Instructors: Yael Kalai, Ron Rivest May 11, 2022

Homomorphic Encryption in Healthcare

Authors: *Taylor Facen, Ruoming Fang, Keithen Shepard, Julian Viera*

1 Introduction

Data security is often thought of as dealing with the secure transmission of information from point A to point B. We have discussed in class how encryption can be used to prevent adversaries from intercepting, understanding, and misusing the information we are trying to send. In these cases, we are comfortable with the recipient of the message being able to decrypt and use the original data or message we are trying to send. However, there are many cases where instead we would like to keep the underlying information private while still allowing the recipient to use the data for their own purposes. Take for example private healthcare data. Patient healthcare data is often highly confidential information that should not be disclosed to any third-party lest it be used in malicious ways. To protect against this, healthcare providers will often use an encryption scheme known as ***homomorphic encryption***. This specific form of encryption allows certain computations to be performed on the encrypted data to yield new data such that any computation performed on the original decrypted data/message would have resulted in the same result as the decrypted result of the encrypted computation. More specifically, given an encryption function E, a decryption function D, some arbitrary (not necessarily arbitrary but we will talk about this later) calculation C, a private key k, and some message m:

$$C(m) = D(C(E(m, k)))$$

The ability to perform calculations on encrypted messages helps both privatize the underlying information while still giving access and computation power to parties that would like to work with these types of confidential information. Another example use case of homomorphic encryption are businesses in the realm of user advertising. Homomorphic encryption gives these companies the ability to capitalize on user information without needing to actually know the underlying information they are working with.

The notion of a homomorphic encryption scheme has been proposed in the 1970s [19], but has only been constructed in partial forms until Gentry's seminal paper [9] that made fully, unlimited homomorphic computations possible. In this project, we explore the possibilities of applying Fully Homomorphic Encryption (FHE) to the healthcare context from both technical and practical analyses, giving comments on the potential of popularizing this amazing technology to advance both public health and public privacy.

2 Background

In this section, we provide a survey of the mainstream FHE schemes from a high-level standpoint, without delving into too much technical details. We then set out to discuss the intrinsic limitations of these schemes and extend to the key challenges in computation over homomorphically encrypted data in practice.

2.1 A brief overview of Fully Homomorphic Encryption schemes

The most commonly adopted FHE schemes are all based on the hardness of Learning With Errors (LWE) problems for their security [10]. They could be categorized based on the type of encryption applied to the plaintext:

- Bit-wise encryption, in which each bit of the plaintext is separately encrypted as a ciphertext. A representative of this type is the Gentry-Sahai-Waters (GSW) scheme [10], which encrypts every plaintext bit using a matrix.
- Word-wise encryption, in which multiple values are encrypted as a ciphertext. The most notable schemes in this class are Brakerski-Gentry-Vaikuntanathan (BGV) and Brakerski/Fan-Vercauteren (BFV) schemes [3][6], which construct their plaintext and ciphertext spaces using two distinct polynomial rings. The two schemes are fundamentally very similar, except that BFV hides the plaintext message in the most significant digits and the BGV hides the message in the least significant digits of the ciphertext (Figure 1). Another difference is that BGV calls explicitly a *modulus switching* procedure that rescales ciphertext modulus and controls noise, whereas BFV does not. Otherwise, they can be viewed as different modes of the same unified scheme and they perform at comparable levels [14].
- The Cheon-Kim-Kim-Song (CKKS) scheme [5] is a variant of word-wise encryption designed to handle approximate value computation. It is basically very similar to the BGV/BFV schemes, with a main difference that the plaintext is combined with a small noise to be the encryption input. The CKKS scheme is suitable for computations involving real and complex numbers, but the computations are only valid up to a certain degree of precision [12].

All of the schemes mentioned follow a paradigm in which a noise is introduced in the encryption to ensure hardness of LWE problem. Nevertheless, each homomorphic computation introduces more noise which, if left untended, will garble the plaintext underneath and cause a loss of information. A **bootstrapping** operation, which clears the noise from the ciphertext without accessing the secret key, is required to ensure unlimited homomorphic opera-

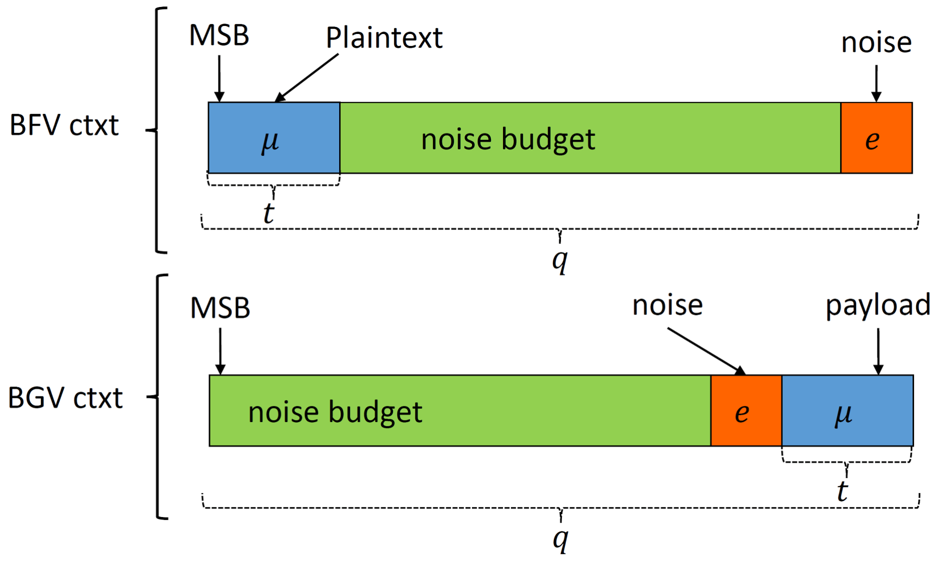

Figure 1: An intuitive diagram showing the encoding strategies of BFV and BGV schemes. Adapted from Inferati [2]

tions. It is typically an expensive procedure that requires tens to hundreds of homomorphic operations, and must be executed every time the noise reaches a threshold.

From a practical perspective, The BGV/BFV and CKKS schemes are most commonly used since they support Single Instruction Multiple Data (SIMD), the packing of multiple data into one ciphertext that renders them advantageous in efficiency. In the following part, we gently unfolds the BGV/BFV scheme and use it as a basis to explore the challenges in current homomorphic cryptography.

2.2 Main challenges to efficient homomorphic encryption

In the BGV/BFV scheme, each plaintext (in a vector form) is represented by a polynomial with N coefficients, modulo some value t which is a prime power commonly referred to as the plaintext modulus [14]. The corresponding ciphertext is a pair of ciphertexts, each with N integer coefficients modulo a ciphertext modulus Q, $Q \gg t$. A small, commonly Gaussian noise term e is introduced in computing ciphertext $ct = (c_a, c_b)$ from a given message m, as required by the underlying Learning With Errors (LWE) assumption that guarantees the encryption scheme's security.

There are three types of operations on the ciphertext:

1. Homomorphic addition. This is done simply by polynomial-wise addition: $ct_0 + ct_1 =$

$(c_{a0} + c_{a1}, c_{b0} + c_{b1})$.

2. Homomorphic multiplication. Multiplying two pairs of polynomials produces three different products: $c_{a0}c_{a1}$, $c_{a0}c_{b1} + c_{a1}c_{b0}$ and $c_{b0}c_{b1}$. A *key-switching* operation is required to convert this triplet to a pair of polynomials encrypted by the original secret key. The key-switching operation is dominates the cost of the whole multiplication process.

3. Homomorphic permutations. This operation permutes the N plaintext values underlying the ciphertext. It is achieved by performing *automorphism* on each ciphertext polynomial, and another key-switching to give the correct output.

Moreover, under the prevailing homomorphic encryption paradigm, each operation on the ciphertext will introduce more noise. Apart from the need for bootstrapping procedures, this limitation has another critical implication: to make sure that an adequate number of operations can be executed before each bootstrapping step, the noise budget need to be sufficiently large, and therefore a large Q value is required, typically at the order of 512- or 1024-bits.

The large bit width of ciphertext polynomial coefficients gives rise to two serious computational challenges. First, ciphertexts are large; for instance, assuming 512-bit width coefficients, representing a plaintext of $N = 16K$ numbers require a ciphertext that takes 2MB space, not to mention the secret key and other intermediates generated during computation steps such as the key-switching. This large inflation factor exerts a huge overhead on the memory of the system used for homomorphic computation. Second, the cost of modular multiplication grows quadratically with the bit widths of the operands, hence having long coefficients imposes a much higher computational burden. Together they drag down the efficiency of homomorphic computation tremendously.

An antidote to these two problems is the Residue Number System (RNS) representation of polynomials: by the good grace of the Chinese Remainder Theorem, we can express a $mod{-}Q$ polynomial equivalently with k polynomials with smaller moduli $q_1, q_2, ..., q_k$, provided that these moduli are coprime and $Q = q_1 q_2 ... q_k$. This technique alleviates the computational cost issue, but still invokes a big space demand.

Another key algorithm involved is the **Number Theoretic Transform (NTT)**, which is the modular-arithmetic variant of the discrete Fourier Transform algorithm. NTT speeds up polynomial multiplication from $O(N^2)$ to $O(Nlog(N))$ modular operations and is critical in homomorphic computation. Furthermore, polynomial addition and automorphism can also be performed in the NTT domain, hence NTT is used as the first step in nearly all important homomorphic operations, including the key-switching step. The cost for this ubiquitous operation, however, is high and has been an important problem in the literature [11] [16] [18]. Typical algorithms, based on butterly operations, introduces complex dataflows and pose a challenge to the memory system [7]. This is particularly tricky in the homomorphic encryption context, in which operands and intermediates are all large in size.

To summarize, the nature of cryptography dictates the computational barrier in applications of fully homomorphic encryption. The requirement for bootstrapping and sufficient noise budget leads to unconventionally large bit width for ciphertexts, which causes immense challenges for both computation speed and memory management. NTT as an ubiquitous prerequisite to homomorphic operations introduces additional complications in data management. As such, given the fundamental structure of homomorphic encryption, the key point for achieving a practical performance lies in handling data and computation schedules, which may not be fulfiled to a satisfactory degree with a standard computing hardware like a CPU or a GPU.

2.3 The state of the art in FHE acceleration hardware

Owing to the unique challenges in homomorphic encryption, a specialized computer architecture is needed to accelerate it to near-plaintext computation speed. Towards that end, various FHE accelerators have been proposed. In this study we cover two of them - F1 and its successor, CraterLake.

F1 [7] is a hardware accelerator designed for both performance and versatility in FHE programs. It has three distinguishing features:

1. F1 optimizes the efficiency of basic primitive operations, such as modular arithmetic, NTT and permutations. Acknowledging the fact that most steps in homomorphic computation are actually compositions of these basic operations, F1 designed specialized functional units and memory architecture to specifically target these primitives. This design also renders F1 capable of accelerating a wide variety of algorithms.
2. F1 recognizes the importance of data and memory management in its design, and therefore implements large, high-speed memory modules while minimizing data movement and improving data re-use through custom scheduling software that explicitly manages the computation and use memory bandwidth economically.
3. F1 comprises a novel compiler that bridge memory optimization and programmability of the chip. A domain-specific language (DSL) is implemented for writing F1 programs, while the F1 compiler translates the program into optimized flow of homomorphic encryptions.

Geared with these optimizations, F1 is the first system to accelerate complete FHE programs and out-performs latest software implementations by gmean 5400$\times$ and up to 17000$\times$.

CraterLake [17] builds on F1 to make FHE more accessible for very large ciphertexts and deep computations including deep neural networks like ResNet and LSTMs. Employing an extremely wide, de-clustered architecture that reduces data footprint, new types of functional units that take advantage of boosted key-switching algorithms (for BGV/BFV) and reduce auxiliary data generated, and a vector chaining technique that enable many concurrent operations in the functional unit pipeline, CraterLake is able to attain a 11.2× speedup over F1, and 4600× speedup over a CPU, in deep computation benchmarks.

3 Related Work on healthcare applications

There exist a few studies exploring the use of Homomorphic encryption in healthcare, but the attempts are as nascent as the technology itself. Kocabas et al. [15] used both Gentry's scheme and a BGV-scheme setup to calculate the average heart rate of a patient using ECG data on cloud storage. While simple operations like averaging could be computed in real time with an approximately 70 ms latency, they found a huge storage expansion of 217x which necessitates a customized cloud storage mechanism. Sun et al. [21] proposed a four-layer mobile healthcare framework and experimented on three types of computation: average heart rate, long QT syndrome detection and chi-square tests. The reported latency for these calculations ranged from several hundred to over one thousand milliseconds. Jiang et al. [13] implemented an IoT-and-cloud system that performs diabetic retinopathy on homomorphic encrypted data, based on a customized scheme.

Apart from standalone systems, homomorphic encryption is also integrated in other collaborative analytics system design, most prominently federated learning. Chen et al. [4] devised FedHealth, a federated transfer learning framework for wearable healthcare, in which additive homomorphic encryption was employed to aggregate data. They demonstrated FedHealth in the context of human activity recognition and Parkinson's disease auxiliary diagnosis. In FAMHE, a federated analytics system proposed by Froelicher et al. [8], local parties compute partial results and use homomorphic encryption to secure these intermediaries for further aggregation. FAMHE was utilized to perform vital biomedical analysis including Kaplan-Meier survival analysis in oncology, and genome-wide association studies in medical genetics. Both studies reported high accuracies comparable to those in conventional frameworks, showing that federated learning with homomorphic encryption can achieve both privacy and accuracy at the same time.

In summary, it isn't hard to see the various inadequacies of homomorphic encryption that we discovered. It still runs very slow, produces large keys and ciphertexts, and can perform only limited types of operation such that all applications are customized to fit one or two analytics tasks. Purely homomorphic encryption-based systems are largely limited in processing capacities, and less experimented with true data; whereas homomorphic encryption with federated learning might prove to be a more practical solution currently. In all, there are huge spaces for improvement if the best technologies can be used, but significant barriers, cost-wise and expertise-wise, are in place.

4 Proposed security policies for FHE in healthcare

Having reviewed the technological aspects of FHE and major needs in the healthcare sectore, we propose some simple security policies for managing homomorphic encryption in the healthcare sector:

- Only the owner of the data should be able to decrypt it/have access to the secret key. This relates to confidentiality as the party that is manipulating the data won't be able to attain the underlying private information.
- Users of the data should be able to perform meaningful computations on the data while it is encrypted. This relates to availability as the users know they can compute relationships between the encrypted data that hold the same functionality as relationships between the decrypted form of the same data.
- Owners of the data should be able to decrypt the result of a computation on an encrypted message and receive the same result as simply performing the same computation on the original decrypted message. This relates to availability as it ensures the definition of homomorphic encryption holds and users can properly interact with the data.
- Users of the data should not be able to perform computations on the encrypted data and alter the underlying encrypted information. This relates to integrity as the original decrypted data should remain unmodified and untouched to preserve the confidential information.

5 Analysis of FHE Schemes on Healthcare Data

To get a better understanding of the applications and associated limitations of the use of FHE in healthcare, we ran a test to study the efficiency of two different FHE schemes for potential healthcare data. UK Biobank is a large long-term biobank study in the United Kingdom that has collected various medical data on thousands of patients. While getting access to the actual data requires an account, and much of the data is in a format difficult to clean, we have used the User Guide [1] for a list of the type of medical data the Biobank collects.

For our analysis, we considered visual acuity (i.e. 20/20 vision), intraocular pressure, BMI (Body Mass Index), total cholesterol level, and sex (male or female). While we did not gain access to actual data points from the UK Biobank, we researched the distributions for each of these data points, and created random values for n potential patients.

With a set of dummy data that could resemble a real analysis needed by a hospital or research facility, we sought to create a linear classifier for these patients based on their values for each feature. To accomplish this, we considered and analyzed multiple options. The first was to simply perform all of this work directly, without using any type of fully homomorphic encryption. This process would mimic a hospital or medical research group performing data calculations on their own. The alternative was to use one of two FHE schemes to perform the calculations, acting as an untrusted server to which a hospital or medical research group could offload the computations. Both BFV and CKKS encryption schemes were tested in this analysis. The two sections below go into further detail on how we tested these schemes, and an overview of our results.

5.1 BFV

For our analysis of BFV, we used the `py-fhe` library on GitHub [20]. Our experiment considered two methods: performing the calculations for a linear classifier without any BFV and sending the encrypted data of each patient to the untrusted server to perform the computations (a weighted sum of each encrypted data point) for the linear classifier. In practice, both methods were run locally, so our time comparisons do not consider the time taken to send the encrypted data to an FHE server and to get the encrypted results back from the server. However, the efficiency of FHE computations is the key area of interest here, so we leave sending and receiving time analysis to future research.

It is also important to note here that although BFV can technically handle real numbers, it is rather inefficient on numbers that are not integers. So for our BFV analysis, we have ensured that all of the data is represented as integers, and each weight is an integer. Thus, the weighted sum of each data point is also an integer.

In Figure 2, we have a plot showing how computation time changes based on the number of patients in the data set and whether the owner of the data performs the computations directly or uses an FHE server to obtain results.

Clearly, using this implementation of BFV to calculate the weighted sums and perform linear classification takes significantly more time than if the data owner performs the calculations

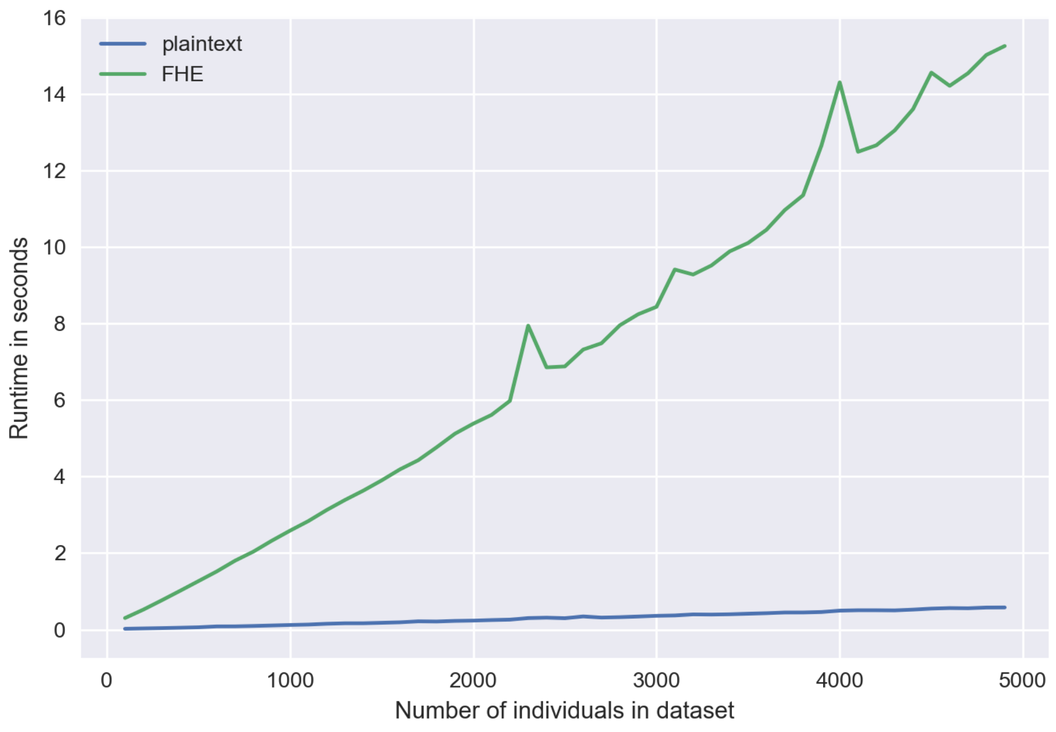

Figure 2: Comparison of runtimes between plaintext computations and BFV

directly. For example, for 5,000 patients, using this BFV library takes over 15 times as long. It is also important to note that BFV here is significantly restrictive, as integers are the only real data we can operate on in practice.

In the next section, we will show how we used CKKS and analyze its results.

5.2 CKKS

For our analysis of CKKS, we used the same `py-fhe` library on GitHub [20] that we used for BFV. Since CKKS works efficiently on real numbers, we can leverage this to perform computations on our data involving real numbers, such as standardizing our values.

We analyzed CKKS under two different methods. The first was using the same exact dataset as we used with BFV, again taking weighted sums and performing a linear classification. Similarly, this dataset is only comprised of integers. However, before sending the data to the FHE server, the owner of the data first standardizes all values to have mean $\mu = 0$

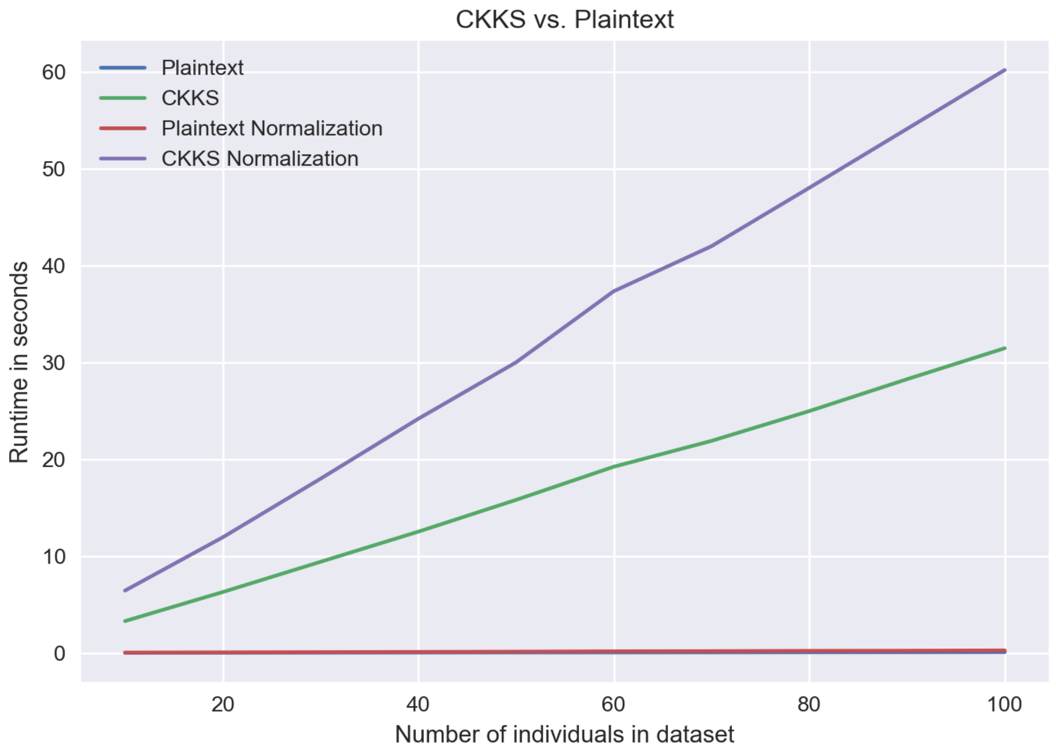

Figure 3: Comparison of runtimes between plaintext computations and CKKS

and standard deviation $\sigma = 1$. In this process, the owner of the data sends the encrypted, standardized data to the FHE server, where we use the CKKS library to perform all of our desired operations.

The second method was to perform the standardization in the CKKS process. Since this CKKS does not directly support division, we precompute the mean μ and standard deviation σ for each data point. The owner of the data then sends each encrypted datapoint, as well as each encrypted μ and each encrypted $1/\sigma$. With the CKKS library, we then compute $(x - \mu) \cdot 1/\sigma$ for each datapoint x, and compute weighted sums and linear classification.

In Figure 3, we plot 4 different scenarios: performing linear classification directly on plaintext without normalization, performing linear classification on the plaintext with normalization, using CKKS with the data standardized beforehand, and using CKKS for linear classification and standardization.

We can see here that performing the computations directly on the plaintext, with or without normalization, is by far the fastest. We can also see that when offloading computations to CKKS, it takes roughly twice as long when the normalization computations are also offloaded to CKKS (30 seconds vs. 60 seconds for 100 patients).

5.3 Discussion of Results

As we can see from the two experiments above, FHE computations take significantly more time and also currently require significant preparation on the owner of the data to make sure the data works for the specific FHE scheme they are trying to use.

It is also important to consider the differences in efficiency between BFV and CKKS. For 5,000 patients, the BFV library we used performed the desired calculations in just under 16 seconds. On the other hand, the CKKS library took over 60 seconds to perform the desired calculations for only 100 patients. It is also important to note that realistically, computations on medical data and any intermediate values often contain floating point numbers, so this runtime slowdown with CKKS is a serious concern.

Another key point here is that relative to the variety of computations we may perform on medical data, linear classification is very simple. More realistic neural networks or other more complicated algorithmic techniques may take substantially longer to perform, both on integers and real numbers. There is clearly a tradeoff here between offloading computations you do not want to or cannot perform on your own, and sacrificing time by doing so.

6 Conclusion and Future Work

This paper has shown a review of some popular FHE schemes, an introduction to FHE hardware accelerators F1 and CraterLake, an overview of security protocols in the healthcare industry, and an example test of how two FHE schemes (BFV and CKKS) can be used on potential healthcare data. We compared the efficiency of schemes on simulated data to understand how data size and pre-processing steps affect FHE computation times. There are still significant breakthroughs needed to make FHE more efficient and many potential avenues for future work.

One limitation for our analysis is that all encryption and calculation work was completed locally. Due to the advanced software, hardware, and knowledge needed to perform homomorphic encryption, it is more likely that a medical facility would use a third party provider to perform the FHE computations via an untrusted server. Therefore, it would be beneficial to include the time necessary to both encrypt and transmit the data to and from the host and third party in the analysis. Future work could include comparing the computational pros and cons between running FHE locally and remotely. This work could also explore any pre-processing steps that can be completed locally that would shorten the transmission time to and from the third party.

Our analysis was based on a relatively simple algorithm, linear classification. Although this algorithm is used significantly in practice, it is not the only computation used in the real world. More work can be done to test FHE schemes on more complicated algorithms such as neural networks and image recognition. Intuitively, one could hypothesize that more advanced algorithms would require longer processing time on standard computing hardware. Any analysis using these types of computations should also introduce tools and methods specializing in optimizing FHE computation speeds. For example, one could simulate performing advanced computations with F1 or CraterLake. This future work could explore the technical and operational feasibility of implementing these acceleration technologies in a real-world setting.

Lastly, with security being a top priority in the healthcare sector, it is important to conduct serious review of how secure FHE could be when utilizing a third party server. One could review the type of information an adversary gains through various hacks and if this information offers any clues to the structure and nature of the underlying data. Within the realm of trust and security, future work also includes exploring how the owner of the data can verify that the results given by the FHE server are valid. Surely, one could perform the computation locally and compare the results. However, this would defeat the purpose of outsourcing computational work to a third party.

7 Acknowledgement

We sincerely thank Professor Ron Rivest and Professor Yael Kalai for their teachings and inspirations throughout this course. We also thank our TAs, Kyle, Andres and Lucas, for their continued support to our project. Lastly, we appreciate the help from Nikola Samardzic, who provided valuable insights for FHE acceleration hardware and their practical prospect during his discussions with the team.

References

[1] Uk biobank showcase user guide: Getting started, Aug 2017.

[2] Introduction to the bgv encryption scheme, February 2022.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[4] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 35(4):83–93, 2020.

[5] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[6] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[7] Axel Feldmann, Nikola Samardzic, Aleksandar Krastev, Srini Devadas, Ron Dreslinski, Karim Eldefrawy, Nicholas Genise, Chris Peikert, and Daniel Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption (extended version). *arXiv preprint arXiv:2109.05371*, 2021.

[8] David Froelicher, Juan R Troncoso-Pastoriza, Jean Louis Raisaro, Michel A Cuendet, Joao Sa Sousa, Hyunghoon Cho, Bonnie Berger, Jacques Fellay, and Jean-Pierre Hubaux. Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption. *Nature communications*, 12(1):1–10, 2021.

[9] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[10] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*, pages 75–92. Springer, 2013.

[11] David Harvey. Faster arithmetic for number-theoretic transforms. *Journal of Symbolic Computation*, 60:113–119, 2014.

[12] Ilia Iliashenko and Vincent Zucca. Faster homomorphic comparison operations for bgv and bfv. *Proceedings on Privacy Enhancing Technologies*, 2021(3):246–264, 2021.

[13] Linzhi Jiang, Liqun Chen, Thanassis Giannetsos, Bo Luo, Kaitai Liang, and Jinguang Han. Toward practical privacy-preserving processing over encrypted data in iot: an assistive healthcare use case. *IEEE Internet of Things Journal*, 6(6):10177–10190, 2019.

[14] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. Revisiting homomorphic encryption schemes for finite fields. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 608–639. Springer, 2021.

[15] Ovunc Kocabas and Tolga Soyata. Towards privacy-preserving medical cloud computing using homomorphic encryption. In *Virtual and Mobile Healthcare: Breakthroughs in Research and Practice*, pages 93–125. IGI Global, 2020.

[16] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *International Conference on Cryptology and Network Security*, pages 124–139. Springer, 2016.

[17] Aleksandar Krastev Nathan Manohar Nicholas Genise Srinivas Devadas Karim Eldefrawy Christopher Peikert Daniel Sanchez Nikola Samardzic, Axel Feldmann. Craterlake: A hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th International Symposium in Computer Architecture (ISCA-49)*, June 2022.

[18] Özgün Özerk, Can Elgezen, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. Efficient number theoretic transform implementation on gpu for homomorphic encryption. *The Journal of Supercomputing*, 78(2):2840–2872, 2022.

[19] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[20] Saroja Erabelli Sarojaerabelli. Py-fhe: A python library for fully homomorphic encryption.

[21] Xiaoqiang Sun, Peng Zhang, Mehdi Sookhak, Jianping Yu, and Weixin Xie. Utilizing fully homomorphic encryption to implement secure medical computation in smart cities. *Personal and Ubiquitous Computing*, 21(5):831–839, 2017.