PolyFHEmus: Rethinking Multiplication in Fully Homomorphic Encryption

Charles Gouert and Nektarios Georgios Tsoutsos

University of Delaware
{cgouert, tsoutsos}@udel.edu

Abstract. Homomorphic encryption (HE) is a powerful technology that solves key privacy concerns in cloud computing by enabling computation on encrypted data. However, it has not seen widespread adoption due to high latencies resulting from extensive operations over high-degree polynomials with large coefficients. In this paper, we identify polynomial multiplication as a bottleneck and investigate alternative algorithms to accelerate encrypted computing. Most popular open-source HE implementations utilize the NTT or FFT algorithms to perform point-wise multiplication in the Fourier domain; on the other hand, we investigate applying alternative strategies that decompose the problem of multiplying polynomials to that of multiplying large integers. We combine the Schönhage–Strassen (SS) algorithm for integer multiplication and the discrete Galois transform (DGT) to achieve faster ciphertext multiplication speeds for large polynomial degrees. For a degree-5 polynomial approximation of the nonlinear Sigmoid function utilized in logistic regression, we observe that this alternative approach can result in a speedup of approximately 1.3× relative to the Microsoft SEAL library for the CKKS scheme. Notably, the proposed approach can also apply to other HE schemes that utilize large polynomial degrees, such as the BFV and BGV cryptosystems.

Keywords: Homomorphic encryption · Hardware acceleration · Polynomial multiplication · Secure computing.

1 Introduction

Cloud computing has emerged as a ubiquitous paradigm that allows companies to leverage powerful remote servers to perform computational tasks like analytics and classification. Indeed, this allows companies to pay for computational resources on demand instead of maintaining costly local infrastucture to perform important tasks. Unfortunately, cloud computing clients yield control of their data to the cloud service provider who owns the remote server. A curious service provider can view customer data stored on their own servers provided no confidentiality mechanisms are used to protect the data. Standard encryption techniques, such as the AES encryption scheme, can be used to accomplish this task and shield sensitive data from other parties, but it also renders the data unusable by the cloud. In essence, this strategy only allows for outsourced storage, but not meaningful computation.

A unique form of encryption called *homomorphic encryption* (HE) can simultaneously solve the problem of data privacy in the cloud while also enabling arbitrary computation over ciphertext data (i.e., the encrypted data) [14]. With modern HE schemes, users can encode their data as vectors of integers or floating point numbers into a single ciphertext [6,9,12]. HE ciphertexts typically take the form of tuples of high-degree polynomials with large coefficients and operations between ciphertexts are composed of polynomial arithmetic. As a result, operations on encrypted data are significantly slower than computing on cleartext data (i.e., unprotected data in its original and natural form). State-of-the-art implementations of HE schemes leverage hardware acceleration in the form of ASICs (application-specific integrated circuits) [23, 24], FPGAs [1, 27] and GPUs [15, 17] to reduce the latency and improve the throughput of encrypted operations. Even still, encrypted computation remains orders of magnitude slower than equivalent operations on cleartext data.

In the following sections, we identify polynomial multiplication as a key bottleneck in HE operations and examine alternative methodologies to replace widespread techniques utilized across the board in popular open-source implementations of homomorphic encryption schemes. Specifically, we observe that we can



Fig. 1. Secure Outsourcing with HE: The client encrypts sensitive data and then transmits the ciphertext over a network to the remote cloud server. The cloud server can then execute an algorithm over the ciphertexts and generate an encrypted answer, which is transmitted back to the client. Lastly, the client can decrypt and receive the result of the computation. No information about the inputs or outputs is revealed to the cloud.

decompose the problem of polynomial multiplication in HE to the problem of multiplying two large integers. We find that the Schönhage–Strassen algorithm [25] is well suited for multiplying very large integers with thousands of digits and further observe that we can optimize the technique further by substituting the fast fourier transform with the discrete Galois transform. We compare our approach against the Microsoft SEAL HE library [26] and investigate the comparative performance of our techniques across several parameter sets that align with current use-cases of HE in the research community. Relative to SEAL, we observe speedups of 1.3× for a representative ML-based workload. Overall, this work aims to accomplish three goals:

- Analyze low-level HE primitive operations to definitively outline current performance bottlenecks;
- Propose a strategy that adopts Kronecker substitution and the Schönhage–Strassen integer multiplication algorithm to enable faster polynomial multiplication in HE;
- Demonstrate meaningful speedups of HE primitives relative to the state-of-the-art.

2 Homomorphic Encryption Primer

All types of HE incorporate *malleable* ciphertexts that can be modified by computation to affect the underlying message of the ciphertext. However, not all HE schemes exhibit the same capabilities in terms of the number and types of operations that can be computed on the encrypted data. In general, all HE implementations can be delegated into one of three categories (depicted in Figure 2: partial HE (PHE), leveled HE (LHE), and fully HE (FHE). PHE will not be a focus of this paper as it is not functionally complete and therefore suited for a limited number of applications.

The next form of HE, called LHE, allows for both addition and multiplication in the encrypted domain. These capabilities constitute a functionally complete set of operations, meaning that any algorithm can be constructed with these arithmetic primitives. Ciphertexts take the form of tuples of high-degree polynomials and unlike PHE algorithms, most LHE schemes derive their security from a hard mathematical problem known as learning with errors (LWE) [22]. This problem is akin to solving a series of equations that boils down to finding s for all i instances: $b_i = a_i \cdot s_i + e_i$, where b_i and a_i are known and e_i is a small random error. At a high-level, plaintext values are converted to polynomials upon encryption and small errors are added to the coefficients. An unfortunate consequence of this is that the noise compounds as operations are conducted on the encrypted data. Specifically, ciphertext addition (which is composed of addition between the polynomials of the first ciphertext with the corresponding polynomials of the second ciphertext) causes the noise to grow linearly while multiplication causes exponential growth [6]. Eventually, the noise will exceed a certain threshold after which it will begin corrupting the underlying message. Increasing the degree of the polynomials and growing the coefficient size will yield a larger noise threshold and therefore allow for more operations. For applications that have a large depth, meaning that the number of subsequent multiplication operations over a ciphertext is high, LHE exhibits poor scalability as ciphertexts become unmanageably large and operations become more computationally intensive.

The third and most computationally powerful form of HE is fully homomorphic encryption (FHE), which allows for unbounded addition and multiplication via a mechanism called *bootstrapping* that is capable of refreshing the noise and can be invoked an unlimited number of times (unlike modulus switching). In fact, any

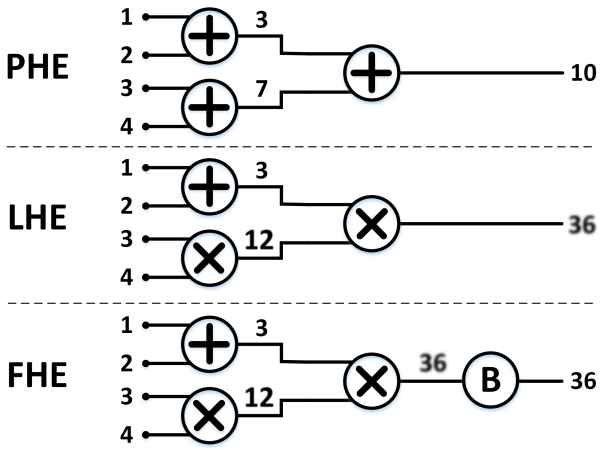


Fig. 2. Types of Homomorphic Encryption: PHE can only perform one type of operation, but is noiseless. On the other hand, LHE can perform both multiplication and addition, but accumulates noise and eventually the message becomes corrupted. Lastly, FHE has the same capabilities of LHE, but can refresh the ciphertext noise with a bootstrapping operation ("B" in the diagram).

LHE construction can become FHE with the introduction of this mechanism. While the relative latency of bootstrapping is significantly higher compared to all other HE operations, FHE evaluation exhibits superior scalability compared to LHE for applications that exhibit a very large multiplicative depth, such as neural networks with several layers. In effect, FHE mitigates the scalability problem as a single parameter set that supports bootstrapping can evaluate arbitrarily deep programs.

In this work, we compare against the popular Microsoft SEAL homomorphic encryption library [26], which supports strictly LHE contexts. Specifically, SEAL supports the BGV [6], BFV [12], and CKKS [9] cryptosystems. Both BGV and BFV are used to encrypt vectors of integers, while CKKS encrypts vectors of complex or floating point numbers. Even though SEAL only supports LHE contexts, we remark that our proposed techniques outlined in the following sections are equally applicable to FHE, as the core primitive operations are similar and consist primarily of large polynomial operations.

As a case in point, the bootstrapping procedure of CKKS requires evaluating linear transformations homomorphically, which can be viewed as large matrix-vector products. Evaluating this linear algebra routine in the encrypted domain requires a large number of multiplications between ciphertexts and plaintexts, where the plaintexts are represented as polynomials with the same large degree as the ciphertext data [7]. Additionally, the CKKS decryption circuit contains a modular reduction, which is difficult to evaluate homomorphically; instead, a high-degree polynomial approximation of the sine function (which mimics modular reduction in a specific range) is employed to approximate the mod function [8]. Prior work has shown that careful selection of the plaintext modulus in BFV allows us to replace the digit extraction-based bootstrapping technique with the CKKS strategy [18] by adding of a few extra operations. As a result, both BFV and CKKS bootstrapping can rely a polynomial approximation of the modular reduction, while BGV bootstrapping relies on a polynomial evaluation procedure to accomplish the digit extraction step [16]. Both bootstrapping strategies are multiplication-intensive as they involve polynomial approximations, as well as linear transformations, and therefore can benefit from fast polynomial multiplication methods.

Threat Model: Similarly to other works that rely solely on homomorphic encryption primitives, we assume that the computing party (i.e., the remote server owned by the cloud service provider) is *honest-but-curious*. This means that the cloud provider will faithfully execute the desired algorithm on the encrypted data, but has an incentive to view the sensitive client information being manipulated. In this scenario, homomorphic encryption instantiated with the proper, secure parameters ensures complete data confidentiality of inputs, intermediate results, and outputs. The only information that the cloud can glean is potentially the relative sizes of the plaintext of the encrypted data. As a simple example, assuming that a non-batching context is

used and only scalars are encrypted, the cloud can infer that the plaintext consists of N integers or floating point numbers if the client sends N ciphertexts as input.

3 Faster Polynomial Arithmetic for HE Operations

We begin our investigation by profiling the Microsoft SEAL BFV implementation for a degree-2 polynomial approximation of the ReLU function [3], which is widely used in machine learning applications. consisting of a ciphertext multiplication and a ciphertext addition. We find that approximately 40% of the evaluation time consists of the ciphertext multiplication, while the addition constitutes a negligible percentage of the overall runtime (i.e., $< 1\%$). The remaining overheads consist primarily of keyswitching, which is a crucial ciphertext maintenance procedure and also includes polynomial multiplication operations. The fast speed of the addition is due to the fact that the only operations involved include a set of element-wise modular additions across each pair of ciphertext polynomials. In fact, the modular reduction in this case is simplified relative to the reduction employed by the multiplication. By taking advantage of the fact that the sum will be less than twice the modulus, SEAL performs a simple check to see if the sum is greater than the modulus and subtracts the modulus if it is (otherwise no reduction is required). Additionally, the element-wise modular additions are embarrassingly parallel, which can be exploited to achieve very fast speeds. The remaining runtime percentage of the program consists primarily of generating keys, which we note is a one-time cost (the keys can be re-used for future HE programs assuming the same parameter set can be utilized).

3.1 Current Strategies for Polynomial Multiplication

The textbook method of multiplying polynomials is the most straightforward algorithm, but exhibits the worst asymptotic performance ($\mathcal{O}(N^2)$) as each coefficient of the first polynomial is multiplied with every coefficient of the second polynomial. Considering that homomorphic ciphertext polynomials can plausibly contain up to 2^{17} coefficients, the poor scalability of this approach results in extremely significant performance penalties. Instead, most HE libraries use an alternative approach with better scalability for large polynomial degrees based on the discrete Fourier transform (DFT).

This technique involves using a forward transform to convert the polynomials to the Fourier domain, where a polynomial multiplication translates to a point-wise multiplication between the coefficients. Ciphertexts are typically kept in this Fourier representation until an operation needs to be conducted in the original coefficient representation (e.g., for certain steps of keyswitching). Overall, this results in an asymptotic complexity of $\mathcal{O}(N \log N)$, which significantly outperforms the textbook, naive approach for the large polynomial degrees required for secure HE evaluation. To execute the DFT, two of the more widely used options are the fast Fourier transform (FFT) and number theoretic transform (NTT). Among these, most HE libraries opt for the NTT as it works naturally over the integers, whereas the FFT requires the coefficients of the ciphertext polynomials to be mapped to the floating point domain. Working in the floating point domain also has the negative side effect of rounding errors, which can exacerbate noise growth during computation. With these configurations, the core bottleneck of computation is the DFT; to give additional context about the scale of this problem, a dot product between two encrypted vectors of length 500 in the CGGI FHE cryptosystem results in over a billion NTT invocations [15].

3.2 Computing Polynomial Multiplication using the Discrete Galois Transform

An alternative to the FFT and NTT for computing the DFT is the discrete Galois transform (DGT). This transform can offer significant advantages over the other DFT techniques as it allows us to cut down the overall transform size to $N/2$ instead of N. Additionally, the DGT has been observed to have lower memory bandwidth requirements and higher arithmetic intensity in the context of GPU acceleration relative to state-of-the-art NTT and FFT constructions [4]. Because of the smaller transform size, the DGT can be implemented with smaller hardware circuits [20]. Similarly to the NTT, the DGT operates directly over integers, which is more natural in the context of FHE where the ciphertext coefficients are modular integers.

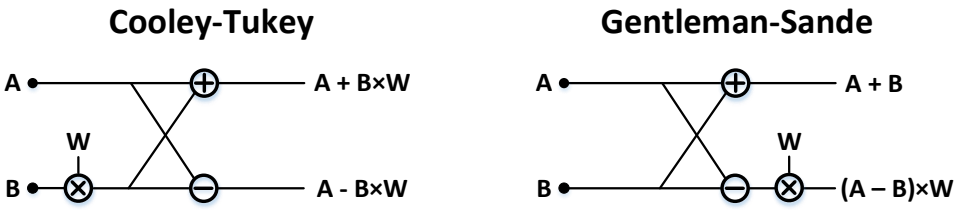


Fig. 3. A visual depiction of the two most popular DFT "butterflies": the Cooley-Tukey and Gentleman-Sande constructions. These are referred to as butterflies due to the shape created by the intersecting paths from the top and bottom wires. Both of these illustrate a radix-2 butterfly, which can be is used as the core primitive block of larger DFTs. A and B are both data inputs and W represents a twiddle factor, which is a multiplicative constant used in DFT computations.

Furthermore, like the NTT, the DGT introduces no additional error because its computation is exact and avoids the accumulation of floating point errors seen in the FFT. However, the key difference is that the NTT operates in the finite field $\mathbb{F}_p$, where p is a constituent prime of the ciphertext modulus. On the other hand, the DGT operates in $\mathbb{F}_{p^2}$ and works over Gaussian integers, which contain real and imaginary parts in $\mathbb{Z}_p$ (i.e., the set of integers modulo p). In this case, we have that addition and subtraction of two Gaussian integers involve adding or subtracting the real parts and the imaginary parts independently, which can be done in parallel. For Gaussian integers $x = a + bi$ and $y = c + di$, the product of the two is defined as $z = (a \cdot c - b \cdot d) + (a \cdot d + b \cdot c)i$.

The DGT can utilize the same data paths as the FFT or NTT, of which there are two key variations: Cooley-Tukey [10] and Gentleman-Sande [13] butterflies, which are depicted in Figure 3. Both algorithms are similar and utilize a constant known as a "twiddle factor". Twiddle factors are always complex roots of unity, where the nth root of unity is defined as $W_n = e^{-2\pi \mathrm{i}/N}$. We note that the twiddle factors used in the computation of the DFT can be pre-generated prior to any HE evaluation, but it is also possible to compute these on-the-fly [28]; the core benefit of this is reducing the memory required to store these factors, however it also increases the computational overhead of the transform at runtime. Prior works have additionally proposed hybrid approaches that compute some proportion of the twiddle factors on-the-fly and others in advance [19] to strike a balance between storage requirements and added computational overhead. For the purposes of our experimental evaluation (Section 4), we opt to pre-compute these factors to minimize the latency for evaluation on the CPU; an additional benefit of the DGT versus the NTT or FFT is that only half of the twiddle factors need to be stored or computed as the size of the transform is reduced by half and the number of twiddle factors needed corresponds to $N - 1$, where N is the transform size. The Cooley-Tukey construction multiplies the second input (i.e., B in the figure, which represents a ciphertext polynomial coefficient) by the twiddle factor before the sum and difference are computed. On the other hand, the Gentleman-Sande butterfly performs this multiplication after the difference of the coefficients A and B are computed. In practice, the Cooley-Tukey butterfly can be used for the forward transform while the Gentleman-Sande butterfly achieves the inverse transform [5, 21].

3.3 Schönhage–Strassen: Fast Multiplication for Big Integers

There have been many proposed algorithms for multiplication of large integers (i.e., hundreds or thousands of digits long). Similarly to polynomial multiplication, the computational complexity of multiplying two integers of n digits is $\mathcal{O}(n^2)$. The first approach exhibiting better asymptotic complexity was Karatsuba in 1960, which achieves a complexity of $\mathcal{O}(n^{\log_2 3})$. Shortly after, the Toom-Cook algorithm generalized Karatsuba and achieved a computational complexity of $\mathcal{O}(n^{\log_k(2k-1)})$, where k is a parameter that refers to the number of chunks the input integers are broken up into. We note that $k = 2$ corresponds exactly to the Karatsuba algorithm, with the most popular variant being $k = 3$ and yielding an overall complexity of approximately $\mathcal{O}(n^{1.465})$. While both approaches outperform the standard integer multiplication

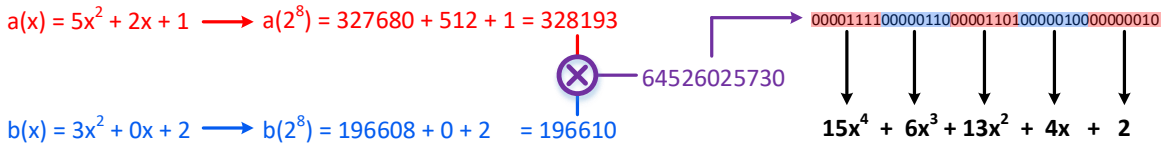


Fig. 4. Kronecker Substitution: Any polynomial can be converted to an integer representation by evaluating it at a power of 2 that is larger than any coefficient. After both input polynomials are converted, they can be multiplied in the integer domain. Then, the coefficients of the product polynomial can be derived directly from the binary representation of the product of the integer multiplication.

algorithm, better techniques have emerged for extremely large integers with several thousand digits. The Schönhage–Strassen (SS) algorithm, introduced in 1971, utilizes the FFT recursively and achieves a complexity of $\mathcal{O}(n \times \log n \times \log(\log n))$, which is more efficient than both Karatsuba and Toom-Cook for a very large number of digits.

An immediate question is how the SS algorithm may be used in the context of polynomial multiplication, since the coefficients in HE ciphertexts are rarely larger than 2300 bits in length [17]. Our key observation is that we can reduce the problem of multiplying two polynomials to the problem of multiplying two integers by leveraging the *Kronecker substitution* technique to converts between polynomials and integers. A simple example is depicted in Figure 4, which demonstrates multiplying the polynomial $5x^2 + 2x + 1$ with $3x^2 + 2$. Both polynomials are converted to the integer domain by evaluating at $x = 2^8$ and then multiplied. Finally, the expected product can be recovered by mapping chunks of 8 bits to the corresponding coefficient. As a case in point, the most significant byte of the integer product is equal to `00001111`, which maps to the polynomial term $15x^4$. With this approach, we can convert ciphertext polynomials to large integers to compute the encrypted multiplication and then convert back to the original polynomial domain (which is necessary for downstream operators like keyswitching).

3.4 Our Contribution: DGT using SS

Recall that the original Schönhage–Strassen utilizes the FFT construction to compute the integer multiplication; however, the FFT can easily be swapped out for another transform that can accomplish the general DFT. Indeed, Doroz et al. [11] proposed an NTT-based SS integer multiplier for a fixed, small RLWE parameter set; however, the DGT only requires a transform of half the size relative to both the NTT and FFT, and allows us to achieve greater scalability for larger ring dimensions. Next, we note that there are two primary methods to compute the polynomial multiplication. The first involves computing the SS+DGT algorithm across the ciphertext polynomials with very large coefficients (i.e., hundreds or thousands of bits in length). However, we observe that the forward Kronecker substitution, which involves evaluating the input polynomials with extremely large powers of 2, is quite costly. Even though each term of the polynomial can be shifted in parallel, summing all of the huge shifted coefficients becomes the bottleneck.

Instead, we can exploit more parallelism by adopting the residue-number system (RNS) to break up the original ciphertext polynomials into several polynomials with significantly smaller coefficients. After evaluation, these smaller polynomials can be combined to regenerate a single polynomial with large coefficients; this is made possible by exploiting the structure of the coefficient modulus, which is a product of primes: $q = p_0 \times p_1 \ldots \times p_N$, where q is the ciphertext modulus and p_i is a prime. Each constituent prime is typically chosen to be between 40 and 64 bits in length and the RNS decomposition will give us N polynomials, where N is the number of prime factors of the modulus. Now, the ith RNS polynomial will consist of coefficients modulo the prime p_i instead of q, allowing us to work with coefficients that are more naturally suited for the native word sizes supported by modern computers. Another key benefit of our proposed strategy is that it enables us to compute the SS+DGT across each polynomial with smaller coefficients in parallel, which can be recombined later to recover the original polynomial representation. Indeed, a similar approach is adopted by most modern HE libraries such as Microsoft SEAL [26] and OpenFHE [2].

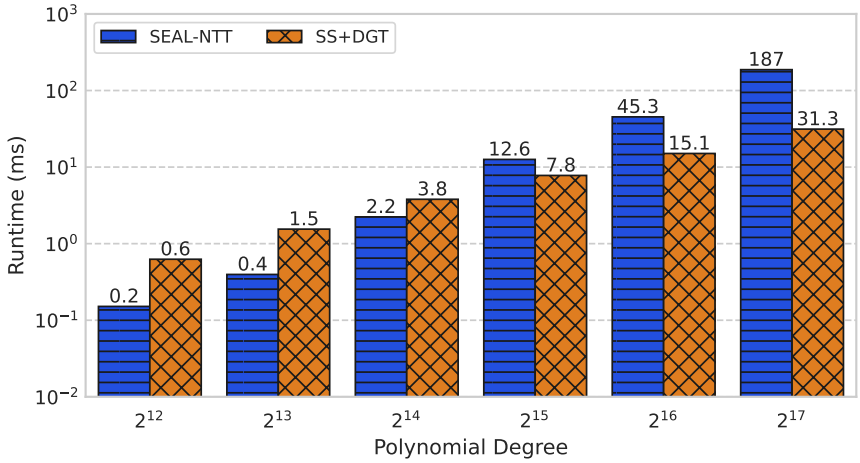


Fig. 5. Encrypted Multiplication Latency: This graph outlines the average latency of performing an encrypted multiplication across two "fresh" ciphertexts (i.e., ciphertexts that have not yet been computed upon). Notably, we strictly measure the multiplication step for both SEAL and our proposed approach. While we allow both programs to use as many threads as necessary, we remark that the SEAL library only utilizes a single thread.

4 Experimental Results

Our approach is implemented as a multi-threaded C++ library that provides an interface for converting to and from polynomials (represented as a vector of integers) and arbitrary precision integers via Kronecker substitution, as well as a multiplication mechanism that operates across two integers of arbitrary precision encoded as arrays of digits. We compare our approach with the current polynomial multiplication strategy implemented in the Microsoft SEAL library, which utilizes the number theoretic transform (NTT) to allow for pointwise multiplication. To facilitate a direct comparison, we target the popular CKKS cryptosystem and replace the multiplication operation with invocations to our library; since ciphertexts in the BGV and BFV cryptosystems have the same structure (i.e., tuples of high degree polynomials), the SS+DGT technique can apply to these schemes as well. All experiments were ran on an `r5.24xlarge` AWS server with 784 GB of RAM and 96 vCPUs across parameter sets corresponding to 128 bits of security under current attack models. Our smallest parameter set utilizes a polynomial degree of 2^{12}, which allows for approximately 1-2 multiplications. We remark that smaller parameters are possible, but these are rarely used for the BGV, BFV, and CKKS cryptosystems. As the polynomial degree doubles, we also double the coefficient modulus, resulting in ciphertexts that are $4\times$ larger. Therefore, the ciphertexts generated using our largest parameters with a polynomial degree of 2^{17} are $1024\times$ larger than our smallest parameters. These parameter sets run the entire gamut of plausible parameter sizes that are in use today for a broad range of use cases. The results of our evaluations are depicted in Figure 5.

We observe that for smaller polynomial degrees SEAL outperforms our SS+DGT approach by a factor of $3\times$ for 2^{12}-degree polynomials and nearly $4\times$ for 2^{13}-degree polynomials. This confirms our expectation, as the SS multiplication is well-suited for very large inputs and performs sub-optimally for smaller inputs. Therefore, for applications that only exhibit a low multiplicative depth (and can be executed correctly with small parameters) the NTT approach is a better choice, and applications such as matrix multiplication, logistic regression inference, and facial recognition via the squared Euclidean distance fall under this category. On the other hand, the SS+DGT multiplication begins to outperform the baseline SEAL approach when the polynomial degree grows over 2^{15} and offers significantly better scalability for larger sizes.

Additionally, we perform experiments with a degree-5 polynomial approximation of the sigmoid function, which forms an integral part of machine learning constructions such as logistic regression. For a polynomial degree of 2^{14}, which is the smallest parameter set that can evaluate the polynomial, we observe that the SEAL library incurs a latency of 120 milliseconds; on the other hand, if the NTT-based multiplication

is swapped out for the SS+DGT approach, we observe a latency of approximately 129 milliseconds. This slowdown is expected, as the SS+DGT approach scales better for larger polynomial degrees. With a degree of 2^{17}, however, we observe that the encrypted polynomial evaluation is approximately 1.3× faster using our SS+DGT multiplication relative to the baseline. Indeed, in the context of a larger program, like a neural network inference with several layers where non-linear activation functions are evaluated by low-degree polynomials, ciphertexts with high degree polynomials are required to accommodate the noise budget. Overall, we remark that our proposed SS+DGT approach offers a notable improvement over the NTT-based polynomial multiplication for complex applications that either require bootstrapping or otherwise exhibit a large depth (for example, deep neural networks, encrypted sorting, and image processing).

5 Concluding Remarks

In this paper, we propose a new strategy for evaluating homomorphic multiplications. As this operation forms a key bottleneck of current schemes, optimizing it has large ramifications for the overall performance of encrypted applications. While many current HE libraries utilize the NTT to facilitate polynomial multiplication, we incorporate the Schönhage–Strassen integer multiplication algorithm and combine this approach with the DGT using Kronecker substitution. We report that our approach outperforms the popular Microsoft SEAL library for large polynomial degrees commonly used for general computation with homomorphic encryption.

Acknowledgments

This work has been supported by NSF Award #2239334.

References

1. Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption. In *2023 IEEE International symposium on high-performance computer architecture (HPCA)*, pages 882–895. IEEE, 2023.
2. Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, et al. Openfhe: Open-source fully homomorphic encryption library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 53–63, 2022.
3. Ramy E Ali, Jinhyun So, and A Salman Avestimehr. On polynomial approximations for privacy-preserving and verifiable ReLU networks. *arXiv preprint arXiv:2011.05530*, 2020.
4. Pedro Geraldo MR Alves, Jheyne N Ortiz, and Diego F Aranha. Faster homomorphic encryption over gpgpus via hierarchical dgt. In *International Conference on Financial Cryptography and Data Security*, pages 520–540. Springer, 2021.
5. Jonas Bertels, Michiel Van Beirendonck, Furkan Turan, and Ingrid Verbauwhede. Hardware acceleration of fhew. In *2023 26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 57–60. IEEE, 2023.
6. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.
7. Jung Hee Cheon, Hyeongmin Choe, Minsik Kang, and Jaehyung Kim. Grafting: Complementing RNS in CKKS. *Cryptology ePrint Archive*, 2024.
8. Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part I 37*, pages 360–384. Springer, 2018.

9. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *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 23*, pages 409–437. Springer, 2017.
10. James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965.
11. Yarkın Doröz, Erdinç Öztürk, and Berk Sunar. Accelerating fully homomorphic encryption in hardware. *IEEE Transactions on Computers*, 64(6):1509–1521, 2014.
12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.
13. W Morven Gentleman and Gordon Sande. Fast Fourier transforms: for fun and profit. In *Proceedings of the November 7-10, 1966, fall joint computer conference*, pages 563–578, 1966.
14. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.
15. Charles Gouert, Vinu Joseph, Steven Dalton, Cedric Augonnet, Michael Garland, and Nektarios Georgios Tsoutsos. Arctyrex: Accelerated encrypted execution of general-purpose applications. *arXiv preprint arXiv:2306.11006*, 2023.
16. Shai Halevi and Victor Shoup. Bootstrapping for helib. *Journal of Cryptology*, 34(1):7, 2021.
17. 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*, pages 114–148, 2021.
18. Jaehyung Kim, Jinyeong Seo, and Yongsoo Song. Simpler and faster BFV bootstrapping for arbitrary plaintext modulus from CKKS. Cryptology ePrint Archive, Paper 2024/109, 2024.
19. Sangpyo Kim, Wonkyung Jung, Jaiyoung Park, and Jung Ho Ahn. Accelerating number theoretic transformations for bootstrappable homomorphic encryption on gpus. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 264–275. IEEE, 2020.
20. Guangyan Li, Donglong Chen, Gaoyu Mao, Wangchen Dai, Abdurrashid Ibrahim Sanka, and Ray CC Cheung. Algorithm-hardware co-design of split-radix discrete galois transformation for kyberkem. *IEEE Transactions on Emerging Topics in Computing*, 2023.
21. Suraj Mandal and Debapriya Basu Roy. KiD: A Hardware Design Framework Targeting Unified NTT Multiplication for CRYSTALS-Kyber and CRYSTALS-Dilithium on FPGA. In *2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID)*, pages 455–460. IEEE, 2024.
22. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.
23. Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Srinivas Devadas, Ronald Dreslinski, Christopher Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 238–252, 2021.
24. 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*, pages 173–187, 2022.
25. Arnold Schönhage and Volker Strassen. Fast multiplication of large numbers. *Computing*, 7:281–292, 1971.
26. Microsoft SEAL (release 4.1). `https://github.com/Microsoft/SEAL`, January 2023. Microsoft Research, Redmond, WA.
27. 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.
28. Cheng Wang and Mingyu Gao. Sam: A scalable accelerator for number theoretic transform using multi-dimensional decomposition. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.