DUALITY TECHNOLOGIES / TECHNICAL WHITE PAPER

Hardware Acceleration of Fully Homomorphic Encryption: Making Privacy-Preserving Machine Learning Practical

Ahmad Al Badawi, David Bruce Cousins, Yuriy Polyakov, Kurt Rohloff

Accelerating fully homomorphic encryption to make privacy-preserving machine learning practical at scale.

1. Executive Summary

The Problem: Protecting Sensitive Data

Organizations are increasingly collaborating on data, often using the cloud, to address some of the most important challenges of our time. Protecting privacy is crucial, particularly when handling sensitive data such as personally identifiable information (PII), personal health information (PHI), intellectual property, and intelligence insights.

> Data has three basic states: at rest, in transit, and in use. Typically, sensitive data is encrypted or otherwise protected while being stored (at rest) and transmitted (in transit). However, when the data is being processed in any way (in use), it must first be decrypted–making it vulnerable to cyberattacks.

Fully Homomorphic Encryption

Often described as the "holy grail" of encryption technologies, Fully Homomorphic Encryption (FHE) enables arbitrary computations over encrypted data without decrypting it at all, potentially solving the data-in-use problem.

FHE for Machine Learning

One of the most promising application domains for FHE is Machine learning (ML). ML has advanced rapidly in recent years, with the number of applications multiplying to include medicine, finance, natural language processing, and more. ML often requires collaboration among several parties, and the need to decrypt data for processing and analysis (the *in use* state referred to above) creates vulnerability. Given the often sensitive nature of the data, performing ML with FHE is an effective privacy-preserving solution.

Challenges of FHE for ML

When FHE is used for intensive computations like those required for ML training, functional challenges arise which require additional complex operations, such as bootstrapping (which allows large chains of encrypted computation on data). FHE with bootstrapping is required to perform most ML training tasks.

While bootstrapping addresses many functional challenges, it requires prohibitive amounts of compute power and time. For example, an unencrypted computation might take hundreds of milliseconds to complete on a standard laptop, yet could take many hours when running with FHE on a high-end server.

The Solution: Hardware Acceleration

The most promising efforts to make bootstrapping in FHE practical are focused on acceleration via hardware platforms. FHE workloads exhibit a high level of task and data parallelism that can be exploited by parallel processors. A low-cost but computationally efficient and highly optimized hardware co-processor is an ideal platform for accelerating the execution of core FHE operations.

Figure ES-1 shows the achieved and projected speedup from hardware acceleration of FHE using several different hardware backends, including FPGA, GPU, and ASIC, which are further discussed in this paper. Target speedup is what we believe is practical for generic computation.

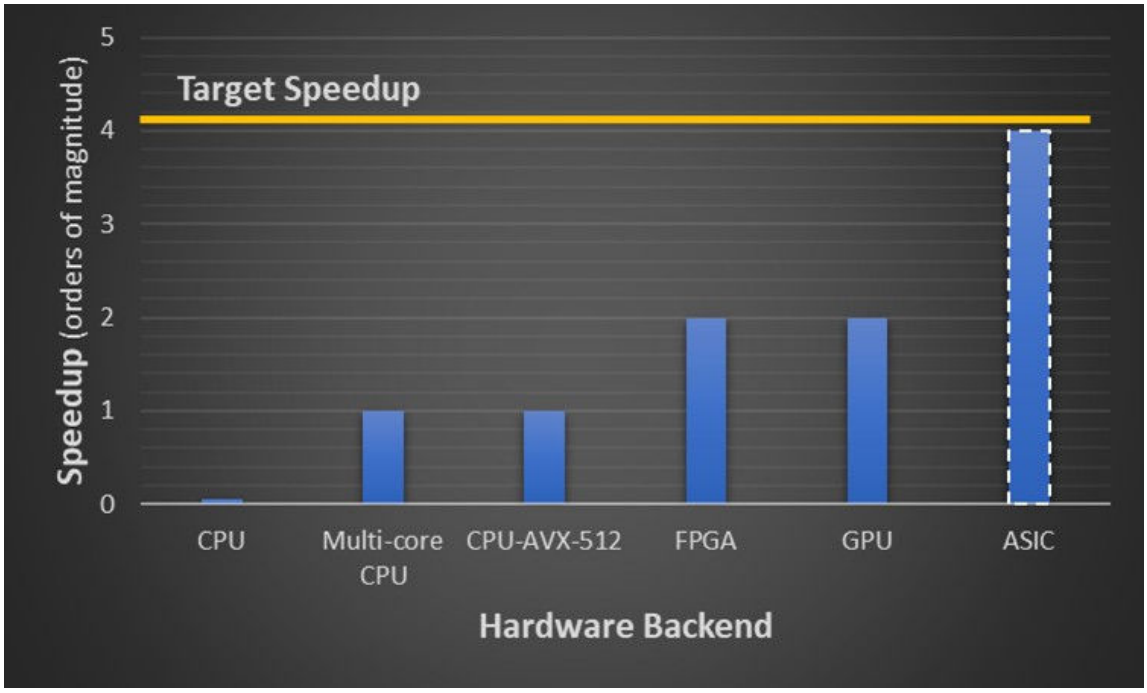

***Figure ES-1**: Achieved and projected speedup from hardware acceleration of FHE using different hardware backends. Multi-core CPU and GPU results are from [APAV+19], CPU-AVX-512 results are from [BKSD+21], and FPGA results are from [CRS16]. ASIC results are projected based on preliminary results of our ongoing research (to be published soon).*

Hardware acceleration poses several significant challenges, including both computational and memory bottlenecks. Solving these problems has transformative potential for enabling privacy-preserving ML using FHE.

Cryptographic Software Framework: The OpenFHE Library

Hardware acceleration requires reliable software implementation of the target functionalities on general-purpose CPUs. One prominent FHE software library is OpenFHE. A community-driven, open-source project, the OpenFHE Library has a diverse group of contributors from both industry and academia, including Duality, Samsung, Intel, MIT, UCSD, and others. With simpler APIs, modularity, cross-platform support, and integration of hardware accelerators, it is a resource for organizations–including providers of advanced hardware capabilities–to engage with the expanding field of FHE. Providers do not need to be experts, as Open FHE simplifies the user's access to many of the complicated cryptographic capabilities.

Hardware Abstraction Layer

Unique among current FHE libraries, OpenFHE provides a standard Hardware Abstraction Layer (HAL) designed to support different hardware acceleration technologies, such as Advanced Vector eXtensions (AVX), Graphics Process Unit (GPU), Field-Programmable Gate Array (FPGA), and Application-Specific Integrated Circuit (ASIC). The Intel HEXL library backend is a living example of HAL instantiation in OpenFHE.

The Opportunity for Hardware Acceleration Providers

Forward-looking providers of advanced hardware capabilities are increasingly exploring opportunities to accelerate FHE. Given the potential of hardware acceleration to make FHE practical in a variety of applications, this market trend is expected to grow. Such providers can refer to OpenFHE as the library to support their backends.

2. Introduction

2.1 The Challenges of Secure Collaboration on Sensitive Data

Enterprises across regulated industries increasingly need to collaborate on sensitive data. Protecting the privacy of this data is crucial, as reports of data breaches and attacks seem to appear daily. Data has three basic states: at rest, in transit, and in use. Organizations that handle personally identifiable information (PII), personal health information (PHI), intellectual property, intelligence insights, and other forms of sensitive information typically encrypt or otherwise protect that data while it is being stored (at rest) and when it is being transmitted (in transit). However, when the data is processed, analyzed, or manipulated in any way (in use), it must first be decrypted–making it vulnerable and increasing risk.

2.2 Defining HE and FHE

Homomorphic Encryption (HE) [RAD78] stands as a strong candidate solution to solve the data-in-use problem. HE is a set of encryption methodologies that allow computations to be performed on encrypted data while it is being used–without ever decrypting it–keeping data secure in all three states.

The mechanics of how HE works can be described as follows: an HE scheme maps elements of one mathematical set to elements of another set. The first set is known as plaintext (its elements are unencrypted data), while the second is known as ciphertext (its elements are encrypted data), as shown in Figure 1-a. The mapping is special to allow manipulations of ciphertext elements to produce ciphertext elements that upon decryption correspond to plaintext results that would be obtained after the desired manipulations on plaintext elements, as shown in Figure 1-b.

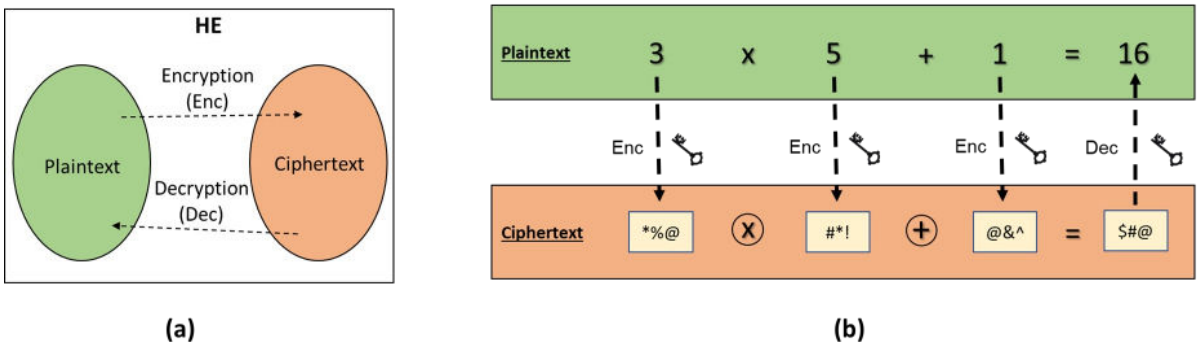

***Figure 1**: (a) Illustration of homomorphic encryption mapping between plaintext and ciphertext sets. (b) Homomorphic evaluation on ciphertext elements has the same effect as if the evaluation was conducted on the plaintext elements. Addition and multiplication are typical HE operations.*

Two important aspects to highlight are the number and types of operations one can do on ciphertexts. If the HE scheme supports only one operation type (either addition or multiplication) or a bounded number of any of them, it is dubbed a *Partial* or *Somewhat* HE scheme, respectively, whereas HE schemes that support an arbitrary number of both operations are known as *Fully* HE (FHE) [Gentry09]–our main focus in this document.

The main component in an FHE scheme that enables the evaluation of an arbitrary number of operations is a procedure known as bootstrapping, which is extremely compute-intensive. As a result, most FHE hardware acceleration efforts have been targeted toward accelerating bootstrapping. We will provide a more detailed description of this topic in subsequent sections.

3. FHE for Machine Learning

One of the most important and promising application domains for FHE is Machine learning (ML). ML is a subfield of Artificial Intelligence concerned with developing methods that leverage data to build models based on training data (the training phase) in order to make predictions or inferences from new unseen data (the testing/inference phase). ML has advanced rapidly in recent years, with the number of applications multiplying to include medicine, finance, transportation, natural language processing–and a variety of other fields where conventional algorithms cannot perform the needed tasks. Often ML requires collaboration among several parties: for example, one party develops ML models and makes them available to others as a cloud service (inference as a service). The need to decrypt sensitive data for processing and analysis (the *in use* state referred to above) creates vulnerability to both data breaches and IP theft, where the developer's ML model is stolen. Given the sensitive nature of the data involved, performing ML with FHE is a powerful solution, enabling computations on encrypted data and protecting private and personal information.

ML inference, when a trained ML system makes predictions using new data, is currently possible with FHE and even reasonably fast for several ML algorithms. By contrast, ML training, when an ML system first builds a model using the training dataset, is not currently practical with FHE (other than simple ML algorithms such as logistic regression), due to the huge computational requirements, most of which are needed to clear the untenable accumulation of error/noise resulting from FHE intrinsic mechanics, as discussed later in this section. Bootstrapping is one of the main noise maintenance operations in FHE; it is required for intensive workloads which are often needed in ML.

In this section, we discuss how bootstrapping can help make machine learning training practical with FHE. We then address the most significant barrier to bootstrapping in FHE.

3.1 Bootstrapping in FHE

FHE can be instantiated as a somewhat scheme, which means that the system parameters can be set to support homomorphic applications of certain computational requirements: that is, a limited number of operations. However, somewhat FHE cannot be used to perform generic computations, particularly training medium- to high-level machine learning models. For such tasks, FHE with bootstrapping is required.

Existing FHE schemes are based on lattice cryptography, which protects data by presenting attackers with complex mathematical challenges that are extremely hard to solve even with quantum computers. The encryption requires adding some noise, or error, to guarantee a certain level of security. Ciphertexts that include a small magnitude of noise are termed fresh ciphertexts. The noise grows at some controlled rate with each homomorphic operation. As long as the noise is under a certain magnitude, the decryption returns the correct result for a given computation. However, when FHE is used to perform intensive computations, e.g., those required for machine learning training, the noise can accumulate to the point that it is no longer possible to recover the original underlying plaintext from the now exhausted ciphertexts. Essentially, the data has been lost. A compute-intensive procedure called *bootstrapping* helps to mitigate this noise accumulation and transform exhausted ciphertexts into fresh ones that can be computed on further.

At a very high level, bootstrapping means homomorphic evaluation of the decryption function of the FHE scheme. Normal decryption requires the ciphertext and secret key and ensures the plaintext message as output. By contrast, bootstrapping requires an encryption of the secret key and the ciphertext, and ensures an equivalent ciphertext as output–equivalent in the sense that both ciphertexts encrypt the same message. However, the output ciphertext of bootstrapping has lower noise, that is, it can be used for further computation. Note that the encrypted secret key is never decrypted during bootstrapping; in other words, the secret key is never exposed. We also underline that bootstrapping is the main performance-critical operation in FHE since it requires huge computational resources.

Training an ML model involves multiple passes over the training data to learn and fine-tune a set of parameters of an objective function that maps input samples to certain outputs. In each pass, the parameters are updated toward meeting some predetermined performance metrics. This iterative process transforms the input data via linear and non-linear transformations. Evaluating this process in FHE results in a deep computational workload incurring large noise growth that can be mitigated via bootstrapping. Since bootstrapping is the most performance-critical operation in FHE, evaluating it several times inhibits the overall performance.

3.2 The Main Barrier to FHE for ML

While bootstrapping addresses noise accumulation in applications such as ML, it requires huge amounts of compute power and time. For example, a computation might take a millisecond to complete on a standard laptop, yet would take hours when running FHE on a high-end server. In other words, where bootstrapping is required, FHE is not yet practical.

FHE schemes can be categorized into two families: bit-wise (or scalar-level) schemes that work on (typically small) encrypted integers and vector-level schemes that work with vectors of integers or real numbers. The former work on individual encrypted numbers, whereas the latter allow for Single-Instruction Multiple-Data (SIMD) packing (packing vectors of numbers in one encrypted message). The challenges of bootstrapping differ for scalar-level and vector-level schemes. Bootstrapping[1] for scalar-level schemes is relatively fast and can run on a conventional CPU in a sub-second time. Yet it is highly serial (non-parallel), requiring many bootstrapping operations for a large number of encrypted numbers. By contrast, bootstrapping for the vector-level family is slow, requiring multiple seconds, but it is inherently parallel so that thousands of numbers can be processed in one bootstrapping operation, achieving faster amortized time per every bootstrapped number. However, in both cases, the necessary amount of compute power and time often makes bootstrapping impractical.

Theoretical developments in FHE with bootstrapping are almost saturated as of now, with mainly incremental algorithmic advancements anticipated. Instead, the most promising efforts to make bootstrapping in FHE practical are focused on acceleration via hardware platforms.

4. Hardware Acceleration

As discussed, FHE is not yet practical for certain applications like machine learning training, in large part due to computational bottlenecks in lattice-based FHE schemes when run on commodity CPU-based computation devices. FHE workloads exhibit a high level of task and data parallelism that can be exploited by parallel processors, making them ideal platforms for accelerating FHE. A low-cost but computationally efficient and highly optimized hardware co-processor might effectively accelerate the execution of core FHE operations.

This section provides an overview of different hardware backends that might make FHE for ML practical, then discusses the main challenges in hardware acceleration.

4.1 Options for Hardware Acceleration

Currently, homomorphic computations on encrypted data are typically several orders of magnitude (often between 4 and 6) slower than plaintext computations. Theoretical and practical studies are exploring several different options to accelerate FHE operations, including Graphics Process Unit (GPU), Field-Programmable Gate Array (FPGA), and Application-Specific Integrated Circuit (ASIC), as discussed in this section.

1 We use “bootstrapping” to refer to bootstrapping as an atomic FHE operation, that is, bootstrapping one ciphertext.

FPGA

Collaborators at Duality reported on early advances in designing and implementing an FPGA-based computation accelerator as part of a Homomorphic Encryption Processing Unit (HEPU) co-processor [CRS16]. The strategy is to accelerate key computational bottlenecks that are common across FHE schemes, which can be much more rapidly executed on “custom” non-conventional computers. The focus is particularly on accelerating complex and compute-intensive mathematical operations that form the low-level building blocks of FHE schemes. This capability was evaluated on Xilinx Virtex-7 FPGAs that attach to a host computer through a PCI-Express port as HEPU co-processors. The design was intended to be adaptable and extensible, with a modular software architecture that supported rapid prototyping with alternative FHE schemes, easier integration of the HEPU into broader computing infrastructures, and increased parallelism for efficient execution performance. This approach yielded experimentally a two orders-of-magnitude improvement in runtime, compared to a reference CPU-based implementation, for performance-critical operations.

GPU

Dramatic recent advances in usable GPU technology have brought wider bit-width operations needed for secure and practical FHE closer to reality. GPUs have significant penetration into the cloud computing commodity market, and as such, may prove a valuable implementation platform for the acceleration of FHE. There have been some successful early efforts to support homomorphic encryption with GPU-based implementation. Based on a performance evaluation of two optimized FHE variants by a group of collaborators, including several at Duality [APAV+19], GPU implementation showed two orders of magnitude speedup as compared to single-thread conventional CPU results, providing a solution that might be desirable for cloud environments supporting GPU computations.

DARPA DPRIVE and ASIC

The U.S. Defense Advanced Projects Agency (DARPA) recently awarded contracts to four teams to develop ASIC accelerators for fully homomorphic encryption as part of its data protection in virtual environments (DPRIVE) program. The goal of the 3.5-year DPRIVE program is to enable computation on FHE-encrypted data within one order of magnitude of the compute time of current unencrypted computation [Ward21].

Figure 2 below shows the achieved and projected performance of several hardware backends. Target speedup is what we believe is practical for generic computation.

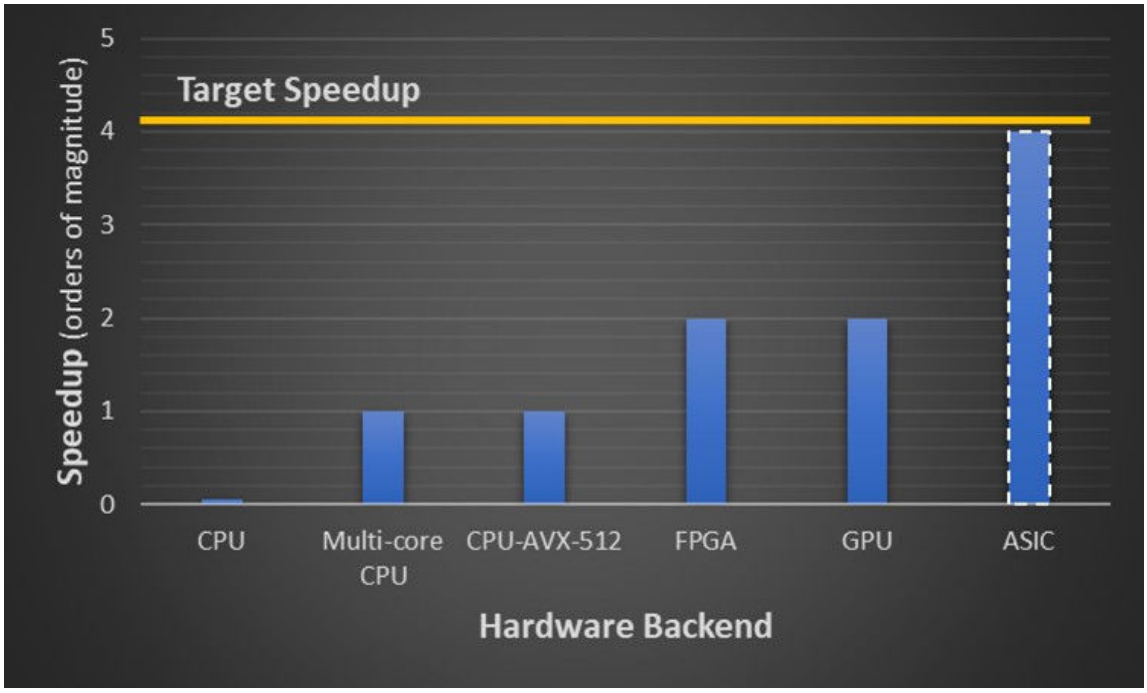

Figure 2: Achieved and projected speedup from hardware acceleration of FHE using different hardware backends. Multi-core CPU and GPU results are from [APAV+19], CPU-AVX-512 results are from [BKSD+21], and FPGA results are from [CRS16]. ASIC results are projected based on preliminary results of our ongoing research (to be published soon).

4.2 Hardware Acceleration Challenges

Hardware acceleration poses several significant challenges, including both computational bottlenecks and memory-bound limitations. Solving these problems has transformative potential to enable practical privacy-preserving machine learning using FHE.

It is important to realize that scalar-level and vector-level FHE schemes impose different computational requirements. The elementary homomorphic operations in scalar-level schemes require much smaller computational resources when compared to those of the vector-level schemes. In addition, scalar-level homomorphic operations are mostly serial in nature (including limited data parallelism), whereas vector-level homomorphic operations are parallel in nature, exhibiting a high level of data parallelism. Moreover, the characteristics of the computational workloads of the applications developed using either family are quite different. For instance, applications developed using scalar-level FHE schemes often include a very large number of encrypted (often Boolean) operations, most of which are independent and can run simultaneously given enough computational resources; such workloads include a large level of task parallelism. On the other hand, applications developed using vector-level FHE schemes often include a smaller number of encrypted arithmetic operations, with each operation exhibiting a large amount of data parallelism, as discussed above. Our focus hereafter is directed toward the vector-level family of schemes, as these schemes are typically more efficient for practical applications.

The computational bottlenecks of FHE are centered around the polynomial arithmetic used for implementing the FHE schemes. FHE schemes work with very large polynomials that include many thousands of large bounded integers (each integer is several hundred bits in size). Essentially, homomorphic operations require manipulating these polynomials via schoolbook polynomial arithmetic such as addition, subtraction, and multiplication with polynomial multiplication being the

main bottleneck in performance. Moreover, since these polynomials include very large coefficients, complex algorithms are required to carry out their arithmetic efficiently. Some of these algorithms are the Number Theoretic Transform (NTT), fast modular arithmetic, and Residue Number System (RNS) basis extension and scaling. Describing these operations in detail would require a paper on its own. We refer the reader to the reference list for further resources.

Posing an additional challenge to hardware acceleration, recent theoretical and practical hardware acceleration studies suggest that deep FHE computations [ABBC+22], which often leverage bootstrapping, are not just compute-bound but are also memory-bound. For instance, the theoretical study [DAYC+21] shows that secure implementations of CKKS[2] bootstrapping exhibit low arithmetic intensity (< 1 Operation/byte) and require large caches (> 100 MB), making it heavily bound by the main memory bandwidth. An FHE hardware acceleration study [SFKD+21] describes data movement as the main bottleneck in FHE hardware acceleration of deep homomorphic computations based on BGV[3] and CKKS, and proposes a design that minimizes the data movement.

5. OpenFHE Library

Hardware acceleration requires reliable software implementation of the target functionalities on general-purpose CPUs. One prominent FHE software library is OpenFHE [ABBC+22]. The OpenFHE Library is a software library for organizations–including hardware acceleration providers–to engage with the expanding field of FHE. The Library is at the forefront of making FHE technology practical, user-friendly, and applicable in a wide variety of contexts. The OpenFHE library is a community-driven, open-source project with a diverse group of contributors from both industry and academia, including Duality, Samsung, Intel, MIT, UCSD, and others. The OpenFHE Library provides efficient implementations of lattice cryptography building blocks and all common FHE schemes. The Library prioritizes usability, providing simpler APIs, modularity, and cross-platform support. Addressing both present practicality and the future of the technology, the Library is designed to support bootstrapping and integrate multiple hardware acceleration backends.

Compliant with the HomomorphicEncryption.org post-quantum security standards for homomorphic encryption, OpenFHE is offered under the 2-clause Berkeley Software Distribution (BSD) open-source license, making it easier to wrap and redistribute OpenFHE in products. The OpenFHE Library is formally affiliated with the NumFocus stable of open-source software projects. The Google Transpiler uses the CGGI (TFHE) implementation as the FHE backend. The first stable version (v1.0.1) was released on November 30, 2022.

2 CKKS is a popular vector-level FHE scheme that is highly suitable for floating-point FHE applications.

3 BGV is a popular vector-level FHE scheme that is highly suitable for integer FHE applications.

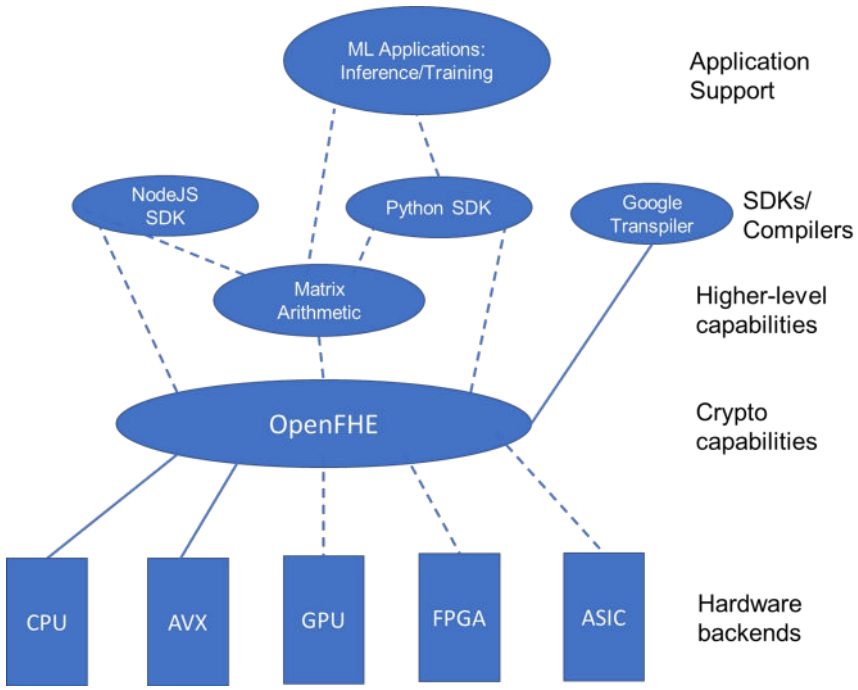

***Figure 3**: OpenFHE future vision with support for multiple hardware backends and auxiliary helper libraries and Software Development Kits (SDKs) to achieve practical ML applications in FHE. Solid lines correspond to publicly available features in OpenFHE, whereas dashed lines correspond to new features under development.*

This section discusses the OpenFHE Library, particularly as it relates to bootstrapping for all schemes and the Hardware Abstraction Layer (HAL) to support various hardware acceleration backends for FHE mathematical operations. We also describe the Intel HEXL library backend as an example of HAL instantiation in OpenFHE.

5.1 Bootstrapping for All Schemes

As discussed in previous sections, there are multiple FHE schemes for different data types. Since some schemes are preferable for some types of computations, while others are better for others, the ability to switch between schemes is crucial to allow for a practical execution of arbitrary computations which can be done via variations of bootstrapping. Given the complexity of scheme switching, one approach has been to choose one scheme and attempt to apply it to solve all computations. The OpenFHE Library takes a more future-oriented approach, designed under the assumption that bootstrapping is available for all implemented schemes and switching ciphertexts between schemes is possible. The current version of the library includes prototypes for switching between some FHE schemes.

5.2 OpenFHE: HAL

The developers of the OpenFHE Library assume that hardware acceleration is crucial for the practical use of FHE. Moreover, they recognize that different backends might be preferable, depending on application and context. Before the OpenFHE Library, hardware acceleration work was tailored to a given hardware acceleration technology and application. Unique among current FHE libraries, OpenFHE provides a general abstraction mechanism for various hardware acceleration backends and applications. This standard Hardware Abstraction Layer (HAL) is designed to support different

hardware acceleration technologies, such as Advanced Vector eXtensions (AVX), Graphics Process Unit (GPU), Field-Programmable Gate Array (FPGA), and Application-Specific Integrated Circuit (ASIC). As a result, OpenFHE is the go-to library for supporting smooth hardware acceleration and integration.

The development of HAL and related features has been funded in part by the DARPA DPRIVE program. These features are expected to support the ASIC-based hardware acceleration of cryptographic protocols implemented in OpenFHE, which is one of the goals of DARPA DPRIVE [ABBC+22].

OpenFHE has a modular design with multiple layers, as illustrated in Figure 4. It consists of 5 main layers: Encoding, Application, Crypto, Lattice, and Primitive Math. The goal of the design is to provide a backend-aware implementation for low-level mathematical procedures while using the same implementation of cryptographic capabilities for all backends.

In the primitive math layer, we define abstract classes for native integers, modular vectors, and NTT transformations. These classes describe the interface for NTT, vector addition, vector multiplications, and other operations needed to support polynomial arithmetic for native integers, that is, fit in the underlying processor word size.

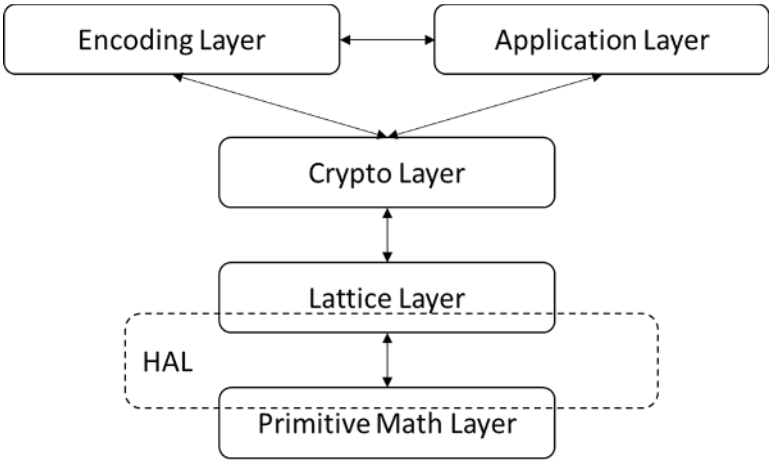

***Figure 4**. Layers in OpenFHE. Adapted from [ABBC+22].*

In the lattice layer, we define abstract classes for polynomials, focusing on efficient polynomial representations and data structures. An array of algebraic optimizations are defined in this layer, such as the Chinese Remainder Theorem and Residual Number System procedures.

All the algorithms for polynomial arithmetic are implemented in the math and lattice/polynomial layers. The purpose of HAL is to support multiple instantiations of the bottleneck operations in these layers. This is achieved by defining abstract classes for the functionality that can have hardware-backend-specific implementation.

The higher layers in OpenFHE: 1) the crypto layer, which includes the implementation of different FHE schemes, 2) the application layer the developer uses to build FHE applications, and 3) the encoding layer used to encode/decode data to/from FHE-friendly datatypes from/to conventional datatypes, stay intact regardless of the target hardware backend as the HAL layer abstracts (hides) the low-level hardware details.

The OpenFHE hardware integration paradigm allows multiple hardware backends to be supported by using distinct git repositories. A hardware-specific backend implementation can be housed in a separate repository. There is also a special openfhe-configurator module that automatically builds OpenFHE for a selected backend by merging the code from the main OpenFHE repository with the code in the backend repository.

5.3 Intel HEXL

The Intel HEXL library [BKSD+21] backend is a living example of HAL instantiation in OpenFHE [ABBC+22]. OpenFHE includes a hardware backend implementation for the Intel Homomorphic Encryption Acceleration Library (HEXL), which is available in the openfhe-hexl[4] Github repository under the OpenFHE Github organization. HEXL is an open-source C++ library that provides efficient implementations of integer arithmetic on finite fields. It is available at https://github.com/intel/hexl under the Apache 2.0 license. HEXL uses the Intel® Advanced Vector Extensions 512 (Intel® AVX512) instruction set to implement polynomial operations with word-sized primes on 64-bit Intel processors. Near-optimal speedups have been achieved for low-level polynomial arithmetic operations such as NTT, modular additions, subtractions, and multiplications.

6. Concluding Remarks

In this paper, we have discussed the challenge of protecting sensitive data while allowing organizations to collaborate. FHE is a potential solution, enabling arbitrary computations over encrypted data without using the secret key. We particularly focused on the important and rapidly expanding field of machine learning (ML). Bootstrapping is almost always required in FHE for ML. An extremely compute-intensive procedure with a high memory footprint, bootstrapping is the performance chokepoint on regular computers.

The most promising efforts to make bootstrapping in FHE practical are focused on acceleration via hardware platforms. We discussed several options to accelerate FHE operations and the greatest challenges. We then explored hardware acceleration in the context of the OpenFHE Library, an open-source cryptographic software library for organizations–including hardware acceleration providers–to engage with the expanding field of FHE. Unique among current FHE libraries, OpenFHE provides a standard Hardware Abstraction Layer (HAL) designed to support different hardware acceleration technologies.

Forward-looking providers of advanced hardware capabilities are increasingly exploring opportunities to accelerate FHE. Given the potential of hardware acceleration to make FHE practical in a variety of

4 Available at: https://github.com/openfheorg/openfhe-hexl

applications, this market trend is expected to grow. Such providers can refer to OpenFHE as the library to support their backends. Providers do not need to be experts as OpenFHE addresses cryptographic capabilities.

References

[ABBC+22] Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D. B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., RV, S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., & Zucca, V. (2022, November). OpenFHE: Open-source fully homomorphic encryption library. In Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (pp. 53-63).

[APAV+19] Al Badawi, A., Polyakov, Y., Aung, K. M. M., Veeravalli, B., & Rohloff, K. (2019). Implementation and performance evaluation of RNS variants of the BFV homomorphic encryption scheme. IEEE Transactions on Emerging Topics in Computing, 9(2), 941-956.

[BKSD+21] Boemer, F., Kim, S., Seifu, G., DM de Souza, F., & Gopal, V. (2021, November). Intel HEXL: accelerating homomorphic encryption with intel AVX512-IFMA52. In Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography (pp. 57-62).

[CRS16] Cousins, D. B., Rohloff, K., & Sumorok, D. (2016). Designing an FPGA-accelerated homomorphic encryption co-processor. IEEE Transactions on Emerging Topics in Computing, 5(2), 193-206.

[DAYC+21] De Castro, L., Agrawal, R., Yazicigil, R., Chandrakasan, A., Vaikuntanathan, V., Juvekar, C., & Joshi, A. (2021). Does Fully Homomorphic Encryption Need Compute Acceleration? Cryptology ePrint Archive, Paper 2021/1636. https://eprint.iacr.org/2021/1636 https://eprint.iacr.org/2021/1636.

[Gentry09] Gentry, C. (2009, May). Fully homomorphic encryption using ideal lattices. In Proceedings of the forty-first annual ACM symposium on Theory of computing (pp. 169-178).

[RAD78] Rivest, R. L., Adleman, L., & Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. Foundations of secure computation, 4(11), 169-180.

[SFKD+21] Samardzic, N., Feldmann, A., Krastev, A., Devadas, S., Dreslinski, R., Peikert, C., & Sanchez, D. (2021). F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (Virtual Event, Greece) (MICRO '21). Association for Computing Machinery, New York, NY, USA, 238–252. https://doi.org/10.1145/3466752.3480070.

[Ward21] Ward-Foxton S. (2021). Darpa Funds ASIC Accelerators for Fully Homomorphic Encryption. Url: https://www.eetimes.com/darpa-funds-asic-accelerators-for-fully-homomorphic-encryption/, Accessed: Dec 2022.

About Duality

Duality's mission is to deliver practical privacy-enhancing technologies to the market. Duality comprises world-renowned cryptographers and data scientists, including Shafi Goldwasser, a Turing award winner, and Vinod Vaikuntanathan, a Godel Prize winner. Backed by Intel Capital, Hearst Ventures, and Team8, Duality has been funded by multiple DARPA projects to prototype and apply FHE. One such initiative is DPRIVE, with the objective to design and implement a hardware chip that accelerates machine learning workloads on encrypted data.