Caring is Sharing – Exploiting the Value in Data for Health and Innovation
M. Hägglund et al. (Eds.)

doi:10.3233/SHTI230116

HEIDA: Software Examples for Rapid Introduction of Homomorphic Encryption for Privacy Preservation of Health Data

Rickard BRÄNNVALL[1], Henrik FORSGREN and Helena LINGE
RISE Research Institutes of Sweden

Abstract. Adequate privacy protection is crucial for implementing modern AI algorithms in medicine. With Fully Homomorphic Encryption (FHE), a party without access to the secret key can perform calculations and advanced analytics on encrypted data without taking part of either the input data or the results. FHE can therefore work as an enabler for situations where computations are carried out by parties that are denied plain text access to sensitive data. It is a scenario often found with digital services that process personal health-related data or medical data originating from a healthcare provider, for example, when the service is delivered by a third-party service provider located in the cloud. There are practical challenges to be aware of when working with FHE. The current work aims to improve accessibility and reduce barriers to entry by providing code examples and recommendations to aid developers working with health data in developing FHE-based applications. HEIDA is available on the GitHub repository: https://github.com/rickardbrannvall/HEIDA.

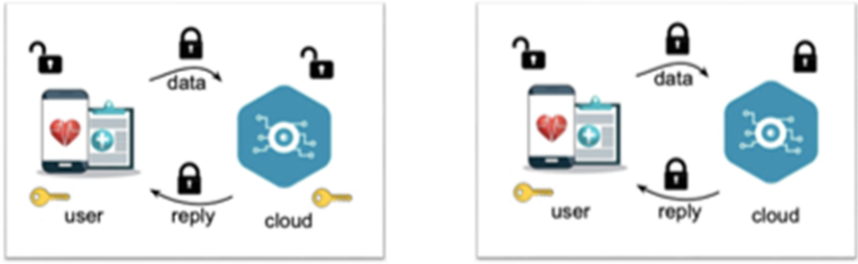
Keywords. Artificial Intelligence, GDPR, Sensitive Data, Privacy Preservation

1. Introduction

The application of AI and machine learning has made remarkable advances in healthcare in the last decade and holds promise to revolutionize the field by improving diagnosis, treatment, and prevention of diseases, e.g., in applications of personalized medicine [1] or medical image analysis [2]. The use of AI in healthcare applications raises concerns as it involves processing sensitive personal information. Privacy is a fundamental human right protected by privacy laws and regulations, e.g., GDPR in the EU and HIPAA in the USA. Adequate privacy protection builds trust and facilitates the sharing of personal health information. This can enable the development of more comprehensive and integrated healthcare services that improve patient care and outcomes, as well as develop commercial opportunities. Several alternative approaches have been proposed, e.g., cryptographic techniques, differential privacy, and federated learning [3]. We aim to show the feasibility of homomorphic encryption for this end.

The remainder of this section briefly reviews Fully Homomorphic Encryption (FHE) and challenges it poses to developers. Following sections discuss methods, present results for the library, and then conclude with lessons learned.

[1] Corresponding Author: Rickard Brännvall, E-mail: rickard.brannvall@ri.se



(a) conventional. (b) homomorphic.

Figure 1. Comparison of alternative implementations of a digital health service.

1.1. Background

Homomorphic encryption was first proposed as a mathematical challenge in the late 1970s by Rivest, but it remained unsolved until 2009, when Gentry presented the first feasible scheme. Progress had been made in the interim years with schemes that offered partial homomorphic encryption, e.g., for either multiplication or addition, but not both. Gentry's construct is the first Fully Homomorphic Encryption (FHE) scheme that supports arbitrary depth computational circuits through the so-called bootstrap. After this breakthrough, a decade of intense research followed, and recently practical solutions have emerged[2]. This work uses the TFHE scheme by Chillotti et al. [5] as implemented in the Concrete [6] library for the Rust programming language.

Data sharing under FHE. We use advanced encryption technology daily: web browsers, car-key fobs, and bank transfers all rely on conventional encryption that protects sensitive data while it is stored or transferred. However, conventional encryption technologies require data to be decrypted before it is processed, as illustrated in Figure 1a for a digital health service hosted in the cloud. Note in the (left) figure how the cloud must have access to the secret decryption key. Figure 1b (right panel) depicts the same service delivered under homomorphic encryption where the cloud does not need the secret key as it can directly process the encrypted data. The cloud returns the results in an encrypted reply without ever having had plain text access to the original data or any results. Besides the benefits of oblivious computation, FHE also prevents unintended secondary use of data – even if sold further, encrypted data cannot be meaningfully used for other purposes. Future privacy remain as long as the secret key is kept safe.

Some notations. The method is named after the mathematical notion of homomorphism, which is used to signify that elements of one set are transformed into elements of a second set while maintaining the relationships between the individual elements in each of the sets; for example, for the arithmetic operations: $E(x+y) = E(x) + E(y)$ and $E(xy) = E(x)E(y)$, whereby $E(x)$ we mean applying the encryption operation to plaintext variable x. It is a common convention to use the word *plain text* for unencrypted data and *cipher text* for encrypted data. Decryption, which maps a cipher text to its corresponding plain text element, is possible only for the secret key holder. Modern FHE schemes are considered unbreakable under very strong cryptographic guarantees and held to be resilient even against hypothetical quantum computer-based attacks [7].

Programming for FHE is complex and may require both cryptography and numerical computation expertise. To ameliorate this, FHE libraries like the Simple Encrypted

[2] Many references in this section are omitted for brevity. Please consult [4] for a comprehensive review.

Arithmetic Library (SEAL) [8] or the Fast Fully Homomorphic Encryption over the Torus (TFHE) [6] implement the underlying cryptographic operations and expose a higher-level API. Viand et al. [9] recently surveyed different libraries and compilers to compare FHE tools and sum up state-of-the-art. Examples of FHE applications, including medical, are provided by Optalysys [10], OpenFHE [11], or Zama [12].

Programming paradigm. The computation model used for FHE provides a challenge for algorithm implementation. Under the standard programming paradigm, one can use flow control operations to arbitrarily break loops when a specific condition is observed (while-loops) or conditionally select which code branch to execute (if-then-else statements). Program flow for FHE computations cannot, by necessity, depend on the data without violating privacy guarantees. Otherwise, an attacker with a clock can get information about encrypted variables by following the program flow of branching code through a so-called timing attack.

Noise management. Numerical precision is generally of little concern when programming in a modern high-level language like Python; with FHE programming, it instead becomes the central concern. By construction, the homomorphic encryption introduces noise into the data. This noise grows for every operation, potentially rendering the results of large computations meaningless even after decryption. Some strategies to manage this problem are bootstrapping, level constraints, and client-side refresh. *Bootstrapping* is possible when the scheme has sufficient capacity to execute its own decryption and encryption circuits under encryption. This refreshes the cipher text by removing some but not all cryptographic noise. In a *leveled approach*, one carefully has to design the algorithm and limit the number of operations such that the results are obtained before bootstrapping becomes necessary. Practically, only simple algorithms without iteration are feasible. The depth of the circuit depends on the parameter choice for the encryption scheme, which we will refer to as key size. *Client-side refresh* combines a leveled approach with roundtrips back to the client. This replaces the costly server-side bootstrap operation with a decryption re-encryption on the client side, which similarly cleans the data of cryptographic noise. It comes at the cost of additional communication overhead and potentially reveals intermediate results of the computation to the client. It is suitable when there naturally are iterations of data transfer between parties, such as remote monitoring and control applications or federated learning.

Computational overhead. The privacy benefits FHE offers come at a price, as it is generally computationally very demanding. It can show orders of magnitude slower execution compared to conventional computation, even in leveled approaches that do not use bootstrapping (which is slow because it executes a complex circuit). It also places greater demands on storage and memory as data files and keys generally have large sizes. Some FHE schemes only support arithmetic operations (addition, multiplication, subtraction) and are hence limited to polynomial function evaluation, but the TFHE scheme selected for this work [6] can approximate general functions through the functional bootstrap [5].

2. Method

HEIDA is an open-source library with examples of health data analysis under homomorphic encryption. It is built in the Concrete library for FHE on the Torus by Zama [6] and designed to be easily imported and used for model performance assessment. HEIDA includes standardized helper functions for key generation, data encryption, and

some higher-level mathematical operations. It is hosted, supported, and version-controlled in a repository on GitHub[3] under the open-source BSD 3-Clear License.

We tested executing with and without encryption to validate the results and compare computational performance. Different key sizes and other parameters were tested and summarized in tables that can assist with the planning and design of data structures and code for a specific problem. To facilitate the use of the library code, we provide worked-through examples of how to implement simple analysis of health data under FHE.

The code examples were chosen to illustrate common machine learning techniques and to show how to overcome technical challenges, such as conditional statements of branching trees or iteration over time series. Only open data sets that did not include personal information were used to avoid privacy concerns with the open-source release.

Example 1: Logistic regression for coronary disease. A logistic regression model trained to estimate a patient's risk of future coronary heart disease based on data about demographic, behavioral, and medical factors. It illustrates a simple machine-learning prediction method implemented through a leveled approach (without bootstrapping).

Example 2: Cardiac disease risk assessment. This is for a similar application as example 1 but built on different input data and a different method of inference. It is included to illustrate how a decision-tree structure can be handled within the constrained programming paradigm that doesn't permit conditional execution of branching code. It also gives an example of parallel execution to make FHE applications run faster.

Example 3: Diabetes self-care analysis. Continuous blood glucose measurements are analyzed under FHE to provide risk scores and gamified advice. This example is based on a solution [13] that was selected as the winner of the 2021-22 Vinter innovation competition arranged by the Swedish innovation agency Vinnova. It showcases time-series analysis with scalable performance and how to implement general non-polynomial functions by applying the functional bootstrapping method [5].

Example 4: Activity monitoring by a neural network. We assume that encrypted personal data, such as blood glucose measurements, carbohydrate intake, physical activity, and insulin administration, to be analyzed by a neural network.

Example 5: Federated learning with encrypted aggregation. A ResNet-18 deep learning model is trained by transfer learning for the classification of dermatoscopic images. This is an example of how to realize gradient-based training of a deep-learning model by a distributed algorithm taking a leveled approach in each aggregation sub-steps which is iterated over many training epochs. It is possible because federated learning inherently builds on data communication round-trips between multiple parties.

3. Discussion

FHE allows computation without exposing data, but it can be computationally very expensive. It is most suitable when one party (a) owns sensitive data that must be processed by a party (b) that is not privy to plain text access to this data. For example, when (b) possesses a superior proprietary algorithm that must be used but cannot be shared with (a). This is illustrated in the case of Electronic Health Records (EHR) in examples 1 and 2; and for what could be considered Personal Health Records (PHR) in examples 3 and 4. Instead of using a proprietary algorithm, it can also be that the tasks consist of combining a second confidential data set that (b) owns but cannot share with

[3] https://github.com/rickardbrannvall/HEIDA

(a). This is the case in example 5, which also illustrates the intended secondary use of data. Note that modern FHE schemes protect data against unintended future use due to their quantum computer resilience, which is especially important for EHR and PHR.

As an alternative, secure multiparty computation can be used to jointly compute any function, but it can have a significant communication overhead and furthermore requires complex coordination between parties (at least compared to the simplicity of FHE cloud deployment in Figure 1b). Differential privacy protects data by adding noise and can be a good complement to federated learning, but it can adversely affect model accuracy.

Lessons Learned. We consider FHE to be most appropriate for high-value applications of moderate complexity at the level of the examples given in Section 2. Large-scale deep learning models, such as transformer language models, are infeasible for implementation under FHE, at least at the time of the writing of this article. The same would count for high-velocity data applications like online video analysis.

On-going work utilizing the library includes collaboration with two regional healthcare providers on personal data protection, particularly with an aim to explore synergies with other advanced privacy-preserving technologies such as federated learning. Currently, we are also investigating new neural network architectures that execute more efficiently under encryption, particularly with an eye to Natural Language Processing.

Conclusions. Homomorphic encryption is a technology that enables cloud services with strong privacy. This work presents open-source[3] software examples intended to facilitate the use of homomorphic encryption for privacy protection in the medical data sector. Future directions include providing Python bindings as well as building software examples for recurrent neural networks.

References

[1] Schork NJ. Artificial intelligence and personalized medicine. In: Precision medicine in Cancer therapy. Springer; 2019. p. 265-83.
[2] Litjens G, et al. A survey on deep learning in medical image analysis. Med Image Analysis. 2017 dec.
[3] Torkzadehmahani R, et al. Privacy-preserving AI techniques in biomedicine. Methods of Information in Medicine. 2022.
[4] Halevi S. Homomorphic Encryption. In: Tutorials on the Foundations of Cryptography. Springer International Publishing; 2017. p. 219-76.
[5] Chillotti I, et al. TFHE: Fast Fully Homomorphic Encryption Over the Torus. Journal of Cryptology. 2019 apr;33(1):34-91.
[6] Chillotti I, et al. CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TFHE. WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography; 2020.
[7] Augot D, et al. Initial recommendations of long-term secure post-quantum systems; 2015.
[8] Microsoft SEAL (release 3.6); 2020. Microsoft Research, Redmond, WA. Accessed 2023-03-05. https://github.com/Microsoft/SEAL
[9] Viand A, Jattke P, Hithnawi A. SoK: Fully Homomorphic Encryption Compilers. In: 2021 IEEE Symposium on Security and Privacy (SP). IEEE; 2021.
[10] Optalysys;. Accessed 2023-03-05. https://github.com/jw-optalysys
[11] OpenFHE;. Accessed 2023-03-05. https://www.openfhe.org/publications
[12] Zama;. Accessed 2023-03-05. https://github.com/zama-ai
[13] Brännvall R, et al. Homomorphic encryption enables private data sharing for digital health: winning entry to the Vinnova innovation competition Vinter 2021-22. In: Proc. Swedish AI Symposium; 2022.