Towards Real-World Private Computations with Homomorphic Encryption: Current Solutions and Open Challenges

Michela Iezzi[1(✉)], Carsten Maple[2], and Andrea Leonetti[1]

[1] Banca d'Italia, Rome, Italy
michela.iezzi@bancaditalia.it
[2] The Alan Turing Institute, London, UK
cmaple@turing.ac.uk

Abstract. There is an increasing need to share sensitive information within and beyond organisations. Protecting this information is vital for commercial and regulatory reasons. Homomorphic Encryption (HE) has come to the fore as a mechanism to enable the sharing of confidential data in a secure and private manner. Multiple open-source libraries are now publicly available, providing organisations with the tools to utilise the advantages of HE. While research devoted much effort to the academic and cryptographic aspects of HE schemes, research explicitly focusing on real-world financial applications is comparably rare. There is a need to provide a comparative analysis and related benchmarking of the most suitable HE libraries, having fixed the functional and non-functional requirements of the enterprise application of interest. We consider the motivation and background for HE and discuss the most promising open-source HE libraries. Having introduced real-world use cases in a financial context, we then illustrate outstanding challenges and how we plan to circumvent open points, introducing HELT (Homomorphic Encryption Libraries Toolkit).

Keywords: Homomorphic Encryption · Private computation · Real-world applications

1 Introduction

Financial institutions create, process, and control significant amounts of data. This data can have significant value, especially when combined with other sources or types of data. Such sources include other financial institutions and service organisations, government and regulatory bodies, and other units within their own organisation. However, sharing such data is not always possible for legal,

The views and opinions expressed in this paper are those of the authors and do not necessarily reflect the official policy or position of Banca d'Italia.

S. Katsikas et al. (Eds.): ESORICS 2023 Workshops, LNCS 14398, pp. 283–298, 2024.
https://doi.org/10.1007/978-3-031-54204-6_17

regulatory or commercial reasons. As such, methods are required that allow the use and analysis of this data in a confidential manner. Homomorphic Encryption (HE) is a promising approach to the problem of computation of confidential data. Since the introduction of Fully Homomorphic Encryption (FHE) in 2009 [13], much effort has been devoted to optimising the drawbacks brought by the enhancements introduced by the work of Gentry. In this direction, the academic community has developed many libraries to test and allow HE use. The first versions of these libraries were not ready for enterprise and real-world applications. HE realises not only the protection of data *in-transit* or *at-rest* but also *in use*. It represents paramount progress in data analysis due to the increasing usage of personal data and its applications in everyday life. Moving to a more specific context, we can observe how financial institutions and services have access to structured and unstructured personal data due to their varied duties. Furthermore, fintech companies and public institutions could also use public cloud services. It appears more evident that legal instruments, e.g., Non-Disclosure Agreements (NDA) [18], are not protective enough for data owners and financial counterparties. There is a demand for more stable and sound technology solutions to the problem of confidentiality of personal data, such as PETs (Privacy Enhancing Technologies). Of course, HE is only one of the available PETs on the market. We focus on this cryptographic technique taking into account two different motivations: (i) many PETs fail with the so-called *privacy-utility trade-off*: an amount of leakage about confidential data should be accepted to obtain valuable results from the computation [27]; (ii) the recent and consistent investments in the HE field in the last years [20].

Despite attempts to make HE libraries more user-friendly, building enterprise-ready applications that act on homomorphically encrypted data still requires a range of technical experts: (i) the *data analyst*, which could either be a data scientist or an artificial intelligence expert, or a (typically untrusted) researcher that may be external to the organisation; (ii) the *software developer*, which helps in the integration of the new application in the already existing legacy environment, provided all the functional and non-functional constraints; (iii) the *cryptographer*, which is pivotal to choose the suitable library and to choose the right HE context, that as we will see in the remainder of the paper, is not an easy task for a non-cryptographer. Moreover, new and existing enterprise applications should be translated into a HE-friendly format to support the computation. It is not easy to make these three actors work together. Furthermore, an expert skilled in all these aspects is yet to be available in the market. Enterprises need a solution to fix this gap, at least partially.

This paper conveys the idea that it is necessary to build open-source benchmarking tools that allow understanding of some crucial issues:

- if the existing HE libraries are usable for the data analyst;
- which library fits a given application;
- what means integrating them with the existing enterprise environment.

We propose a two-step approach for adopting HE in an enterprise context to reach this objective. The first step is providing the data analyst with encrypted confidential data and an HE environment with selected open-source libraries. The cryptographer will guide the analyst in building the proper analysis in a HE-friendly format. A phase of control of the results by the Data Owner may be envisaged to avoid re-identification due to analyses aimed at extracting confidential data. The second step will provide an HE toolbox where some standard functions, e.g., the primitive blocks of statistical computations or some simple machine learning or Natural Language Processing tools, are implemented, given a chosen library, and then exposed to the data analyst, apart from a set of parameters. This step should be carefully implemented, considering the functional and non-functional requirements of the involved actors. The choice of parameters is challenging because the HE context highly depends on the target application and the employed data. The goal is to provide a set of rules and predefined parameters. This paper is a starting point for analysing some available HE libraries. It represents a short work-in-progress report for developing a playground where the data analysts, helped by the cryptographers in the initial phase, can experiment with which library is more suitable for the selected application.

Outline. The remainder of the paper is organised as follows. Section 2 describes the industrial context we are touching. Section 3 introduces a proper background for HE. In Sect. 4, we describe the main libraries, giving a valuable overview of implemented schemes and their parameters. Section 5 discusses the requirements for a possible HE benchmarking tool. In Sect. 6, we review the main results available in the literature. Finally, Sect. 7 concludes the paper and gives directions for future work.

2 Industrial Context

Although HE has been successfully employed in medicine [33], we focus on the financial institutions' context. Indeed, governments and institutions have access to confidential data for their purposes. Among institutions, we will consider the case of a central bank, which owns different confidential assets, like datasets about companies, balance sheets of banks and intermediaries, payment systems data, suspicious transactions data, and many others. These datasets are instrumental in ensuring that a central bank acts, e.g., as the supervisory authority or is proficient in economic research, collaborating with academia and other statistics institutions. Each dataset has a different data owner who is the only one in charge of manipulating and analysing the data. The possibility of collaboration between external actors, such as data scientists or academic researchers, and the sharing of confidential assets with other institutions and academia are challenging. It may undergo ad-hoc remote processing systems [4] or anonymisation processes [7]. Furthermore, the confidential nature of the data assets cannot fully unravel the potential of public cloud computing. In Fig. 1, the depicted use cases are locked due to the confidential nature of the data. We can deduce that, in

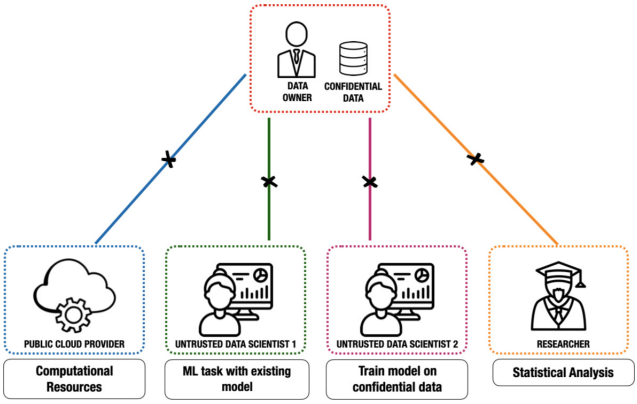

Fig. 1. Locked enterprise use cases due to privacy concerns.

the financial context, two are the main scenarios that ask for a solution based on HE [17,32], as in Fig. 2:

- *private outsourcing computation*, where an institution may decide to use public cloud resources to execute computational intensive tasks on confidential data;
- *privacy preserving data science*, where an external actor (that may be a machine learning expert, a researcher from another institution, or a data scientist from the same institution that is not the data owner) has to perform tasks like, e.g., Private Prediction as a Service (PPaaS), Private Training as a Service (PTaaS), or statistical analysis on confidential data.

Moreover, the use of HE in this second scenario protects confidential data from untrusted third parties and the intellectual property of the algorithm developed by the external actor [5]. Public cloud resources could enable this scenario, especially in the case of PTaaS. We may deal with these different situations in these three scenarios: (i) encrypted data, plain model, (ii) encrypted data, encrypted model, and (iii) multiple encrypted data from different data owners. Multikey Homomorphic Encryption could enable the last one [21]. With their different use cases, these scenarios can represent practically relevant HE applications in our industrial context. Sharing a statistical dataset ensuring confidentiality requires fixing functional and non-functional requirements that enable a wise choice of the HE library to support private computation. In the case of private computation with HE, confidential data will undergo a new data-processing pipeline: as we will see in Sect. 3, working with homomorphically encrypted data requires a phase of Encoding before Encryption; the classical Extract-Transform-Load (ETL) process should be complemented with these two phases before sharing data between counterparties. The design of these phases is closely tied to the nature of data (e.g., textual or numeric, structured or unstructured) and the application we need to deploy: having a benchmarking environment will help

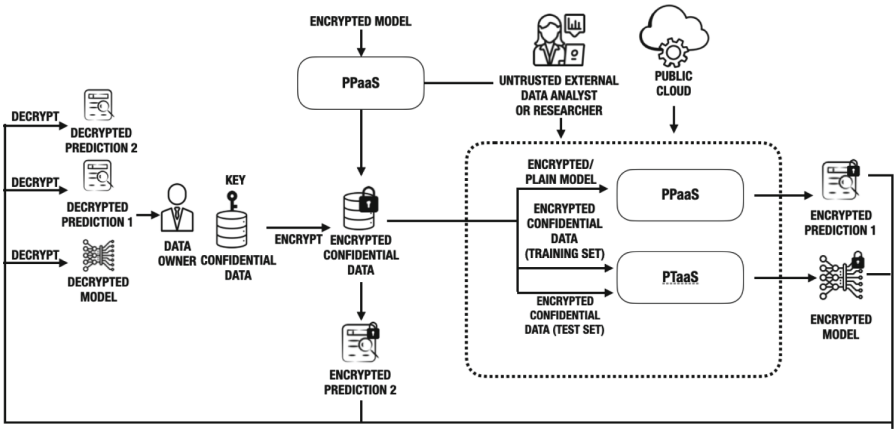

Fig. 2. Private Prediction as a Service (PPaaS) and Private Training as a Service (PTaaS)

choose the most appropriate HE library for each application. In Table 1, we gathered the requirements from the point of view of the three actors that are involved in our design process, as seen in 1: (i) the data analyst, (ii) the data owner, and (iii) the software developer. From this requirements gathering, we will exclude the cryptographer that will set up the HE test environment and assist these actors in the application deployment.

Table 1. Requirements for private computation with Homomorphic Encryption.

Actor	Requirement
Data Analyst	Usability
	Documentation and toy examples
	Parameters setting
	Programming language
	Ease of data preparation phase
	Confidentiality of models
Data Owner	Data confidentiality
	Correctness of results
	Performance
	Cloud/on-premise solution
Software Developer	Integration in the enterprise environment
	Code maintenance and upgrade

3 Background

We introduce the main results of HE theory to pose a common language for the remainder of the paper. The interested reader may find an in-depth discussion in [1,22].

A Primer. Homomorphic Encryption (HE) allows computation on encrypted data by employing an *homomorphism*, a structure-preserving map between two algebraic groups, i.e., the plaintext group P and the ciphertext group C.

For an asymmetric HE encryption scheme, it holds:

$$Encr(pk_{encr}, p_1 \diamond p_2) = Encr(pk_{encr}, p_1) \circ Encr(pk_{encr}, p_2) = c_1 \circ c_2 \quad (1)$$

where $Encr(\cdot)$ is the encryption function, $\diamond$ and $\circ$ are defined over the group P and C.

Equation 1 means that computing $\diamond$ over plaintexts p_1 and p_2 is equivalent to computing $\circ$ on related ciphertexts c_1 and c_2.

HE schemes are required to work with complex functions, not just single operations. Beyond key generation, encryption, and decryption functions, the HE scheme defines an *evaluation function* as follows, $Eval(pk_{eval} f, [c_1, c_2])$, where the function f should belong to the set of the admissible functions for that particular scheme to be correct:

$$p_1 \diamond p_2 = Decr(sk, c_1 \circ c_2) \quad (2)$$

To this end, the function f should be polynomial, i.e., expressible as a combination of multiplications and additions; in other words, the function f has to be *homomorphic-ready*. For practical applications of our interest, such as machine learning tasks or statistical analysis, this is not to be taken for granted: many approximation techniques for complex f have been explored, such as (i) Taylor expansion series approximation, (ii) Chebychev polynomials, (iii) look-up table, (iv) least squares.

Lattice and Ring-Based Cryptography. The security of HE schemes relies on the hardness of known problems on lattices that act as the basis for their construction. Learning With Errors (LWE) and Ring Learning With Errors (RLWE) are the most relevant.

Roughly, the encryption operation can be seen as adding random noise to the plaintext, and the decryption operation is recovering the plaintext after filtering the noise using the secret key. The random noise grows as the number of operations increases. When the number of multiplications exceeds a fixed depth, the noise becomes a burden for correct decryption that cannot be ensured anymore. Noise growth is typically associated with a noise budget that depends on the type of scheme. Furthermore, an unwanted expansion in the ciphertext and the key size happens during computation. Moreover, the hardness of these problems contributes to the computational complexity of the HE scheme.

Bootstrapping and Optimizations. When the depth of the function to be homomorphically evaluated is not known a priori, e.g., if we are dealing with a

neural network, LHE is not enough, and *bootstrapping*, introduced by Gentry in 2009 [13], is the instrument to efficiently transform an LHE scheme with certain properties to an FHE scheme.

The *bootstrapping* operation reduces noise and allows correct decryption for any function. A second level of homomorphic encryption with a noise budget greater than the first level is applied to the ciphertext. Then, it is possible to decrypt the ciphertext with respect to the first encryption key, removing accumulated noise and restoring a new noise budget. The sufficient condition to apply bootstrapping to an LHE scheme is that the decryption circuit of the latter is included in the set of admissible functions to be evaluated. Various optimisations have been proposed to overcome the described inherent limits; we will find them optional in many libraries:

- *modulus-switching*, that prevents the ciphertext from growing without control.
- *key-switching*, that enables the revert of the new secret key to the original secret key, decreasing the size of the ciphertext.
- *relinearization*, that reduces the size of the ciphertext modulus.

These optimisation techniques mostly manage noise in LHE schemes and are employed when the depth of the circuit to be evaluated is known in advance. At the same time, bootstrapping cannot be avoided in all other use cases.

HE Schemes. HE schemes are classically classified into four main classes, depending on the type and the number of elementary operations. For real-world applications, we are mainly interested in (i) *Leveled Homomorphic Encryption (LHE)* schemes, where additions, multiplications, and their combination are allowed but only up to a fixed number, and (ii) *Fully Homomorphic Encryption (FHE)* schemes, where both addition and multiplication are allowed, without limits on the number of operations.

Various HE schemes have been implemented in HE libraries, mainly based on LWE or RLWE problems. One of the first is the BGV scheme that works over the integers, based on the LWE problem, and it is characterised by noise and ciphertext growth. Other popular schemes are, e.g., (i) CKKS which allows working with real numbers, and (ii) TFHE, which is based on the hardness of LWE assumption over the torus and provides fast bootstrapping.

Usually, LHE schemes are more practical and ensure a good trade-off between computational complexity, performance time, and privacy preservation. This is true in all the use cases characterised by the fixed depth of the computation.

Encoding Raw Input Data. Input data to be encrypted could have many different formats, highly dependent on the chosen HE scheme. Batching allows the packing of many plaintexts into a single ciphertext, enabling parallel homomorphic computation and slot-wise operations in a SIMD fashion. Each value is independently encoded in a slot inside an array. Binary representation of plaintexts is often available. More sophisticated encoding methods can be found for schemes based on the TFHE scheme, where binary or real values for each slot are taken on the torus.

4 Available Libraries

This section presents an overview of the HE libraries we will compare. We choose the following seven libraries considering various features, as stated in Table 1. We excluded from our analysis wrappers or compilers; we aim to explore HE libraries already employed in the research community [17]. All the libraries are open-source since they are still at a research level. These libraries are addressed to a developer with advanced cryptographic skills: defining a cryptographic context where HE parameters for the selected scheme are chosen is necessary. Table 2 and 3 summarise supported operations and main features for each library, except for OpenFHE, that we plan to include in our future work.

4.1 HElib

HElib is an open-source C++ library developed by IBM and the Algorand Foundation. Two schemes are available: leveled and fully BGV for integers and leveled CKKS for real and complex numbers. Packing enables the construction of an LHE scheme and an FHE with bootstrapping where available. Another type of representation is the binary one, which is used only for the BGV scheme and could be employed for FHE and LHE.

The encryption context creation requires setting many parameters, which are inherently tight to the inner logic of the library; even if in HElib scripts and utilities are available to help the developer, this choice is challenging for a non-cryptographer. This is true especially for the BGV scheme, while CKKS default parameters are present. It is worth remarking that in HElib, noise and time of execution are two parameters that need to be optimised, and typically the size and the depth of the SIMD circuits are descriptive of these constraints. We first fix the bound on the noise, i.e., the depth of the circuits, and then optimise the running time. Noise is also tightly related to the security level, which is recommended to be fixed at 128 bits.

4.2 SEAL

SEAL [28] is an open-source C++ library developed at Microsoft. The available schemes are BFV for integers and CKKS for real values. SEAL provides only the leveled mode, so the type of computation should be known in advance to balance the encoding and encryption parameters correctly. The encoding happens in two ways, depending on the selected HE scheme. A batch encoder is employed for the BFV scheme, while for the CKKS scheme, a CKKS encoder is implemented. The encryption context creation usually needs three parameters, which also influence the encoding phase. Two parameters are common to both BFV and CKKS schemes and alter both noise budget and performance, and the third parameter is peculiar to the chosen scheme. Contextually to the encryption context, the SEAL library creates a modulus switching chain, a set of encryption parameters derived from the original ones, which improves performance and communication cost in the BFV scheme. In the CKKS scheme, the modulus switching prevents

Table 2. Admitted operations for each library. CT stands for ciphertext, PT stands for plaintext.

Library	HE Scheme	CT + PT	CT + CT	CT * PT	CT * CT	Column shift	Row shift	Binary circuit
HElib	BGV bin.	✗	✓	✗	✓	✓	✗	✓
	BGV pack	✓	✓	✓	✓	✓	✗	✗
	CKKS	✓	✓	✓	✓	✓	✗	✗
SEAL	BFV	✓	✓	✓	✓	✓	✓	✗
	CKKS	✓	✓	✓	✓	✓	✗	✗
PALISADE	BGV	✓	✓	✓	✓	Rotate	Rotate	✗
	BFV	✓	✓	✓	✓	✗	✗	✗
	CKKS	✓	✓	✓	✓	Rotate	Rotate	✗
	FHEW	✗	✗	✗	✗	✗	✗	✓
	RGSW	✗	✓	✗	✓	✓	✗	✓
Concrete	LWE	✓	✓	✓	External product	Rotate	Rotate	✓
	GLWE	✓	✓	✓	External product	Rotate	Rotate	✓
LATTIGO	CKKS	✓	✓	✓	✓	Rotate	Rotate	✗
	BGV	✓	✓	✓	✓	Rotate	Rotate	✗
	BFV	✓	✓	✓	✓	Rotate	Rotate	✗

noise growth. A valuable feature of SEAL is the automatic parameter selection, which helps the user fix the parameters based on the state-of-the-art attacks against RLWE.

4.3 PALISADE

PALISADE [19] was born as a Sponsored Project of NumFOCUS, a nonprofit charity in the United States, and has the contributions of various cryptographers and developers coming from, e.g., Duality Technologies or the HE community. It is an open-source C++ library built on lattice-based cryptography. It is designed to be modular and extensible, providing a well-documented codebase and a transparent system to choose parameters, encryption schemes, and data encoding methods. The various features are offered in terms of capabilities that the user has to enable, e.g., Encryption, SWHE, and LHE; for each of these capabilities, the list of available schemes and related operations are given, as summarised in Table 2. Various representations for polynomials are available; the recommended one is DCRTPoly, where the Chinese Remainder Theorem format represents polynomial coefficients. Various encoding types are available in PALISADE, like Integer, Fractional, or Packed encoding. The creation of context depends on the enabled capability and related HE scheme, and it requires the choice of ring dimension, multiplicative depth, and batch size for schemes like BGV, BFV, and CKKS.

Table 3. HE main features.

Library	Lang	Multithread	SIMD	Multi-key	Mod Switch	Key Switch	Bootstr	Relin
HElib	C++	✓	✓	✗	✓	✓	BGV	✓
SEAL	C++	✓	✓	✗	✓	✗	✗	✓
PALISADE	C++	✓	✓	✗	✓	✓	✓	BGV
TFHE	C++	✓	✗	✗	✗	✓	✓	✗
Concrete	Rust	✗	✓	✗	✗	✓	✓	✗
Lattigo	Go	✗	✗	✓	✓	✓	CKKS	✓

4.4 OpenFHE

The PALISADE project is converging into the OpenFHE open-source project [6], which is supported by DARPA and has as contributors many of the leading developers of the major open-source HE libraries. It is a C++ library that supports the most useful HE schemes and is based on the hardness of the RLWE problem.

The available schemes are (i) BGV and BFV for modular arithmetic over finite fields, (ii) CKKS for vectors of real and complex numbers, (iii) DM, and CGGI, for boolean circuits and decision diagrams. SIMD packing is available for both vectors of integers and real numbers. This library foresees bootstrapping for all HE schemes in future versions. Each scheme runs in AUTO and MANUAL modes to overcome inherent difficulties in setting parameters and applying optimisations such as modulus/key switching, rescaling for CKKS, or bootstrapping. Integration with the existing compilers is easy to obtain. Multiple backends, e.g., GPU and FPGA, provide hardware acceleration support. It also enables multi-party versions of such HE schemes.

We plan to include OpenFHE in our future benchmarking tool.

4.5 TFHE

TFHE (Fast Fully Homomorphic Encryption Library over the Torus) [11] is a C/C++ library, which improves the bootstrapping time to 0.1 s, as in [10,12]. It implements a generalisation of LWE and RLWE on the Torus.

TFHE provides both leveled and bootstrapping modes; bootstrapping is executed after every boolean operation. The inputs are integer values, and the user decides the precision of the plaintext representation, i.e., the number of bits. The key management is facilitated since only two keysets are required: (i) the Secret Keyset, which is used to encrypt and decrypt confidential data symmetrically, and (ii) the Cloud Keyset, which is used to perform computation and bootstrapping on the ciphertexts. The user has to rewrite the required computation as a boolean circuit using the available binary gates. The encryption context is automatically created by specifying the security level and using the related default parameters.

4.6 Concrete

Concrete [35] is an open-source Rust library developed at Zama. It implements a discretised TFHE version for machine learning tasks and neural networks. This library allows keeping invariant the topology of the original neural network to be evaluated without the need to change it to make it more homomorphic-friendly, as mentioned in Sect. 2. The key enhancement is Programmable Bootstrapping (PBS), which permits the computation of any function, even the non-linear ones, during the bootstrapping phase, resetting the noise simultaneously. PBS is possible if the function can be decomposed and expressed as a linear combination of univariate functions using methods like Ridge decomposition or Kolmogorov superposition theorem. The encoding feature in Concrete is different from TFHE: each array slot is encoded as real Torus elements modulo 1, i.e., real numbers between 0 and 1. Creating an encryption context in Concrete means choosing parameters like the dimension of the vector of integers (LWE) or polynomials (RLWE) and the standard deviation of the noise distribution added to the body value; these two parameters influence the computation time, the ciphertext overhead, and the number of bits of precision that remains available. Recently, Concrete has been enriched with a module on the top of the Concrete library, Concrete ML, to enable machine learning-based applications.

4.7 LATTIGO

LATTIGO [26] is an entirely Go-based HE library developed at EPFL and based on RLWE schemes. The Go programming language has many valuable features: it works well with concurrent systems and is as efficient as C++, despite being more accessible to code. The encryption context creation is easily obtained using default parameters available for 128-bit of security. The ability to efficiently deal with concurrency makes Lattigo one of the HE libraries that offers multi-party computation (MPC) for both implemented schemes, BFV and CKKS, taking in input both integers and real values, with packed representation. Another unique characteristic is that bootstrapping is available for CKKS.

5 Towards Real-World HE Applications: HELT

There are many challenges in developing real-world HE applications starting from scratch. Apart from well-known HE issues like computational costs and performance concerns, more support tools for developers and ready-to-use frameworks for data analysts are needed. Many efforts are currently devoted to providing user-friendly APIs to develop privacy-preserving data science applications quickly, and the vendors are trying to satisfy this enterprise requirement. Notable examples are the IBM Security HE Services or Zama's roadmap for Concrete [16,34].

Another trend is the development of HE Compilers, which should provide high-level functions that prevent the user from working with HE parameters

setting or operations on cyphertexts [31]. However, compilers or DS-ready environments are not general-purpose. Another critical aspect is the selection of HE parameters because it influences the security level, the compactness of the scheme, and the performance. The HE standard, which is in progress, provides tables of recommended parameters to guide the data scientist in the choice [3]. Furthermore, in literature, there are some recent contributions to guide the developers in setting the so-called HE context: [8,25] make a significant step forward in developing a parameter generator that is readily usable in the PALISADE library; the drawback is that it is library dependent and works only for the BGV scheme, which is indeed a state-of-the-art FHE scheme but is not flexible enough to be employed in all business applications. Furthermore, as investigated in [17], many primitives that are part of the business application should be rewritten to be fully HE-ready: this applies to activation functions for machine learning tasks, as well as to simple statistics primitives, such as mean, variance, or linear regression. This is highly tied to integrating or replacing novel privacy-preserving applications with HE in an enterprise or legacy context, which is challenging to achieve.

In Fig. 3, it is depicted the typical protocol for a HE-based application [9, 15,24]: Besides the HE setup, which considers several exchanges between client and server, we would like to highlight that the additional data encoding and encryption is a significant phase of the HE pipeline, as seen in Sect. 2.

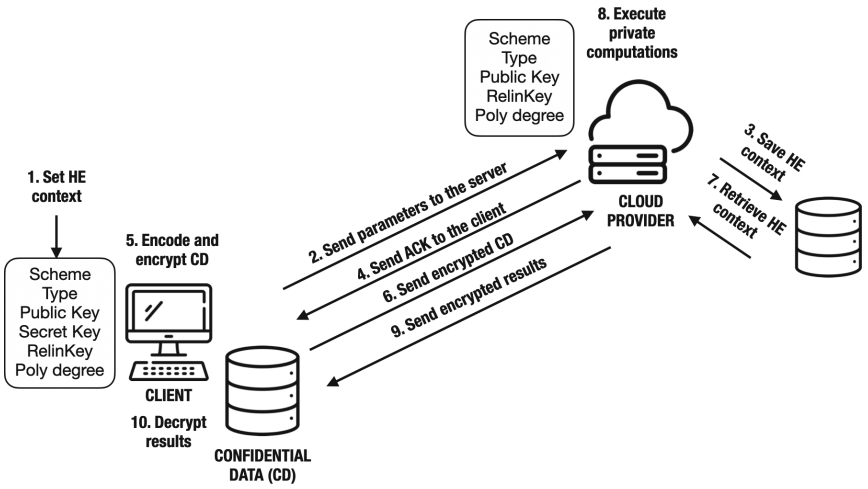

Fig. 3. HE pipeline for a privacy-preserving computation.

Our answer to some of these concerns is the development of the *Homomorphic Encryption Libraries Toolkit* (HELT); this paper represents a work-in-progress report of our experience so far. The purpose of this toolkit will be to compare and benchmark a set of selected HE libraries, which are the ones described in

Sect. 4. HELT is designed as a Docker-based toolkit: each HE library has its directory containing the package with preliminary benchmarks and a Docker file. The user will create a container for each library, solving dependency problems efficiently. Furthermore, a configuration file is provided with suggested HE context parameters that the expert user can modify. In our first version, we plan to compare the libraries by the available HE schemes and their functionalities. The starting point is the computation of simple metrics like the execution time of every single operation, the creation time for the HE context, and the creation time for the keys for every library. At the moment, we ran different tests on the same HE scheme within the same library, changing the parameters. We also measured the dimension in bytes of the generated keys and ciphertext created. The second step is to implement a set of HE-ready building blocks tailored to our business needs and, simultaneously, modular enough to be reused in various enterprise contexts. HELT will also enable us to understand the transition readiness of some critical applications that use confidential data from the standard enterprise environment in the HE domain. Furthermore, we would like to evaluate the selected libraries against the business requirements given in Sect. 2. One of the first conclusions drawn from our analysis is that: (i) HE schemes like BFV and BGV are employable for applications that make use of input that are in the form of strings or integers; (ii) the CKKS scheme is desirable when dealing with machine learning tasks; (iii) the TFHE scheme is robust when there is the need to compare ciphertext, and it is possible to translate the comparison in binary circuits.

Our next steps encompass the following:

- the integration of the OpenFHE library;
- the test of the multikey HE functionality where available, e.g., in Lattigo;
- the investigation of the data integrity problem: we can resort, e.g., to a solution that uses an attached checksum to the ciphertext to detect attacks.

6 Related Work

A few papers about benchmarking HE libraries are available; none encompass comparison frameworks for the financial and statistical applications in scenarios like the ones depicted in Sect. 2. One of the first works about benchmarking is the HEtest framework [30], which tests the main bottlenecks of HE libraries. However, this tool includes only HElib, although the authors remark that it is extensible to other libraries. Melchor et al. [2] compare a modified version of HElib, SEAL, and FV-NFLib to work with large plaintext moduli. The paper is insightful in providing some remarks about the library choice and implementation recommendations. Marrone et al. [23] propose a testbed oriented to evaluate the performance of HE-based applications towards the specific adopted library. Takeshita et al. take a similar approach in HEProfiler [29], focusing on the CKKS scheme. In [14], Gouert et al. answer the problem of the lack of comparison tools for developers proposing Terminator 2 Benchmarking Suite, a compiler that converts benchmarks written in T2, a domain-specific language, into encrypted

programs running on HElib, SEAL, LATTIGO, TFHE, and PALISADE. To our knowledge, none of the reviewed works includes Concrete and OpenFHE libraries inside their benchmarking tools.

7 Conclusion

Homomorphic encryption can preserve data privacy while performing complex computations on it. Nonetheless, it has several challenges in its employment in real-world applications, particularly in the financial context. We described some use cases of interest. The industrial community needs a comparative open-source tool of the most useful HE libraries to gradually let the data analyst be independent of a cryptographer while developing an enterprise application. We propose a two-step approach. Firstly, we aim to provide a HE playground for the data analyst, and the cryptographer is still present to guide the analyst in selecting HE parameters. Then, an environment with built-in homomorphic functions and APIs should be available for the data analyst, apart from a set of HE parameters. We give the reader background for HE theory and describe the most promising HE libraries and their main characteristics. Finally, we describe the requirements and challenges for an HE library in an industrial context, introducing our seminal idea for HELT.

References

1. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: theory and implementation. ACM Comput. Surv. (Csur) **51**(4), 1–35 (2018)
2. Aguilar Melchor, C., Kilijian, M.-O., Lefebvre, C., Ricosset, T.: A comparison of the homomorphic encryption libraries HElib, SEAL and FV-NFLlib. In: Lanet, J.-L., Toma, C. (eds.) SECITC 2018. LNCS, vol. 11359, pp. 425–442. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12942-2_32
3. Albrecht, M., et al.: Homomorphic encryption security standard. Tech. rep., HomomorphicEncryption.org, Toronto, Canada (November 2018)
4. Applied Research Team: Blind learning environment. Tech. rep., Bank of Italy, Rome, Italy (June 2022). https://www.bankit.art/assets/downloads/BLE_Unrestricted.pdf (Accessed 21 June 2023)
5. Armknecht, F., et al.: A guide to fully homomorphic encryption. Cryptology ePrint Archive (2015)
6. Badawi, A.A., et al.: Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022)
7. Bellomarini, L., Blasi, L., Laurendi, R., Sallinger, E.: Financial data exchange with statistical confidentiality: a reasoning-based approach. In: Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, 23–26 March 2021, pp. 558–569 (2021)
8. Biasioli, B., Marcolla, C., Calderini, M., Mono, J.: Improving and automating bfv parameters selection: An average-case approach. Cryptology ePrint Archive, Paper 2023/600 (2023). https://eprint.iacr.org/2023/600

9. Bos, J.W., Lauter, K., Naehrig, M.: Private predictive analysis on encrypted medical data. J. Biomed. Inform. **50**, 234–243 (2014)
10. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1), 34–91 (2020)
11. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption library (August 2016). https://tfhe.github.io/tfhe/
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Paper 2016/870 (2016), https://eprint.iacr.org/2016/870
13. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first annual ACM Symposium on Theory of Computing, pp. 169–178 (2009)
14. Gouert, C., Mouris, D., Tsoutsos, N.G.: Sok: New insights into fully homomorphic encryption libraries via standardized benchmarks. Cryptology ePrint Archive, Paper 2022/425 (2022)
15. Han, K., Hong, S., Cheon, J.H., Park, D.: Logistic regression on homomorphic encrypted data at scale. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 9466–9471 (2019)
16. IBM: Ibm security homomorphic encryption services (2023). https://www.ibm.com/security/services/homomorphic-encryption, (Accessed 30 June 2023)
17. Iezzi, M.: Practical privacy-preserving data science with homomorphic encryption: an overview. In: 2020 IEEE International Conference on Big Data (Big Data), pp. 3979–3988. IEEE (2020)
18. Iezzi, M.: The evolving path of "the right to be left alone" - when privacy meets technology. In: 2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), pp. 225–234 (2021)
19. Library, PHES: Library (2023). https://palisade-crypto.org/software-library/
20. Lloyd, J.: Homomorphic encryption: the future of secure data sharing in finance? (2022). https://www.turing.ac.uk/blog/homomorphic-encryption-future-secure-data-sharing-finance (Accessed 30 June 2023)
21. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing, pp. 1219–1234 (2012)
22. Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F.H., Aaraj, N.: Survey on fully homomorphic encryption, theory and applications (2022)
23. Marrone, S., Tortora, A., Bellini, E., Maione, A., Raimondo, M.: Development of a testbed for fully homomorphic encryption solutions. In: 2021 IEEE International Conference on Cyber Security and Resilience (CSR), pp. 206–211 (2021)
24. Masters, O., et al.: Towards a homomorphic machine learning big data pipeline for the financial services sector. Cryptology ePrint Archive (2019)
25. Mono, J., Marcolla, C., Land, G., Güneysu, T., Aaraj, N.: Finding and evaluating parameters for bgv. Cryptology ePrint Archive (2022)
26. Mouchet, C.V., Bossuat, J.P., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Lattigo: A multiparty homomorphic encryption library in go. In: Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography, pp. 64–70. No. CONF (2020)
27. Sankar, L., Rajagopalan, S.R., Poor, H.V.: An information-theoretic approach to privacy. In: 2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 1220–1227. IEEE (2010)

28. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL (Jan 2023), microsoft Research, Redmond, WA
29. Takeshita, J., Koirala, N., McKechney, C., Jung, T.: Heprofiler: an in-depth profiler of approximate homomorphic encryption libraries (2022)
30. Varia, M., Yakoubov, S., Yang, Y.: Hetest: A homomorphic encryption testing framework. In: Financial Cryptography Workshops (2015)
31. Viand, A., Jattke, P., Hithnawi, A.: Sok: fully homomorphic encryption compilers. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 1092–1108. IEEE (2021)
32. Wood, A., Najarian, K., Kahrobaei, D.: Homomorphic encryption for machine learning in medicine and bioinformatics. ACM Comput. Surv. (CSUR) **53**(4), 1–35 (2020)
33. Wood, A., Shpilrain, V., Najarian, K., Kahrobaei, D.: Private naive bayes classification of personal biomedical data: Application in cancer data analysis. Comput. Biol. Med. **105**, 144–150 (2019)
34. Zama (2022). https://www.zama.ai/post/introducing-the-concrete-framework (Accessed 30 June 2023)
35. Zama: Library (2023). https://github.com/zama-ai