidentiality [1].

FHE is a powerful cryptographic technology that performs computations directly on encrypted data without decrypting it. Consequently, FHE ensures that sensitive information remains confidential even when processed by third-party servers, bridging the gap between functionality and privacy [2, 3]. Most FHE schemes are grounded in the Ring Learning with Errors (RLWE) problem, as introduced in [2, 4]. HE can be broadly divided into two categories: arithmetic homomorphic encryption schemes, such as BGV [5], BFV [3] and CKKS [6], which support homomorphic addition and multiplication, and Boolean homomorphic encryption schemes, such as TFHE [7] and FHEW [8]. The ability of HE schemes to perform large-scale computations in parallel without incurring intermediate communication overhead makes them particularly suitable for privacy-preserving machine learning applications [9–12].

The inherent parallelism in CKKS can be exploited to perform linear operations, e.g., convolution operations. In contrast, the FHEW scheme is faster at performing non-linear operations but is expensive when performing linear operations [6, 8, 13]. There are several FHE accelerator papers [14–18], but those only support one FHE scheme. With scheme switching, it is possible to integrate both FHE schemes by enabling the seamless transition between schemes [19–21]. It will be extremely suitable for applications with both linear and nonlinear functions, such as neural networks. Built upon recent works on hardware acceleration of FHE operations [22–25], this work proposes an efficient FPGA-based scheme switching architecture for switching from the CKKS ciphertext to the FHEW ciphertext. We summarize our technical contributions below:

- We design an FPGA-based scheme switching architecture for switching from the CKKS ciphertext to the FHEW ciphertext.
- We develop a low-latency fixed-point multiplication circuit and a specialized scaling unit to accelerate the modulus operations in scheme switching.

The rest of the paper is organized as follows: We review the background of FHE and scheme switching in Section II. In Section III, we present the proposed architecture. We describe the experimental results in Section IV and conclude this paper in Section V.

II. Background

A. Learning with Error and Ring Learning with Error

The Learning with Errors (LWE) problem is a foundational challenge in lattice-based cryptography [4]. It involves solving a system of noisy linear equations, where the added noise makes it computationally difficult to recover the original variables. Its security is based on the hardness of lattice problems, such as finding the shortest vector in a high-dimensional lattice [26]. The RLWE problem is an extension of LWE, formulated over polynomial rings instead of vectors for compact and faster operations.

B. FHE Schemes: CKKS and FHEW

CKKS [6] is designed to support approximate arithmetic over encrypted data, making it highly suitable for real-world applications where exact precision is not always a necessity [27], such as machine learning, signal processing, and statistical analysis. FHEW is a lattice-based scheme that provides a foundation for strong security while allowing efficient Boolean homomorphic operations [8] with an optimized bootstrapping technique in reducing the time complexity of each gate operation. It empowers essential capability for practical applications such as secure multi-party computation and low-latency cloud computing tasks composed of multiple logic gates [28, 29].

1) Encryption Principles for CKKS: CKKS encryption is based on the RLWE problem. The key mathematical concepts are as follows:

- **Plaintext:** m is represented as a polynomial $m(x) \in R_q$, where $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, q is the modulus, and n is the degree.
- **Ciphertext:** $c = (c_0, c_1) \in R_q^2$.

a) Encryption Process: The ciphertext is generated as:

$$c = (c_0, c_1) = (a \cdot s + e_0 + \Delta \cdot m, a), \tag{1}$$

where:

- a is a randomly sampled polynomial,
- s is the secret key,
- e_0 is Gaussian noise,
- Δ is a scaling factor for mapping plaintext into the encryption domain.

b) Decryption Process: Decryption is performed as follows:

$$m' = \frac{c_0 + c_1 \cdot s \mod q}{\Delta}, \tag{2}$$

where m' approximates the plaintext m.

2) Homomorphic Property Validation for CKKS: CKKS supports homomorphic addition and multiplication. For two plaintexts m_1 and m_2, and their corresponding ciphertexts c_1 and c_2:

$$\text{Enc}(m_1)+\text{Enc}(m_2) = (a_1 \cdot s + \Delta \cdot m_1 + e_1) + (a_2 \cdot s + \Delta \cdot m_2 + e_2). \tag{3}$$

Decryption yields:

$$\text{Dec}(\text{Enc}(m_1) + \text{Enc}(m_2)) = m_1 + m_2. \tag{4}$$

For multiplication, ciphertexts are expanded into three components during the operation:

$$\text{Enc}(m_1) \cdot \text{Enc}(m_2) = (c_0 \cdot c_0, c_0 \cdot c_1 + c_1 \cdot c_0, c_1 \cdot c_1). \tag{5}$$

A process called relinearization reduces the ciphertext dimension back to two components. After decryption:

$$\text{Dec}(\text{Enc}(m_1) \cdot \text{Enc}(m_2)) = m_1 \cdot m_2. \tag{6}$$

3) Encryption Principles for FHEW: FHEW encryption is based on the *Ring Learning with Errors (RLWE)* problem. The key mathematical concepts are as follows:

- **Plaintext:** A binary value $m \in \{0, 1\}$, encoded as $m' = m \cdot q/2$, where q is the modulus.
- **Ciphertext:** A pair $c = (a, b) \in \mathbb{T}^n$, where $\mathbb{T}$ represents the torus and:

$$b = \langle a, s \rangle + m \cdot q/2 + e \mod q, \tag{7}$$

with a as a random vector, s as the secret key, and e as Gaussian noise.

a) Encryption Process: The ciphertext is generated as:

$$Enc(m) = (a, b), \quad b = \langle a, s \rangle + m \cdot q/2 + e \mod q. \tag{8}$$

b) Decryption Process: Decryption is performed as:

$$m = \text{round}\left(\frac{b - \langle a, s \rangle}{q/2}\right), \tag{9}$$

recovering $m \in \{0, 1\}$.

4) Homomorphic Property Validation for FHEW: FHEW enables homomorphic evaluation of binary gates (e.g., AND, OR, XOR) using *gate bootstrapping*. Its efficient gate bootstrapping mechanism enables this, which resets noise growth after every gate operation.

- Addition:

$$c_{\text{add}} = (a_1 + a_2, b_1 + b_2) \tag{10}$$

Decryption yields $(m_1 + m_2) \mod q$.

- Multiplication: Corresponds to logical AND:

$$c_{\text{mul}} = \text{Bootstrap}(c_1, c_2), \tag{11}$$

where the bootstrapping mechanism ensures the ciphertext remains decryptable.

C. Scheme Switching

Scheme switching is an emerging technique in FHE that enables transitions between encryption schemes. This technique could efficiently bring the strength of multiple schemes together in applications with diverse types of operations on encrypted data. Scheme switching allows operations on large datasets in BFV and decision-making in TFHE [30] or combining CKKS for polynomial evaluations with CGGI and BFV in machine learning [21]. These scheme switching algorithms are also integrated into open-source library OpenFHE [19]. Inside scheme switching, modulus switching converts the ciphertext modulus from one FHE scheme to another, e.g., the CKKS modulus to the FHEW modulus. In general, the FHEW modulus is smaller than the modulus in CKKS, which could benefit noise management. Key switching allows ciphertext transformation, e.g., transforming a ciphertext encrypted under a CKKS key into a ciphertext encrypted under an FHEW key without exposing the underlying plaintext. This process is crucial for maintaining security while enabling flexibility in computations that involve multiple keys.

III. FHE Scheme Switching Architecture

A. Overall Architecture

We map our design onto an FPGA platform, as shown in Fig. 1. The system mainly consists of four modules: ModReduce, ModSwitch, KeySwitch, and ExtractLWE, with the X86 host CPU managing data loading and deploying the RTL design to the FPGA. The host CPU communicates with the Alveo U280 FPGA via a PCIe interface. Data exchange between the host and FPGA is facilitated through an AXI4-Lite interface.

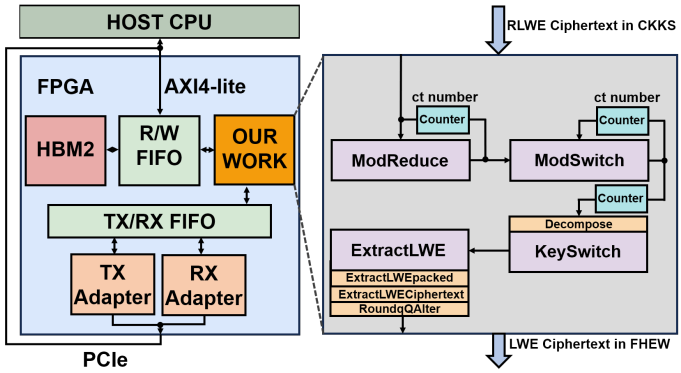

Fig. 1: The overall architecture of the proposed scheme switching accelerator.

B. ModReduce

As shown in Fig. 2, the RLWE ciphertext processed by the CKKS scheme is first input into the ModReduce module. This module consists of several ModReduce modules optimized for low latency. In CKKS, the initial ciphertext is encrypted with a modulus that defines the size of the ciphertext element. Our low-latency design is achieved by configuring each ModReduce submodule to operate in parallel, enabling simultaneous processing of all elements in the CKKS ciphertext. Those submodules go through the FACTOR division steps, each independently handling a portion of the modulus reduction. As homomorphic operations (such as addition and multiplication) are performed, noise accumulates within the ciphertext. By reducing the modulus in parallel, our proposed architecture could manage noise accumulation while maintaining high computational efficiency, thus ensuring that the computation is precise without introducing extra delays.

Scaling unit: Each ModReduce submodule has a specialized scaling unit designed to handle modulus reduction efficiently during the CKKS encryption process. As shown in Fig. 2, by dividing the modulus reduction process across these submodules, our proposed scaling unit can manage each portion of the modulus independently, completing the FACTOR division steps in parallel. This parallelized approach maintains computational accuracy and high throughput, ensuring precise computation without additional delays.

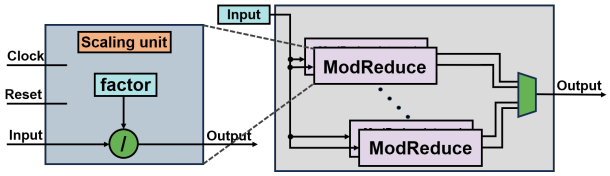

Fig. 2: The ModReduce module.

C. ModSwitch

Low latency design: As shown in Fig. 3, the modulus switching module(ModSwitch) reduces the size of the modulus from a larger modulus Q in CKKS to a smaller modulus in FHEW. To minimize latency, we use a parallelized architecture to implement the ModSwitch, similar to the approach used in ModReduce. By setting the parallel parameter correctly, the ModSwitch could simultaneously deal with the switching operation of all the parts in one CKKS ciphertext. This could significantly reduce the computation time required for modulus switching between CKKS and FHEW schemes, enhancing overall performance.

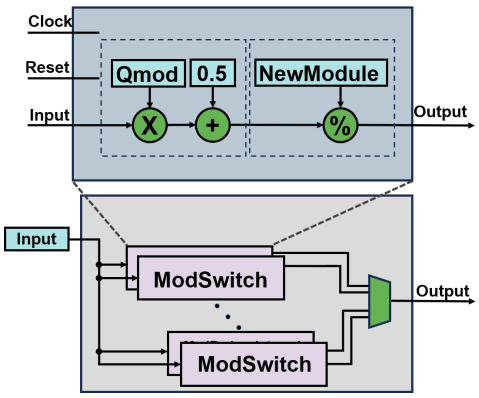

Fig. 3: The ModSwitch module.

D. KeySwitch

As shown in Fig. 4, the key switching module (KeySwitch) transforms a ciphertext encrypted under the CKKS key into a ciphertext encrypted under the FHEW key. In our proposed architecture, the Decompose module performs digit decomposition on a ciphertext to facilitate key switching, following the module switching. This process involves breaking down large coefficients within the ciphertext polynomial into smaller "digits" to efficiently transfer ciphertexts between encryption keys with minimal computational overhead. This design reduces computation time, enhancing the overall performance of the key switching operation.

E. ExtractLWE

This module extracts the ciphertext subresult from the KeySwitch and ModSwitch modules and converts the ciphertext from RLWE to LWE format. The algorithm of this module is based on the ExtractLWE function in the OpenFHE library [19]. The module consists of three functions:

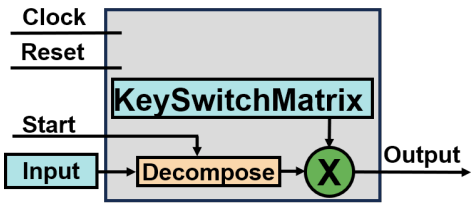

Fig. 4: The KeySwitch module.

ExtractLWEpacked, ExtractLWECiphertext, and RoundqQAlter. ExtractLWEpacked retrieves the necessary components, ExtractLWECiphertext builds the LWE ciphertexts, and RoundqQAlter ensures that the modulus switching and rounding are performed accurately. This module returns the fully processed LWE ciphertexts ready for further cryptographic operations. This process allows seamless extraction and transformation of ciphertexts from CKKS to LWE format, enabling operations across different encryption schemes.

IV. Experimental Results

We implement our scheme switching architecture on the Alveo U280 FPGA using the Xilinx Vivado 2023.1 and Vitis 2023.1 EDA design tools. The software implementation of scheme switching is derived from the OpenFHE library [19], which provides a high-level interface for lattice-based cryptography. All software baseline implementations are executed on an i7-14900K CPU, with the codebase written in C++. The FPGA design consists of multiple functional units, including URAMs, BRAMs, FIFOs, and control logic. Our RTL design implements 32 memory-mapped 256-bit AXI4 master interfaces, which enable efficient bidirectional data transfers between the FPGA and its global memory. The FPGA has two HBM2 stacks, each with a 4 GB capacity and up to 460 GB/s of memory bandwidth. Data is streamed between the global and on-chip memory using RD and WR FIFOs to ensure high throughput and efficient utilization of the available memory bandwidth.

As shown in Table I, our ModSwitch module on FPGA achieves a $332\times$ speedup, and the KeySwitch module achieves a $6.8\times$ speedup when compared to their software implementations in C++. Overall, the proposed scheme switching accelerator yields a $365\times$ performance improvement over the software counterpart. This speedup is particularly significant, considering the design focuses exclusively on optimizing scheme-switching operations, while the test cases used only simple ciphertexts.

In addition, Table II and Table III present the FPGA resource utilization and power consumption metrics, respectively. These results provide insight into the hardware efficiency of our FPGA-based implementation.

V. Conclusion

This paper presents an FPGA-based architecture that converts CKKS ciphertexts to FHEW ciphertexts. The design includes a low-latency fixed-point multiplication module and a specialized scaling module. The design significantly accelerates the modulus switching operations by achieving a $365\times$ overall speedup compared to its software implementation.

TABLE I: HW and SW time consumption

Function	Time C++(ns)	Time FPGA(ns)	Speedup
ModSwitch	5642	17	332×
Decompose	2258	50	45×
KeySwitch	342	50	6.8×
ExtrackLWEpacked	1289	20	64×
ExtractLWECiphertext	6034	17	355×
RoundqQAlter	5225	23	227×
Overall	76629	210	365×

TABLE II: Resource Utilized

Resource	LUTs	FFs	DSPs	BRAM
Utilized	136044	170944	55	205

TABLE III: Power Consumption

Power(W)	FPGA Power(W)	HBM Power(W)
38.59	28.26	9.97

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation (NSF) under grant numbers CCF-2243053, CCF-2412357, and SaTC-2426299.

References

[1] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[2] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of the ACM (JACM)*, vol. 60, no. 6, pp. 1–35, 2013.

[3] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[4] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2010, pp. 1–23.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *23rd International Conference on the Theory and Applications of Cryptology and Information Security*, 2017, pp. 409–437.

[7] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[8] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Annual*

international conference on the theory and applications of cryptographic techniques, 2015, pp. 617–640.

[9] Z. Ghodsi, A. K. Veldanda, B. Reagen, and S. Garg, "Cryptonas: Private inference on a ReLU budget," *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 961–16 971, 2020.

[10] H. Peng, S. Zhou, Y. Luo, N. Xu, S. Duan, R. Ran, J. Zhao, S. Huang, X. Xie, C. Wang *et al.*, "RRnet: Towards ReLU-reduced neural network for two-party computation based private inference," *arXiv preprint arXiv:2302.02292*, 2023.

[11] A. Wang and Y. Lao, "Homomorphic evaluation friendly vision transformer design," in *2023 57th Asilomar Conference on Signals, Systems, and Computers*, 2023, pp. 7–10.

[12] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography–SAC 2018: 25th International Conference, Calgary, AB, Canada, August 15–17, 2018, Revised Selected Papers 25*. Springer, 2019, pp. 347–368.

[13] S. Bian, Z. Zhang, H. Pan, R. Mao, Z. Zhao, Y. Jin, and Z. Guan, "He3db: An efficient and elastic encrypted database via arithmetic-and-logic fully homomorphic encryption," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2930–2944.

[14] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 882–895.

[15] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "SHARP: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[16] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.

[17] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez, "F1: A fast and programmable accelerator for fully homomorphic encryption," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 238–252.

[18] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 173–187.

[19] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee *et al.*, "OpenFHE: Open-source fully homomorphic encryption library," in *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*, 2022, pp. 53–63.

[20] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient homomorphic conversion between (Ring) lwe ciphertexts," in *International Conference on Applied Cryptography and Network Security*. Springer, 2021, pp. 460–479.

[21] W.-j. Lu, Z. Huang, C. Hong, Y. Ma, and H. Qu, "PEGASUS: bridging polynomial and non-polynomial evaluations in homomorphic encryption," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1057–1073.

[22] W. Tan, B. M. Case, A. Wang, S. Gao, and Y. Lao, "High-speed modular multiplier for lattice-based cryptosystems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 8, pp. 2927–2931, 2021.

[23] W. Tan, B. M. Case, G. Hu, S. Gao, and Y. Lao, "An ultra-highly parallel polynomial multiplier for the bootstrapping algorithm in a fully homomorphic encryption scheme," *Journal of Signal Processing Systems*, vol. 93, pp. 643–656, 2021.

[24] W. Tan, A. Wang, X. Zhang, Y. Lao, and K. K. Parhi, "High-speed VLSI architectures for modular polynomial multiplication via fast filtering and applications to lattice-based cryptography," *IEEE Transactions on Computers*, vol. 72, no. 9, pp. 2454–2466, 2023.

[25] W. Tan, S.-W. Chiu, A. Wang, Y. Lao, and K. K. Parhi, "PaReNTT: Low-latency parallel residue number system and ntt-based long polynomial modular multiplication for homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, 2023.

[26] S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan, "Robustness of the learning with errors assumption," 2010.

[27] P.-E. Clet, O. Stan, and M. Zuber, "BFV, CKKS, TFHE: Which one is the best for a secure neural network evaluation in the cloud?" in *Applied Cryptography and Network Security Workshops*, 2021, pp. 279–300.

[28] Q. Lou, B. Feng, G. Charles Fox, and L. Jiang, "Glyph: Fast and accurately training deep neural networks on encrypted data," *Advances in neural information processing systems*, vol. 33, pp. 9193–9202, 2020.

[29] D. Micciancio and Y. Polyakov, "Bootstrapping in FHEW-like cryptosystems," in *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2021, pp. 17–28.

[30] C. Boura, N. Gama, M. Georgieva, and D. Jetchev, "CHIMERA: Combining Ring-LWE-based fully homomorphic encryption schemes," *Journal of Mathematical Cryptology*, vol. 14, no. 1, pp. 316–338, 2020.