FHERMA: Building the Open-Source FHE Components Library for Practical Use

Gurgen Arakelov[1], Nikita Kaskov[1], Daria Pianykh[1] and Yuriy Polyakov[2]

[1] Fair Math, {gurgen,nikita,daria}@fairmath.xyz
[2] Duality Technologies, ypolyakov@dualitytech.com

Abstract. Fully Homomorphic Encryption (FHE) is a powerful Privacy-Enhancing Technology (PET) that enables computations on encrypted data without having access to the secret key. While FHE holds immense potential for enhancing data privacy and security, creating its practical applications is associated with many difficulties. A significant barrier is the absence of easy-to-use, standardized components that developers can utilize as foundational building blocks. Addressing this gap requires constructing a comprehensive library of FHE components, a complex endeavor due to multiple inherent problems. We propose a competition-based approach for building such a library. More concretely, we present FHERMA, a new challenge platform that introduces black-box and white-box challenges, and fully automated evaluation of submitted FHE solutions. The initial challenges on the FHERMA platform are motivated by practical problems in machine learning and blockchain. The winning solutions get integrated into an open-source library of FHE components, which is available to all members of the PETs community under the Apache 2.0 license.

Keywords: fully homomorphic encryption, privacy-enhancing technologies, challenges, cryptography, privacy

Contents

1 Introduction

Fully Homomorphic encryption (FHE) is a powerful cryptographic primitive that enables performing computations over encrypted data without having access to the secret key. FHE holds considerable promise across diverse sectors such as finance, healthcare, blockchain, and cloud computing, offering groundbreaking solutions for data security and privacy.

In Machine Learning (ML), FHE enables secure outsourcing of computations to an untrusted cloud while ensuring the privacy of sensitive data. This allows organizations to collaborate on ML models without compromising data privacy. FHE plays a critical role in distributed ML as it has the ability to support confidential secure computing scenarios [MSM+22]. In data analysis, FHE enables privacy-preserving analytics on sensitive datasets, allowing organizations to extract valuable insights while protecting individuals' privacy. In blockchain, FHE can enhance privacy and confidentiality by enabling encrypted computations within smart contracts, ensuring that sensitive data remains confidential even when stored on a decentralized ledger.

Although FHE has seen significant progress, its widespread adoption at the developer level faces notable obstacles. FHE is a compute-intensive privacy solution, and its efficient configuration often requires multidisciplinary expertise in cryptography, the application field, and low-level mathematical algorithms. This requirement for a broad range of knowledge makes it challenging for developers to easily integrate FHE into their projects, leading to a slower adoption rate despite its potential benefits for privacy and security. Existing solutions in the FHE field lack an FHE Components layer for developers, which would simplify the development process without requiring deep dives into cryptography and mathematical primitives. Additionally, the computational overhead associated with FHE is a significant barrier, as it demands more processing power and longer execution times compared to non-encrypted computations, further complicating its practical deployment in resource-constrained environments.

In general, we can classify the challenges limiting the broad adoption of FHE technology into three primary groups:

- Lack of FHE application development tools
- High computational requirements for FHE
- No standard methodology for developing and benchmarking FHE-based applications

Presently, there is a clear lack of a strong and comprehensive library of standardized FHE components. This deficiency is a significant obstacle to the widespread adoption of FHE. Without a high-level FHE components library, developers must directly handle the intricate, low-level cryptographic processes involved in FHE. This requires a deep understanding of cryptography and substantially increases the time and effort needed to develop secure, efficient applications. This shortfall impedes the smooth integration of FHE into various software solutions, thereby restricting its potential to boost privacy and security in digital services.

Despite the potential for secure data processing, the significant computational overhead associated with FHE operations limits its practicality, especially in scenarios requiring fast or real-time data handling. The lack of standard methods is also one of the most important problems in FHE. Without universally accepted standards, developers, and organizations face uncertainty regarding the compatibility, security, and efficiency of FHE implementations.

In this paper, we extend the layered approach used in designing the OpenFHE library [AAB+22] for building an FHE ecosystem tailored for application developers. We introduce an FHE Components Layer that provides a simpler-to-use interface, enabling developers to avoid the complex details of cryptography and mathematics. While this layer could greatly

simplify the development process, there are presently no solutions that address this need. We consider the FHE components library as the foundation for application development. It is designed to support the creation of secure and efficient applications using the FHE technology, simplifying the process for developers across various real-world uses. However, assembling this library comes with its own set of hurdles. A major challenge is creating a comprehensive collection with limited developer resources. To overcome this obstacle, we propose a crowdsourced platform where members of FHE community can compete with each other in building practical components for various application domains, with the initial focus placed on ML and blockchain applications. The winning solutions get integrated in the FHE components library.

1.1 Our contribution

Developing FHE applications is a complex task that demands a variety of skills. In this paper, we propose a methodology aimed at simplifying the development process of new applications by building an FHE ecosystem. We introduce a layered methodology, where different libraries and tools for developers are categorized into specific layers.

One of the layers is **FHE Components Layer**, which is intended to serve as the pivotal link between FHE libraries and existing development tools. Currently, there is a significant gap in the market as there are no high-level FHE component libraries available. Constructing such a library poses a substantial challenge, requiring broad community involvement.

To facilitate this development, we have established the FHERMA platform, a dedicated environment for hosting FHE Challenges. This platform is designed to harness the collective expertise and creativity of the FHE community to co-create solutions that can fill the gaps in current FHE application development tools. By focusing on the FHE Components Layer, we aim to develop a comprehensive suite of tools that will significantly enhance the ease and efficiency of creating robust FHE applications.

Through the FHERMA platform, we invite developers, researchers, and enthusiasts to participate in a series of challenges that will push the boundaries of what is currently possible with FHE technology. This collaborative effort is crucial for accelerating the adoption of FHE and enabling the development of secure, efficient, and scalable applications across various sectors. The initial version of the FHERMA platform is based on the OpenFHE library. However, expanding the list of supported libraries is an important step to create a transparent and user-friendly environment for solving challenges. Important aspects of the platform are its focus on community, openness, and fairness. Unlike existing platforms, where participants are required to sign an agreement giving the company the rights to all submitted solutions and also entailing some non-compete and confidential restrictions, our platform only requires that all submitted solutions be distributed under the Apache 2.0 license.

Our contribution, therefore, lies not only in proposing a structured approach to FHE application development but also in actively facilitating the creation of a dynamic and collaborative ecosystem where FHE-based application capabilities can be advanced collectively.

1.2 Related projects

Homomorphic encryption solutions can be broadly classified into three main categories:

- FHE libraries
- SDK / Toolkits
- Applications

Within the category of FHE libraries, OpenFHE [AAB+22] supports all modern FHE schemes and provides an API designed to facilitate the development of homomorphic encryption projects, making it suitable for both research and practical development purposes. Lattigo [MTPBH20] is another notable example, offering an extensive toolkit for lattice-based homomorphic encryption, which aids in the implementation and exploration of various cryptographic protocols. Moreover, tfhe-rs [Zam22] offers a Rust-based version of the TFHE library[1], distinguished for its efficiency and its capability for programmable bootstrapping. FHE libraries vary in terms of supported encryption schemes, programming languages, and licenses. Table 1.2 provides organized details about contemporary FHE libraries.

Table 1: FHE libraries

Project	Supported FHE schemes	Language	License
Λoλ [CP15]	RLWE-based FHE schemes	Haskell	No license specified
Concrete [FSB+23]	CGGI/TFHE	Rust, Python	BSD 3-Clause Clear
HEAAN [CKY18]	HEAAN	C/C++	Attribution-NonCommercial 3.0 Unported
HElib [HS20]	BGV, CKKS	C/C++, Python	Apache-2.0
Lattigo [MTPBH20]	BFV, BGV, CKKS, LMKCDEY	Go	Apache-2.0
OpenFHE [AAB+22]	BFV, BGV, CKKS, DM/FHEW, CGGI/TFHE, LMKCDEY	C/C++, Python, Rust[2]	BSD 2-Clause
SEAL [CLP17]	BFV, BGV, CKKS	C/C++, Python	MIT License

In the category of toolkits, multiple projects have aimed at creating frontends for FHE computations. These initiatives are generally classified as compilers and transpilers, providing tools such as a Domain-Specific Language (DSL), a modified version of a high-level programming language, or a Software Development Kit (SDK) tailored for crafting FHE applications. Among the assortment of toolkits available, there are several notable compilers and transpilers, including:

- **HEIR** [BZZ+23]: MLIR-based toolchain designed for FHE compilers. HEIR utilizes the modular LLVM infrastructure to provide a flexible and powerful foundation for developing FHE applications.

- **IBM HELayers** [AAB+23]: a toolkit that includes capabilities for FHE ML inference with a Neural Network and a Privacy-Preserving key-value search.

- **Concrete-ML** [FSB+23]: a Python-based compiler intended for data scientists without prior knowledge of FHE. It integrates with popular machine learning libraries such as sklearn, pyTorch, and XGBoost, making it possible to apply FHE within machine learning models with a minimal learning curve.

- **EVA**[DKS+19] : a compiler and optimizer for the CKKS scheme, targeting the Microsoft SEAL library.

In Table 2, a selection of current compilers and transpilers is presented, each employing one or more FHE libraries (for a more detailed list of projects, the reader is referred to [Sch]).

[1] https://github.com/tfhe/tfhe

Table 2: Comparison of FHE compilers

Project name	Citations	Frontend	FHE implementation used
Alchemy	[CPS18]	Haskell-based DSL	Λoλ
Cingulata/Armadillo	[CDS14]	C++ dialect	TFHE
Concrete-ML	[FSB+23]	Python-based DSL	TFHE
EVA	[DKS+19]	Python-based DSL	SEAL
HE-Layers	[AAB+23]	C++ and Python	OpenFHE, HEAAN, HELIB
HEIR	[BZZ+23]	Multiple dialects	OpenFHE(PALISADE)
Marble	[VS18]	C++ dialect	HElib
RAMPARTS	[ATD+19]	Julia	PALISADE
Transpiler	[GSPH+21]	C++ dialect	TFHE and OpenFHE

A primary objective of these projects is to facilitate the development of FHE applications with the same ease and familiarity as writing applications in cleartext.

The range of existing toolkits marks significant advancement towards making FHE more usable and accessible for a wide array of applications. They fulfill the demand for certain components and features within the FHE field, catering to different requirements of encrypted data processing and application development. However, the variety of these tools, each based on unique principles and addressing different aspects of FHE application, highlights a scattered FHE development tool landscape. This disparity may pose difficulties for developers in terms of navigating and combining various libraries and toolkits into their work. Adopting a more integrated and standardized methodology could streamline the development process, thus improving the efficiency and broader implementation of FHE solutions.

Another major shortcoming of existing FHE compilers is the lower efficiency of compiled FHE applications, as compared to optimized solutions developed by experts in FHE and related application domains.

In our work, we utilize a competition-based approach to build the core FHE components library. Competitions have previously been utilized in the field of FHE in projects such as the iDASH[3] secure genome analysis competition and the Zama Bounty Program[4]

iDASH, short for integrating Data for Analysis, Anonymization, and Sharing, is an annual competition that focuses on specific challenges related to privacy and security in genomics research. The competition is organized by the iDASH consortium, which includes researchers from various institutions, and aims to promote advancements in privacy-preserving technologies for sharing and analyzing sensitive genomic data. Participants are tasked with developing innovative solutions to real-world problems in genomics while ensuring the privacy and confidentiality of individuals' genetic information. The competition typically features multiple tracks, each targeting different aspects of genomic privacy, and attracts participants from both academia and industry. Through IDASH, researchers collaborate to develop and evaluate cutting-edge privacy-preserving techniques that enable secure and responsible data sharing in genomics research.

Zama Bounty Program offers monetary rewards for individuals or teams that successfully tackle specific challenges. These challenges may include improving existing FHE libraries, developing new algorithms, or creating innovative applications that leverage FHE for privacy-preserving computations.

The competition-based approach has demonstrated its effectiveness. ZAMA Bounty program[5] incentivized researchers to develop various solutions, including a string library for encrypted data using TFHE-rs, encrypted matrix inversion, a homomorphic regex

[3]iDASH secure genome analysis competition: http://www.humangenomeprivacy.org
[4]Zama Bounty Program GitHub page: https://github.com/zama-ai/bounty-program
[5]ZAMA website: https://www.zama.ai/

engine, and an FHE-encrypted key-value database.

The iDASH competitions have had a significant effect on FHE in general. iDASH'17 showed the power of CKKS. A winning solution of iDASH'18 ([KJT+20]) led to the study showing that secure large-scale genome-wide association studies based on FHE are already practical [BGPG20].

While competitions, such as iDASH, have offered important contributions, integrating these advancements into a comprehensive FHE component library remains a challenge.

1.3 Organization

In Section 2.1, we provide an overview of modern FHE schemes and libraries for homomorphic encryption. In Section 2.2, we examine the barriers hindering the widespread adoption of FHE technology, focusing on three main obstacles that we believe are critical for broader adoption. Section 3 discusses a layered approach for developing an FHE ecosystem, aimed at resolving some of the challenges identified in Section 2.2. A significant part of this ecosystem is the FHE Components Layer. In Section 4.1, we tackle the issues involved in creating such a library, review potential existing projects that could fulfill this role, and detail the FHERMA platform we have developed for constructing the FHE components library.

2 Background and motivation

2.1 Cryptographic capabilities

The most common FHE schemes are often grouped into three classes based on the data types they support computations on. The first class of FHE schemes is characterized by its focus on modular arithmetic operations over finite fields. This class includes schemes such as the Brakerski-Gentry-Vaikuntantan (BGV, [BGV11]) and Brakerski/Fan-Vercauteren (BFV, [FV12, Bra12]) schemes. Typically, computations on vectors of integers modulo a prime number or a prime power are supported. This class of schemes is suited for applications involving integer computations and small-integer arithmetic.

The second class of FHE schemes primarily specializes in Boolean circuits and decision diagrams. Examples within this category include the Ducas-Micciancio (DM, [DM14]) and Chillotti-Gama-Georgieva-Izabachene (CGGI, [CGGI16]) schemes. This class offers fast bootstrapping capabilities, but has limited SIMD packing support, as native support for this feature may be limited. Schemes in this class are suitable for tasks involving logical operations and binary decision-making processes. The third class of FHE schemes supports approximate computations over vectors of real and complex numbers. Represented by the Cheon-Kim-Kim-Song (CKKS, [CKKS16]) scheme, this class extends the capabilities of FHE to handle continuous data representations. CKKS enables computations with approximate precision, making it well-suited for applications in machine learning inference/training, signal processing, and other domains where computations involve real or complex numbers.

In addition, schemes can be divided into two categories: schemes with exact results, such as BGV and BFV, and schemes for approximate arithmetic, such as CKKS.

Most modern FHE schemes are based on the hardness of the Learning With Error (LWE) problem or its Ring (RLWE) version, where noise is added to achieve the hardness properties. The magnitude of the noise grows with each encrypted operation. When the noise reaches the level where no more computations are possible, a bootstrapping procedure can be used to reset the noise to a lower level and to enable more computations.

Each scheme has its unique advantages and limitations, and there is no single encryption scheme that is universally optimal for all types of problems.

The Turing completeness of FHE implies the potential to execute any computable function on encrypted data. However, achieving this capability does not inherently provide an algorithm, and the development process is often complex and challenging. Due to performance constraints, practical implementation is usually feasible only for algorithms (circuits) with a multiplicative depth below a predefined threshold L. This substantially complicates application development, imposing considerable restrictions on developers and necessitating innovative solutions to navigate these constraints.

2.2 Barriers to mass adoption

We categorized the issues existing in FHE into three primary groups: Lack of FHE Components, High computational requirements and FHE Standardization. This organization helps to highlight the critical areas where improvements are needed to advance the field. In this section, we delve into these categories in more detail.

2.2.1 Lack of FHE components

The available libraries and tools for Fully Homomorphic Encryption mainly provide fundamental, basic features focused on the underlying cryptography and mathematics. To use them effectively, developers need to have a strong grasp of cryptography. However, the widespread adoption and practical implementation of this technology urgently require the establishment of FHE Components layer to advance development tools. Without it, developers must navigate FHE's complex cryptographic procedures directly, necessitating extensive knowledge in cryptography and leading to increased development time and effort for creating secure and efficient applications.

Within this paper, "FHE components" are understood to be fundamental elements designed to execute operations on encrypted data. These components encompass a range of functionalities, from simple, like computing square roots, to more complex like sorting encrypted arrays. Some components can be constructed using polynomial approximation. The precision of this approximation relies on two factors: the domain over which the function must be evaluated and the maximum degree of the polynomial. Increasing the polynomial degree results in a more accurate approximation, but also increases the requirements for multiplicative levels, affecting performance. Although polynomial approximation provides significant power and utility, its application does not always guarantee sufficient precision and may not be feasible in certain situations.

Despite its potential, building the FHE component library is challenging for several reasons:

1. **Multidisciplinary Expertise Requirement**: Designing FHE components requires expertise in multiple domains, including cryptography, mathematics, and domain-specific fields. Participants with various backgrounds need to collaborate effectively to address this challenge.

2. **Evaluation and Validation**: Ensuring the correctness, efficiency, and security of FHE components is essential for their practical use. Rigorous evaluation and validation mechanisms, including automated testing, peer review, and benchmarking against established standards, are necessary.

3. **Collaboration Difficulties**: Building a library of FHE components requires active participation and collaboration from the FHE research and practitioner community. Facilitating discussions, knowledge sharing, and collaboration among participants foster interaction and teamwork.

4. **Intellectual Property and Licensing**: Addressing intellectual property rights and licensing issues is crucial for ensuring the open accessibility and usability of

FHE components. Providing guidelines and mechanisms for licensing and intellectual property management encourages contribution under open-source or permissive licenses.

The creation of a comprehensive library of FHE components is critically important for accelerating the development and implementation of FHE in various technological and research projects. The availability of such a library will allow developers to choose from ready-made, tested, and optimized modules, significantly reducing the time and resources required to develop new applications and systems based on FHE. The standardization of components also contributes to improved security, as they can be thoroughly tested and certified according to industry standards. In light of the above, we believe that the creation of a library of universal FHE components is extremely necessary for several reasons:

- An FHE component library provides a modular and reusable set of building blocks for FHE applications. Developers can leverage pre-built components to construct complex circuits efficiently, saving time and effort in the development process.
- A well-optimized FHE component library can significantly improve the performance of FHE computations. Optimizations, such as noise reduction techniques and efficient circuit designs, can enhance the efficiency of FHE applications and reduce computational overhead.

In section **4**, we discuss our strategy for developing a community-driven library of FHE components.

2.2.2 High computational requirements

Fully Homomorphic Encryption significantly increases computational demands compared to working with unencrypted data. The elevated complexity is necessary to ensure that computations on encrypted data remain secure, involving sophisticated mathematical operations. FHE also adds "noise" to the data, which must be carefully managed to avoid decryption errors, further increasing computational workload. To manage the noise, FHE uses a technique called bootstrapping, which is the most computationally intensive operation [BP23] but necessary for performing unlimited computations on encrypted data. This process significantly contributes to FHE's performance challenges.

In the meantime, the data encrypted with FHE is much larger than its original form, leading to more data processing for each operation and slowing down performance.

Research is ongoing to address these issues, focusing on:

- Developing new algorithms and encryption methods that are more efficient and produce less noise.
- Designing custom hardware and using Field Programmable Gate Arrays (FPGAs) to improve FHE's computational efficiency.
- Enhancing software and libraries to better use multi-threading and GPU acceleration.

Despite current limitations, the continuous development in FHE aims to reduce these performance barriers, making it more viable for widespread secure data processing applications in the future.

2.2.3 FHE standardization

A standardized approach to FHE is crucial for several reasons. Firstly, the majority of modern FHE schemes are based on Ring-Learning With Errors (RLWE), a hard mathematical problem associated with high-dimensional lattices. While RLWE is instrumental

in constructing FHE schemes, comprehending its security properties can be challenging. Standardization provides clarity and guidance on the security implications of RLWE-based schemes, enhancing understanding and confidence in their deployment. Additionally, standardization fosters interoperability between different implementations, streamlining adoption and ensuring compatibility across systems. Moreover, standardized FHE frameworks establish uniform security and performance benchmarks, promoting trust and reliability among users and stakeholders. Overall, standardization is paramount in advancing FHE technology and enabling its widespread adoption while ensuring consistency, security, and interoperability in its application.

As of April 2024, there are two major (related) efforts to standardize Fully Homomorphic Encryption: (1) the initiative led by HomomorphicEncryption.org - an open consortium of industry, government, and academia to standardize homomorphic encryption, (2) the ISO standardization effort launched by representatives of the HomomorphicEncryption.org consortium. The HomomorphicEncryption.org consortium was founded in 2017, and since its inception, the group has conducted six standards meetings aimed at advancing the standardization and adoption of homomorphic encryption technologies. Since 2018, an FHE community security standard has been available ([ACC+19]), which includes security guidelines and recommendations for working with encryption schemes based on RLWE. The ISO/IEC[6] standardization process started in April 2020 and is currently under development.

Despite obvious progress in the field of standardization, significant efforts are still required to solve various problems. These may include refining existing standards to address evolving technologies and use cases, ensuring widespread adoption and compliance of established standards across industries and regions, and continually assessing and updating standards to keep pace with emerging threats and advances in encryption technology and privacy.

Existing standardization efforts primarily focus on the security aspects and do not directly address the application aspects. However, the presence of a standard for building FHE applications could significantly contribute to the advancement of the entire field.

Another class of standardization challenges is related to many functionality levels and the complexity of data encoding methods. The problem of choosing these can be considered a component configuration problem, as the choice of a particular configuration depends on the computation circuit and can significantly affect the efficiency and accuracy of the algorithm.

A widely adopted FHE component library can establish standardization in FHE application development. It can provide a common framework for developers, enabling interoperability and fostering collaboration in the FHE community. In the forthcoming sections, we will explore methodologies for constructing such a library, alongside the challenges encountered by developers tasked with creating these solutions.

3 Layered design

Our aim is to establish an FHE ecosystem tailored for application developers. The layered strategy, detailed in [AAB+22] and utilized in the development of the OpenFHE library, offers a foundation that can be extended to build out a full FHE ecosystem. This strategy establishes a strong foundation for tackling standardization issues within FHE and for developing the FHE Components Layer. It includes designing high-level interfaces within the FHE Components Layer and handling the intricate details of various encryption schemes in the Crypto layer.

[6] ISO/IEC AWI 28033-1 Fully homomorphic encryption standard development webpage: https://www.iso.org/standard/87638.html

To develop a comprehensive FHE ecosystem, understanding the obstacles faced by developers of FHE applications is crucial. This insight is essential for devising a system that not only meets technical specifications, but is also user-friendly. Recognizing these obstacles allows us to tailor the ecosystem to be more accessible to developers, streamline the FHE application development process, and enhance the technology's overall adoption.

For the development environment to be effective, users must be able to operate the system with minimal awareness of the specific algorithmic complexities of FHE. To this end, the development environment must incorporate several key features:

- **Security Setting**: Users must be able to easily adjust security parameters to meet their specific needs, requiring the development environment to provide a straightforward way to set these parameters and customize the security level.
- **Noise Management**: Effective noise management strategies are essential to mitigate the issue of data degradation and maintain the integrity of computations. Incorporation of noise management techniques within the lower-level "crypto layer", enables addressing noise-related challenges transparently and focusing on higher-level application logic without needing to directly manage or manipulate noise accumulation. Such automated noise management can enhance the usability and effectiveness of the FHE toolchain.
- **Bootstrapping**: Bootstrapping is a crucial technique in FHE that allows for the evaluation of arbitrary functions on encrypted data. Implementing bootstrapping typically involves computationally intensive operations, making it challenging to choose where to call bootstrapping in the higher-level application logic. The bootstrapping functionality can be encapsulated within the lower-level "crypto layer". In any case, the user will continue using a high-level interface.
- **Scheme Switching**: Various FHE schemes present different compromises in terms of performance, security, and appropriateness for certain applications. Scheme switching is a crucial technique that enables the leveraging of the unique benefits offered by different schemes. This flexibility enables users to experiment with and compare different schemes without significant modifications to the higher-level layer.

The FHE ecosystem for developers must include solutions at multiple levels, covering different needs, ranging from hardware up to layers providing abstract, developer-centric solutions. The correct structuring of these layers is critically important for the development of the entire ecosystem. By determining what layers are needed, we can see where the market falls short and address those gaps. This approach allows for targeted improvements, such as adding user-friendly interfaces or better noise management tools. Filling these gaps enhances the FHE ecosystem, making it more accessible for developers and promoting wider use and integration of FHE technology.

We identify the layers depicted in the Figure 1:

- **Developer Interface**: the top-level interfaces for development and tooling built on top of Components layer. The compiler projects like Google HEIR [gooa], Google Transpiler [goob], EVA [eva], and others, reside in this layer.
- **FHE Components**: contains advanced, high-level components that can be utilized as building blocks without requiring extensive knowledge of Fully Homomorphic Encryption. Polycircuit, an FHE components library based on the FHERMA project, belongs to this layer.
- **Cryptography**: provides cryptographic primitives that enable performing computations over encrypted data. FHE libraries reside in this layer.

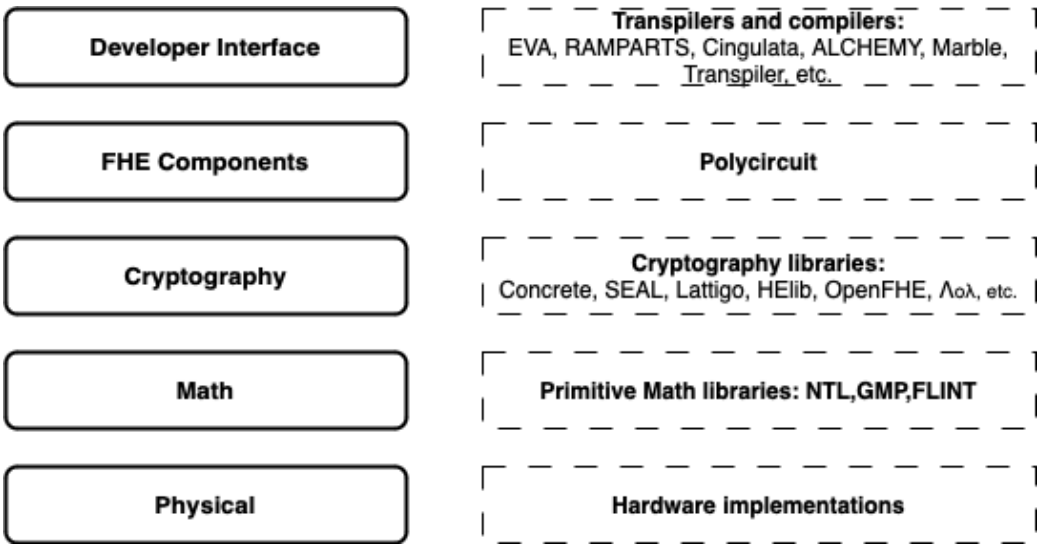

Figure 1: Multi-layer design of FHE development environment

- **Math layer**: provides the implementation of polynomial and low-level modular arithmetic. Common representatives include NTL [ntl], GMP [gmp], FLINT[fli], and the like. The math may also be implemented inside an FHE library, for instance, OpenFHE implements this layer inside the library.
- **Physical**: invokes hardware-specific math implementations under crates and interfaces. Typical examples include CPU, GPU, FPGA, etc.

To construct an ecosystem that aligns with the proposed structure of layers, it is essential to develop a library of components to be situated at the FHE Components layer. This library would serve as the cornerstone of the ecosystem, facilitating the integration of Fully Homomorphic Encryption into various applications by providing developers with a set of high-level components. Such a development is crucial for bridging the gap between complex cryptographic operations and practical application development, thereby enabling broader utilization and innovation within the FHE domain.

4 Building components library via competition

Following the design described in Section 3, we cover the challenges of building the library of FHE Components and describe our competition-based approach to this problem.

4.1 Difficulties in designing the components library

Building a library of FHE components involves a range of obstacles. Some of the issues arise from the usability of specific FHE schemes, notably those employing SIMD (Single Instruction, Multiple Data) operations, where the efficacy of these operations significantly relies on the method of data encoding. This necessity for specialized encoding introduces a significant layer of complexity in the development of FHE components, extending beyond cryptographic operations to include data management optimization for encrypted computation.

Moreover, the endeavor to create such a library without community engagement is hindered by multiple factors:

- **Limited Resources**: The extensive nature of constructing a complete FHE library demands considerable resources, including time, expertise, and financial investment, which may exceed the capacity of any single organization.

- **Multidisciplinary Expertise**: The creation of various component library components requires knowledge spanning cryptography, software engineering, and specific application domains, often beyond the reach of a single entity.

- **Rapid Advancements in FHE**: The field of FHE is marked by swift technological progress, necessitating ongoing research and development efforts to remain current.

- **Integration Issues**: Ensuring seamless interoperability among various components tailored for different applications adds another layer of difficulty.

- **Community Engagement**: Collaborating with a broader community for diverse insights and innovative solutions can significantly benefit the development process.

- **Standardization Concerns**: In the absence of comprehensive standards in FHE, ensuring the compatibility of components with both existing and forthcoming systems presents an additional challenge.

- **Standardization Needs**: The lack of standardized components in the FHE field complicates compatibility with existing and future systems, underscoring the need for establishing common standards to guide the development and interoperability of components.

These challenges underscore the complexity of developing a comprehensive FHE component library and highlight the necessity for collaborative efforts to pool resources, share expertise, and foster innovation in the development of FHE technologies.

While there is no comprehensive, unified library of FHE components available, several initiatives have developed specific components aimed at enhancing security and privacy in various applications. These initiatives cover a wide range of applications, from secure databases and private information retrieval systems to blockchain projects that incorporate FHE, as well as tools designed for privacy-preserving machine learning, which safeguard data during analysis. These projects are directed at tackling distinct challenges associated with FHE, offering specialized tools and features for different types of encrypted data processing tasks. They showcase the ongoing efforts to make FHE more accessible and applicable across a variety of fields. A selection of these projects is listed in Table 4.1, and the detailed compilation of FHE resources can be found at the provided link[7].

Table 3: Subset of application-specific libraries

Project name	Citations	Domain	FHE implementation used
blyss	[MW24]	private information retrieval	[MW22]
CryptDB	[PZB15]	SQL queries	Paillier crypto system
HEMat	[JKLS18]	matrix algorithms	HEAAN
Rosetta	[WGC18]	TensorFlow opcodes	CKKS
Spark FHE	[HAC+20]	distributed data access	SEAL, HElib
TenSEAL	[BRCB21]	operations on tensors	SEAL

[7] https://github.com/jonaschn/awesome-he?tab=readme-ov-fileapplications

4.2 FHERMA: Platform for building the components library via competitions

Addressing the challenges identified in Section 4.1, especially regarding the development of a comprehensive library of components, necessitates extensive community involvement. For this goal, we have developed the FHERMA platform for FHE challenges. The main goal of the project is to develop an open-source library of FHE components. Such a library can significantly simplify application development and accelerate the adoption of FHE. The initial focus of the platform is on components for Machine Learning and Blockchain applications. The FHERMA platform aims not only to overcome the technical challenges associated with FHE, but also to build a vibrant community of practice. By engaging a wide array of participants—from academic researchers and industry professionals to hobbyists—the platform seeks to harness collective intelligence and innovation. This approach is essential for tackling the complex problems inherent in FHE and for accelerating the translation of theoretical advances into practical, deployable solutions.

- **Community Collaboration:** The platform encourages participants to collaborate, share insights, and contribute to each other's work, fostering a rich environment of peer learning and mutual assistance. This community aspect is facilitated through forums, collaborative tools, and regular webinars where participants can discuss challenges and share best practices.
- **Educational Resources:** To lower the entry barrier for new participants and to enhance the skills of existing users, FHERMA provide a range of educational resources. These include tutorials, detailed documentation of the FHE components, and case studies that showcase successful applications of the technology.
- **Continuous Development:** The platform is continuously updated with new challenges and features, reflecting the latest research and user feedback. This ongoing development ensures that FHERMA remains relevant and effective in addressing the evolving needs of the FHE community.
- **Incentive Mechanisms:** Recognizing the importance of motivation and reward, FHERMA includes incentive mechanisms such as bounties for challenge winners, certificates, and recognition in community highlights. These incentives not only reward contributions but also spur ongoing participation and competition.

The role of FHERMA extends beyond just a platform for competition; it acts as a catalyst for advancing FHE technology by providing a structured, supportive, and stimulating environment. The integration of multiple encryption schemes and the diverse array of challenge types are strategically designed to cultivate a broad spectrum of solutions that could be beneficial for numerous applications across different sectors.

There are two main types of challenges on the platform:

1. **Black Box Challenges:** Participants are tasked with creating solutions based on specified requirements and submitting only the serialized final ciphertext. These challenges focus on the output's accuracy and the efficiency of the solution, without the need to reveal the solution's underlying code or methodology.
2. **White Box Challenges:** These challenges require the submission of source code for performance evaluation, aiming to identify the most efficient solutions. The platform assesses these submissions for performance and accuracy, keeping the solutions confidential and inaccessible to other participants.

Participants develop algorithms/components, which are then tested on the platform.

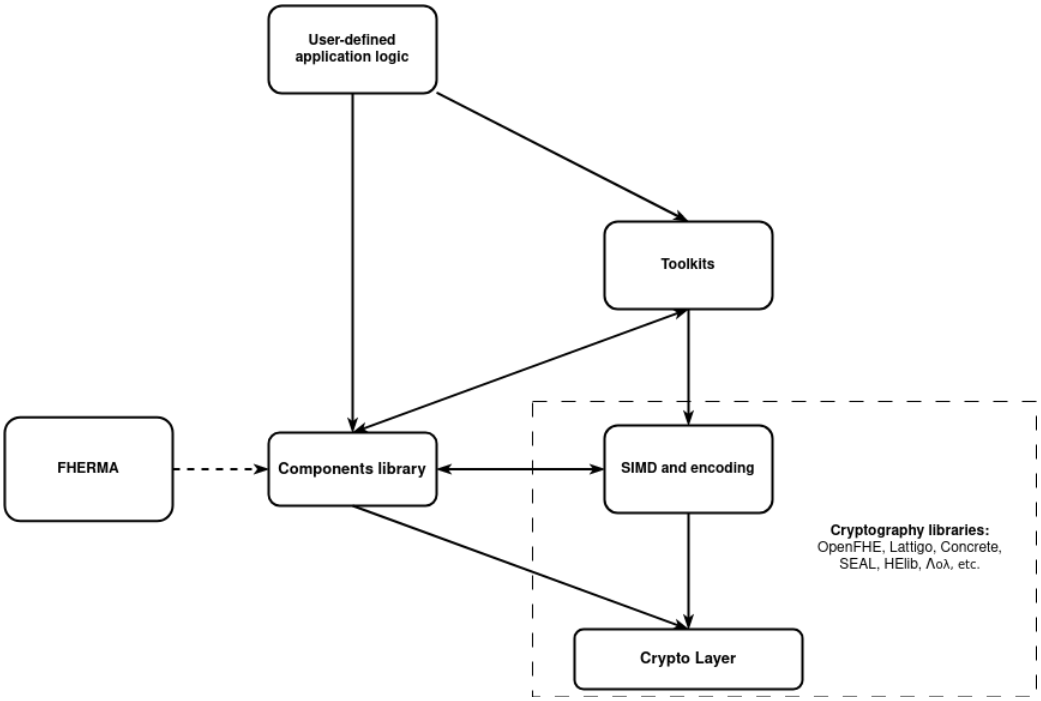

Figure 2: Building the library via FHE Challenges

One of the unique features of FHERMA is that for the Black Box challenges participants can check the quality of their solutions without revealing their details. They process the test cases with their algorithm, upload the resulting ciphertext to the platform, and the platform evaluates the solution based on parameters like accuracy and required multiplicative depth. The platform provides a dynamic ranked leaderboard, fostering continuous competition among participants to improve their solutions. Some benefits and features of FHERMA include:

- **Fast Evaluation:** Solutions are evaluated within seconds, eliminating the need for participants to wait for months for verification, as seen in traditional bounty programs.
- **Transparency and Fairness:** The evaluation process is fully automated, eliminating the human factor. Additionally, all test cases and secret keys to the encrypted input data are published at the end of the challenge, enabling participants to verify the correctness of assessments.
- **Dynamic Leaderboard:** Participants can monitor their ranking online and aim to improve their solutions if it is needed.
- **Black Box Challenges:** FHERMA is the first platform that provides this type of challenge.

The platform provides the opportunity to create challenges based on a variety of encryption schemes, including CKKS, BFV, BGV, DM/FHEW, and CGGI/TFHE.

For some challenges, it may not be obvious which encryption scheme should be chosen. In such cases, the platform supports a flexible option to create challenges where the implementation can be done using one of several encryption schemes.

The adaptable architecture for challenge development is designed to accommodate the diversity of usage scenarios and task specifications in use cases requiring homomorphic encryption. The role of the FHERMA platform within the overall framework is illustrated in Figure 2.

5 Concluding remarks

Although the FHERMA project is a relatively new effort, the initial challenges have already demonstrated that a competitive approach can produce efficient, practically useful FHE components. The initial challenges of the project included Matrix Multiplication, Sign Evaluation, and Logistic Function. The winning solutions for these challenges are published as part of the open-source Polycircuit library [Fai]. The library will be continuously updated and expanded with winning solutions for current and future challenges on the FHERMA project.

We aim to add support for new FHE libraries, for example, the support for Lattigo is already added for selected new challenges. The platform will provide participants with the option to engage in challenges both individually and collaboratively as teams. We will add more details to this document as the FHERMA project advances further.

References

[AAB+22] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.

[AAB+23] Ehud Aharoni, Allon Adir, Moran Baruch, Nir Drucker, Gilad Ezov, Ariel Farkash, Lev Greenberg, Ramy Masalha, Guy Moshkowich, Dov Murik, Hayim Shaul, and Omri Soceanu. HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data. *Privacy Enhancing Technology Symposium (PETs) 2023*, 2023.

[ACC+19] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption standard. Cryptology ePrint Archive, Paper 2019/939, 2019. https://eprint.iacr.org/2019/939.

[ATD+19] David W. Archer, Jose Manuel Calderon Trilla, Jason Dagit, Alex J. Malozemoff, Yuriy Polyakov, Kurt Rohloff, and Gerard Ryan. Ramparts: A programmer-friendly system for building homomorphic encryption applications. Cryptology ePrint Archive, Paper 2019/988, 2019. https://eprint.iacr.org/2019/988.

[BGPG20] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, and Shafi Goldwasser. Secure large-scale genome-wide association studies using homomorphic

encryption. Cryptology ePrint Archive, Paper 2020/563, 2020. https://eprint.iacr.org/2020/563.

[BGV11] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Paper 2011/277, 2011. https://eprint.iacr.org/2011/277.

[BP23] Ahmad Al Badawi and Yuriy Polyakov. Demystifying bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/149, 2023. https://eprint.iacr.org/2023/149.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. Cryptology ePrint Archive, Paper 2012/078, 2012. https://eprint.iacr.org/2012/078.

[BRCB21] Ayoub Benaissa, Bilal Retiat, Bogdan Cebere, and Alaa Eddine Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption, 2021.

[BZZ+23] Song Bian, Zian Zhao, Zhou Zhang, Ran Mao, Kohei Suenaga, Yier Jin, Zhenyu Guan, and Jianwei Liu. Heir: A unified representation for cross-scheme compilation of fully homomorphic computation. Cryptology ePrint Archive, Paper 2023/1445, 2023. https://eprint.iacr.org/2023/1445.

[CDS14] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: a compilation chain for privacy preserving applications. Cryptology ePrint Archive, Paper 2014/988, 2014. https://eprint.iacr.org/2014/988.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Paper 2016/870, 2016. https://eprint.iacr.org/2016/870.

[CKKS16] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. Cryptology ePrint Archive, Paper 2016/421, 2016. https://eprint.iacr.org/2016/421.

[CKY18] Jung Hee Cheon, Andrey Kim, and Donggeon Yhee. Multi-dimensional packing for HEAAN for approximate matrix arithmetics. *IACR Cryptol. ePrint Arch.*, page 1245, 2018.

[CLP17] Hao Chen, Kim Laine, and Rachel Player. Simple encrypted arithmetic library - seal v2.1. Cryptology ePrint Archive, Paper 2017/224, 2017. https://eprint.iacr.org/2017/224.

[CP15] Eric Crockett and Chris Peikert. $\lambda \circ \lambda$: Functional lattice cryptography. Cryptology ePrint Archive, Paper 2015/1134, 2015. https://eprint.iacr.org/2015/1134.

[CPS18] Eric Crockett, Chris Peikert, and Chad Sharp. Alchemy: A language and compiler for homomorphic encryption made easy. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[DKS+19] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madanlal Musuvathi. Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation, 12 2019.

[DM14] Léo Ducas and Daniele Micciancio. Fhew: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Paper 2014/816, 2014. https://eprint.iacr.org/2014/816.

[eva] EVA. https://github.com/microsoft/EVA.

[Fai] Fair Math. Polycircuit - fhe components library. https://github.com/fairmath/polycircuit.

[fli] FLINT. https://flintlib.org/.

[FSB+23] Jordan Frery, Andrei Stoian, Roman Bredehoft, Luis Montero, Celia Kherfallah, Benoit Chevallier-Mames, and Arthur Meyre. Privacy-preserving tree-based inference with fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/258, 2023. https://eprint.iacr.org/2023/258.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[gmp] GMP. https://gmplib.org/.

[gooa] Google HEIR Project. https://github.com/google/heir/commit/b85bd76a0205fade51d1517aa6657c4e798f9658.

[goob] Google Transpiler. https://github.com/google/fully-homomorphic-encryption/blob/main/transpiler/README.md.

[GSPH+21] Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce Wilson, Asra Ali, Eric P. Astor, Itai Zukerman, Sam Ruth, Christoph Dibak, Phillipp Schoppmann, Sasha Kulankhina, Alain Forget, David Marn, Cameron Tew, Rafael Misoczki, Bernat Guillen, Xinyu Ye, Dennis Kraft, Damien Desfontaines, Aishe Krishnamurthy, Miguel Guevara, Irippuge Milinda Perera, Yurii Sushko, and Bryant Gipson. A general purpose transpiler for fully homomorphic encryption. Cryptology ePrint Archive, Paper 2021/811, 2021. https://eprint.iacr.org/2021/811.

[HAC+20] Peizhao Hu, Asma Aloufi, Adam Caulfield, Kim Laine, and Kristin Lauter. Sparkfhe: Distributed dataflow framework with fully homomorphic encryption. Unknown, 2020. https://ppml-workshop.github.io/ppml20/pdfs/Hu_et_al.pdf.

[HS20] Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. Cryptology ePrint Archive, Paper 2020/1481, 2020. https://eprint.iacr.org/2020/1481.

[JKLS18] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. Cryptology ePrint Archive, Paper 2018/1041, 2018. https://eprint.iacr.org/2018/1041.

[KJT+20] Tsung-Ting Kuo, Xiaoqian Jiang, Haixu Tang, Xiaofeng Wang, Tyler Bath, Diyue Bu, Lei Wang, Arif Harmanci, Shaojie Zhang, Degui Zhi, Heidi Sofia, and Lucila Ohno-Machado. idash secure genome analysis competition 2018: blockchain genomic data access logging, homomorphic encryption on gwas, and dna segment searching. *BMC Medical Genomics*, 13, 07 2020.

[MSM+22] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H.P. Fitzek, and Najwa Aaraj. Survey on fully homomorphic encryption, theory, and applications. Cryptology ePrint Archive, Paper 2022/1602, 2022. https://eprint.iacr.org/2022/1602.

[MTPBH20] Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. Cryptology ePrint Archive, Paper 2020/304, 2020. https://eprint.iacr.org/2020/304.

[MW22] Samir Jordan Menon and David J. Wu. Spiral: Fast, high-rate single-server pir via fhe composition. Cryptology ePrint Archive, Paper 2022/368, 2022. https://eprint.iacr.org/2022/368.

[MW24] Samir Jordan Menon and David J. Wu. Ypir: High-throughput single-server pir with silent preprocessing. Cryptology ePrint Archive, Paper 2024/270, 2024. https://eprint.iacr.org/2024/270.

[ntl] NTL. https://libntl.org/.

[PZB15] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. Guidelines for using the cryptdb system securely. Cryptology ePrint Archive, Paper 2015/979, 2015. https://eprint.iacr.org/2015/979.

[Sch] Jonathan Schneider. Awesome - a curated list of amazing homomorphic encryption libraries, software and resources. https://github.com/jonaschn/awesome-he.

[VS18] Alexander Viand and Hossein Shafagh. Marble: Making fully homomorphic encryption accessible to all. In Michael Brenner 0003 and Kurt Rohloff, editors, *Proceedings of the 6th Workshop on Encrypted Computing Applied Homomorphic Cryptography, WAHC@CCS 2018, Toronto, ON, Canada, October 19, 2018*, pages 49–60. ACM, 2018.

[WGC18] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: Efficient and private neural network training. Cryptology ePrint Archive, Paper 2018/442, 2018. https://eprint.iacr.org/2018/442.

[Zam22] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.