Journal of Physics: Conference Series

PAPER • **OPEN ACCESS**

A Comparative Study of Homomorphic Encryption Schemes Using Microsoft SEAL

To cite this article: Shereen Mohamed Fawaz *et al* 2021 *J. Phys.: Conf. Ser.* **2128** 012021

A Comparative Study of Homomorphic Encryption Schemes Using Microsoft SEAL

Shereen Mohamed Fawaz [1], Nahla Belal [2], Adel ElRefaey [3], Mohamed Waleed Fakhr [4]

[1] Department of Basic and Applied Sciences, College of Engineering and Technology, Arab Academy for Science, Technology, and Maritime Transport, Smart Village, Giza, Egypt
[2] Department of Computer Science, College of Computing and Information Technology, Arab Academy for Science, Technology, and Maritime Transport, Smart Village, Giza, Egypt
[3] Department of Basic and Applied Sciences, College of Engineering and Technology, Arab Academy for Science, Technology, and Maritime Transport, Smart Village, Giza, Egypt
[4] Department of Computer Engineering, College of Engineering and Technology, Arab Academy for Science, Technology, and Maritime Transport, Cairo, Egypt

shereenfawaz@aast.edu, nahlabelal@aast.edu, [c]adel_elrefaey@aast.edu, waleedf@aast.edu

Abstract – Fully homomorphic encryption (FHE) technology is a method of encrypting data that allows arbitrary calculations to be computed. Machine learning (ML) and many other applications are relevant to FHE such as Cloud Computing, Secure Multi-Party, and Data Aggregation. Only the authenticated user has the authority to decrypt the ciphertext and understand its meaning, as encrypted data can be computed and processed to produce an encrypted output. Homomorphic encryption uses arithmetic circuits that focus on addition and multiplication, allowing the user to add and multiply integers while encrypted. This paper discusses the performance of the Brakerski-Fan-Vercauteren scheme (BFV) and Cheon, Kim, Kim, and Song (CKKS) scheme using one of the most important libraries of FHE "Microsoft SEAL", by applying certain arithmetic operations and observing the time consumed for every function applied in each scheme and the noise budget after every operation. The results obtained show the difference between the two schemes when applying the same operation and the number of sequential operations each can handle.

Keywords: Fully Homomorphic Encryption, Machine Learning, BFV, CKKS, Microsoft SEAL

1.0 Introduction

For organizations in the public and private sectors, the ability to do sophisticated calculations on encrypted data promises a new degree of privacy and data protection [1]. Encrypting a document is like placing certain documents inside a locked safe. A locked safe is a fine example, as cryptography and safes both serve the same purpose. Usually, they preserve the confidentiality of sensitive and important data. When people decide to remove their important documents from the safe, this greatly increases the risk of all the common threats. Encryption is usually handled in the same way. Unfortunately, to encrypt a document is not the same concept as putting it in a safe. Because there is a type of encryption that allows us to get around some of these limits. This is defined as homomorphic encryption [2], and its

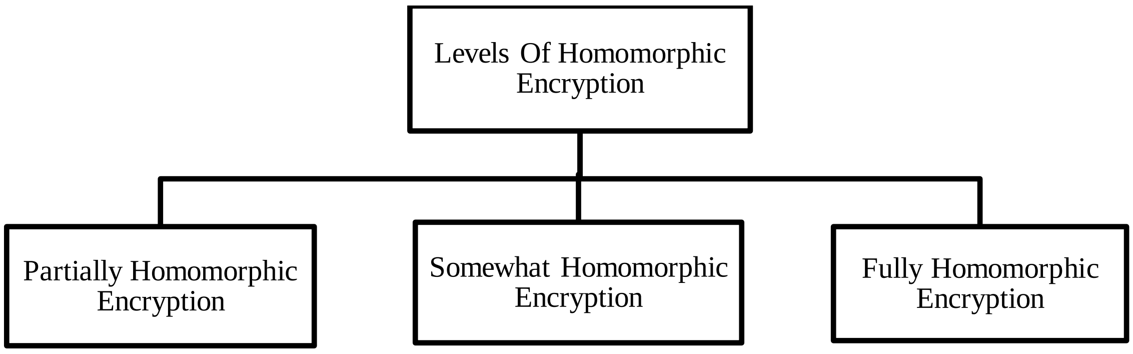
unique feature is the ability to implement meaningful operations on encrypted data without first decrypting them [3]. Homomorphic encryption differs from traditional encryption methods in that it allows computations and operations to be operated directly on encrypted data which is in the ciphertext form without exposing the secret key [1]. The computed results are kept in encrypted form. In 2010 Craig Gentry proposed a new method of encrypting called Fully Homomorphic Encryption (FHE) [4]. The goal behind fully homomorphic encryption is to allow its users to perform certain operations on encrypted data without accessing the encryption key. Particularly, the FHE concept has many applications to improve cloud computing security [5]. This cryptographic holy grail is capable of performing any efficiently computable function an unlimited number of times, making safe multi-party computation more efficient [6]. Unlike other forms of homomorphic encryption, it can perform arbitrary calculations on the ciphertexts. FHE research and development has continued since then, and two more generations of FHE have been developed under Gentry's supervision [4, 7]. The second-generation FHE was released in 2012 with the Brakerski-Fan-Vercauteren (BFV) scheme [8], and the third-generation FHE was released in 2013, with the FHEW scheme being released in 2014 and the TFHE scheme being released in 2016 [1].
Different schemes tend to use completely different techniques to regulate the efficiency and noise growth that is required to with success work with FHE [9]. In addition, each scheme has completely different parameters that ought to be understood to line them during a correct approach [1]. The parameters affect the security of the scheme, the types of plaintexts worked with, the number of homomorphic computations done, and therefore the overall performance of the theme [1]. Lack of information exists explaining the trade-offs between different schemes with the same computational models. To know which is the most suitable scheme, implementation is done with a specific use case for each one to determine which is optimum for the case.
This paper discusses the performance of the BFV and CKKS scheme using one of the most important libraries" Microsoft SEAL" [10] by applying addition, multiplication and squaring arithmetic operations and observe the time consumed for every function applied in each scheme.
The rest of the paper is organized as follows. Section 2 presents the related work and common features of homomorphic encryption including the levels of homomorphic encryption and its developments till it reached the FHE level along with its applications of Homomorphic Encryption and its usage, in addition to homomorphic algorithms, schemes, and libraries. Section 3 shows the experimental work and results. Finally, the last section, Section 4, concludes and gives the future work.

2.0 Related work

In this section, a quick overview of the main classifications of homomorphic encryption is illustrated. Although the schemes of this security technique are still eccentric in the field, nowadays its use has boomed in many research areas and applications such as cloud computing, data analytics, governmental security issues, and most importantly the machine learning world. This paper highlights the perks of its performance through a comparison between BFV and CKKS schemes using arithmetic operations.

2.1 *Levels of Homomorphic encryption*

HE is classified into three levels based on the number and kind of mathematical operations on the encrypted message known as Partially Homomorphic encryption (PHE), Somewhat Homomorphic coding (SHE), and Fully Homomorphic Encryption (FHE) [11].


Levels Of Homomorphic Encryption
Partially Homomorphic Encryption
Somewhat Homomorphic Encryption
Fully Homomorphic Encryption

1. **Partially Homomorphic Encryption (PHE):** In the PHE scheme, just one kind of mathematical process is allowed on the encrypted message, If the operation on ciphertexts yields the encrypted version of the total of the corresponding plaintexts, then the homomorphic scheme is named additive [12], if it yields the encrypted version of the product of the plaintexts, then the homomorphic scheme is named multiplicative but with an unlimited number of times [13].

2. **Somewhat Homomorphic Encryption (SHE):** In SHE, each addition and multiplication operation in ciphertexts is supported, however in practice they permit a restricted number of operations. Most SHE schemes admit a large number of additives and a tiny low variety of products on ciphertexts [14]. This limits the computations which will truly be outsourced to the cloud, so it restricts the quality of SHE schemes for certain applications [8].

3. **Fully Homomorphic Encryption (FHE):** FHE permits an outsized variety of various forms of analysis operations on the encrypted message with an unlimited variety of times. Unfortunately, the computational costs of FHE are large [15]. Performing too many multiplications uses an expensive bootstrap algorithm to process the encrypted data, allowing additional multiplications [6].

2.2 Applications
Homomorphic encryption has a wide range of applications. Let's imagine a corporation wants to show it has the financial resources to complete a project, or it needs to submit information for an audit by a third-party corporation or government organization [6]. It could submit sensitive financial data and establish that it meets requirements or is in compliance using homomorphic encryption without ever knowing the actual data.

- **Data in the Cloud : How to Keep it Safe**

Data may be protected in the cloud while also keeping the option to calculate and search ciphered information that can be afterward decoded without harming the integrity of the data as a whole by using homomorphic encryption [11].

- **Enabling Data Analytics in Regulated Industries**

Homomorphic encryption protects the privacy of users and patients by encrypting data and sending it to commercial cloud environments for research and data sharing. It can be used by businesses and organizations in a range of industries, like financial services, retail, information technology, and healthcare, to let users access data without viewing it in its unencrypted form [15].

- **Increasing the Security of Elections**

Homomorphic Encryption employs addition operations, which is ideally suited for voting-related applications since it allows users to add up diverse numbers in an unbiased manner while maintaining their privacy. Not only could this technology secure data against alteration, but it could also allow authorized third parties to independently verify it [16].

- **Private Machine Learning**

In recent years, there has been a lot more interest how to learn from data (through machine learning) while keeping the data itself private [5]. One technology that can support us in achieving this goal is FHE. Many machine learning models have attempted to create cryptographic systems using homomorphic ciphers for the prediction and classification of private information [15]. Such as processing encrypted patient data and obtain an encrypted diagnosis in health. Moreover, in the field of computational linguistics, performing a secure neural machine translation securely without revealing any of the text needed to be translated using homomorphic encryption schemes which will be focused on in the future work

2.3 Homomorphic encryption algorithms, schemes, and libraries

Here, the procedure of encrypting data is analyzed. Plaintext refers to unencrypted data, whereas ciphertext refers to encrypted data. The three main algorithms used in an encryption scheme are listed below [11]:

- **KeyGen (security parameters):** The security parameters are fed into the Key Generation process. The scheme's keys are then output.
- **Encrypt (plaintext, key, randomness):** A plaintext, a key, and some randomness are all inputs to the Encryption algorithm. It then outputs the ciphertext that corresponds to it.
- **Decrypt (ciphertext, key):** The Decryption algorithm has two inputs: a ciphertext and a key. It returns the equivalent plaintext.

The same key is used to encrypt and decrypt in a private key encryption technique. This means that everyone who can encrypt plaintext can decipher ciphertext as well [11]. Because the link between encryption and decryption is symmetric, certain sorts of encryption systems are referred to as "symmetric" [17]. But, different keys are used to encrypt and decrypt in a public key encryption method. Although a person may be able to encrypt data, he or she may not be able to decrypt the ciphertext. Because the link between encryption and decryption is asymmetric, these sorts of encryption systems are also referred to as "asymmetric" [17]. FHE schemes can be classified into three categories based on model processing [18]. Computations are modeled as Modular arithmetic, Boolean operations, and Floating-point arithmetic [19]. In Table 2 there are several open-source libraries of homomorphic encryption algorithms [9]. Each scheme has a specific use case. There is a lack of information to explain the tradeoffs between the different schemes with the same computational model. The tables below show the operations done for each scheme in order to help the user to choose the suitable scheme depending on the research application criteria.

Table 1: Homomorphic Encryption Libraries and their Supported Schemes

Libraries	Supported Schemes
Microsoft SEAL	Microsoft's widely used open-source library supports the BFV and CKKS schemes [10].
HElib	IBM's first and most frequently used library, which supports the CKKS and BGV schemes as well as bootstrapping [20].
Lattigo	This is a Go-Lang lattice-based cryptographic library [21].
PALISADE	A widely-used open-source library that supports numerous homomorphic encryption algorithms with multiparty support, including BGV, BFV, CKKS, TFHE, and FHEW [22].

3.0. Experimental Work and Results

The arithmetic operations studied in this paper are addition, multiplication, and squaring two vectors with different sizes, in addition to calculating the time taken by each operation. By performing the arithmetic operations homomorphically, noise becomes a problem that is likely to be faced, and it should be taken into consideration. The noise budget is consumed at a rate determined by the encryption parameters for homomorphic computations. During applying one of the operations to the vectors, the noise budget left should be observed because after every operation done the noise increases in the ciphertext, which means that the noise budget remaining decreases. If the noise budget reached 0 it would be impossible to decrypt the result as it will be corrupted due to the huge increase in the noise. After each operation, additional noise is added to the resulting ciphertext. In an addition operation, the resulting noise will be thought of as two added values in Equation (1). The risk of losing the data arises with larger operations such as multiplication and squaring operations as shown in Equation (2). After several sequential times, the whole data can be corrupted due to the exploding noise resulting in decryption failure. The experiment in the paper is carried out by choosing size 2^n values for the vectors and suitable encryption parameters for each scheme. In this work the parameters chosen for the BFV scheme are polynomial modulus degree(2^n value), ciphertext modulus and the plaintext modulus. For the CKKS scheme, the chosen encryption parameters are poly modulus degree and the ciphertext modulus in a different technique than BFV.CKKS does not use the plaintext modulus encryption parameter. The below tables and figures show the detailed addition, multiplication and squaring arithmetic operations carried out and compared between the two schemes.

$$\text{Noise}\,(x' + y') = \text{Noise}(x') + \text{Noise}(y') \qquad (1)$$

$$\text{Noise}\,(x' * y') = \text{Noise}(x') * \text{Noise}(y') \qquad (2)$$

Table 2: BFV Multiplication

Vectors size |Poly modulus degree(2^n)	Encoding (µs)	Encryption (µs)	Multiplication (µs)	Decryption (µs)	Decoding (µs)	Total Time (µs)
4096	135	2834	4720	423	62	8174
8192	259	7968	16339	1493	130	25989
16384	524	26797	61986	6377	248	95932

The BFV multiplication of two vectors is done as shown in Table 2 by passing through 5 operations which are encoding, encryption, multiplication, decryption and decoding. The total time of the multiplication increases with the increase of the vector size and the most two operations consumed time are the encryption and multiplication.

Table 3: BFV Addition

Vectors size |Poly modulus degree(2^n)	Encoding (µs)	Encryption (µs)	Addition (µs)	Decryptio n (µs)	Decoding (µs)	Total Time (µs)
4096	135	2834	62	309	85	3425
8192	259	7968	239	1104	103	9673
16384	524	26797	1081	4294	225	32921

Table 3 shows the operations needed for the BFV Addition. The Addition operation compared to the multiplication in Table 2 shows that the multiplication consumes a larger amount of time than addition. For example, using vector size 4096, the multiplication operation consumed 4720 µs, while the addition consumed 62 µs. Moreover, total time for multiplying the two vectors homomorphic-ally is 8714 µs, while the homomorphic addition of the two vectors consumed 3425 µs. This shows that the average time consuming for multiplication is 42% larger than addition in the BFV scheme.

Table 4: BFV Squaring

Vectors size |Poly modulus degree(2^n)	Encoding (µs)	Encryption (µs)	Squaring (µs)	Decryptio n (µs)	Decoding (µs)	Total Time (µs)
4096	65	1450	3210	716	55	5436
8192	128	3975	11052	2234	98	17487
16384	264	13412	47461	8883	196	70216

Table 4 shows the BFV squaring of a vector which is nearly the same concept of multiplying the two vectors but with only encoding and encrypting one vector which decreases the total time compared to multiplication.

For the BFV scheme, it allows the user to pack integers into one plaintext polynomial, and to operate on those integers in a SIMD (Single Instruction Multiple Data) manner called batch encoding. However, in the CKKS scheme, it is also encoded to polynomials but in a different technique than BFV and using different parameters. The theory behind the CKKS encoding is completely different than BFV although both encoding functions looks similar to each other.

Table 5: CKKS Multiplication

Vectors size	**Encoding (µs)**	**Encryption (µs)**	**Multiplication (µs)**	**Decryption (µs)**	**Decoding (µs)**	**Total Time (µs)**
4096	8907	9991	1451	298	3989	24636
8192	18094	19589	2924	556	8470	49633
16384	37806	41156	5973	1253	18277	104465

As it is shown in Table 5 the CKKS multiplication performed between two vectors and the time consumption of every operation done to the vectors. The two most operations consumed time are the encoding and encryption as shown in the table.

Table 6: CKKS Addition

Vectors size	**Encoding (µs)**	**Encryption (µs)**	**Addition (µs)**	**Decryption (µs)**	**Decoding (µs)**	**Total Time (µs)**
4096	8907	9991	56	330	3767	23051
8192	18094	19589	120	552	8063	46418
16384	37806	41156	315	1133	16826	97236

In Table 6, CKKS addition between two vectors is performed and it is shown that the time consumption of the addition compared to the multiplication is small. The total time difference between the multiplication and addition between the two vectors is small which is 7% average larger than addition in the CKKS scheme.

Table 7: CKKS Squaring

Vectors size	**Encoding (µs)**	**Encryption (µs)**	**Squaring (µs)**	**Decryption (µs)**	**Decoding (µs)**	**Total Time (µs)**
4096	4287	5078	680	904	3964	14913
8192	9079	9782	1378	1725	8125	30107
16384	18840	20446	2884	3844	16913	62927

Finally, the time consumption of the CKKS squaring operation shown in Table 7. The two most operations consumed time are the encoding and the encryption

As shown in Table 3, 4, 5,6,7, and 8 the timings are calculated by taking the average of 50 times for each function needed to calculate the operation in µs for each scheme done which are: encoding, encrypting, the operation done (multiplying or adding or squaring) to the vectors then decrypting and decoding the resulted vector in each scheme. Tables 3,4 and 5 include the multiplication, addition, and squaring operations done to the BFV scheme subsequently. In addition to Tables 6,7 and 8 with the same operations but done in the CKKS scheme. The tables show the difference in timings between the two schemes and the figures below show the operation timings done in each scheme compared with each other. Figure 1 illustrates the timing difference between the two schemes in the addition operation. The results show that the BFV scheme has less timing in all vector sizes than the CKKS scheme. In Figure 2, squaring operation compared between the two schemes which shows that when the size of the vectors was 4096 and 8192 the BFV scheme had less time consumption than the CKKS scheme. Meanwhile, the 16384 vector size showed that CKKS had slightly less time consumption than BFV scheme. In Figure 3, Multiplication is compared between the two schemes and it showed that the BFV scheme had less time than the CKKS scheme.

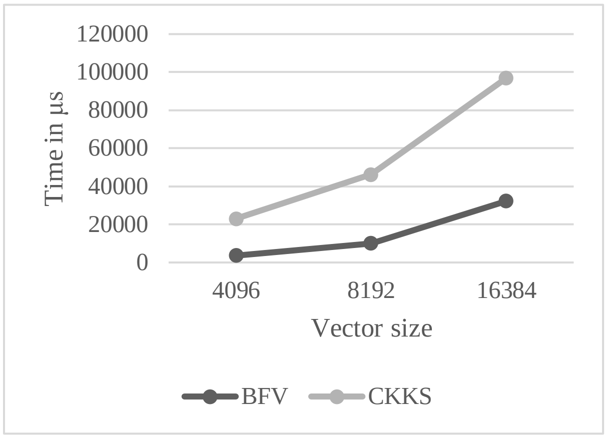


Figure 1. Addition Time in BFV vs CKKS

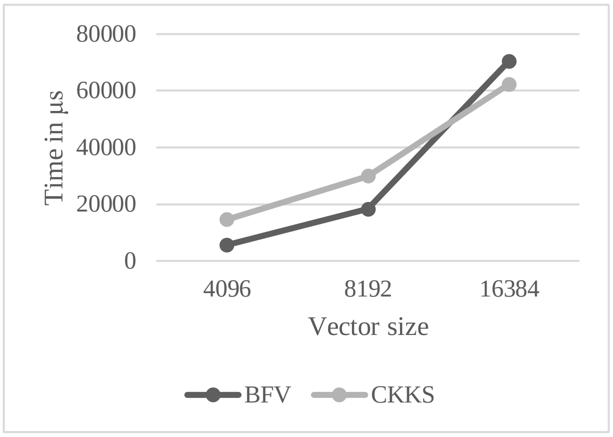


Figure 2. Squaring Time in BFV vs CKKS

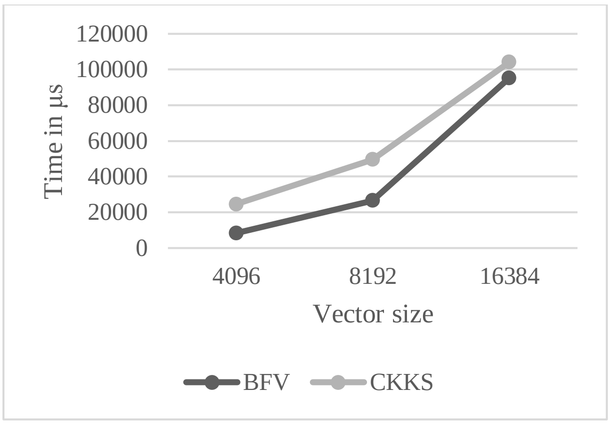


Figure 3. Multiplication Time in BFV vs CKKS

As it is mentioned before that noise should be taken into consideration especially in the multiplication process. For a sequential multiplication, the remaining noise budget should be observed and controlled because every operation done increases the noise in the ciphertext. The below figure shows the relation between the allowed number of times for the sequential multiplication and the vector size. In the BFV scheme, before reaching the data corruption due to the noise increase after every multiplication is done, it succeeded in multiplying the vectors 3 sequential times for 4096 slots in the vector using the relinearization technique. Relinearization is a procedure that decreases the size of a ciphertext back to its original size, 2. Even though relinearization has a large computational cost, relinearizing one or both input ciphertexts before the next multiplication can have a big positive influence on both noise growth and performance. Both the BFV and the CKKS methods use relinearization in the same way. However, in the CKKS scheme with the same vector slots, the calculated number of sequential multiplications was 2 times. The rest of the vector sizes is shown in figure 4. When the noise budget reaches zero, decryption cannot be guaranteed to produce the desired output. The noise budget for homomorphic operations on large ciphertexts is substantially higher than for calculations on tiny ciphertexts. Computing on smaller ciphertexts is also substantially less expensive computationally. All our experiments were conducted on an Intel(R) Core (TM) i7-8750H CPU @2.20GHz, Nvidia GeForce GTX 1050 Ti.

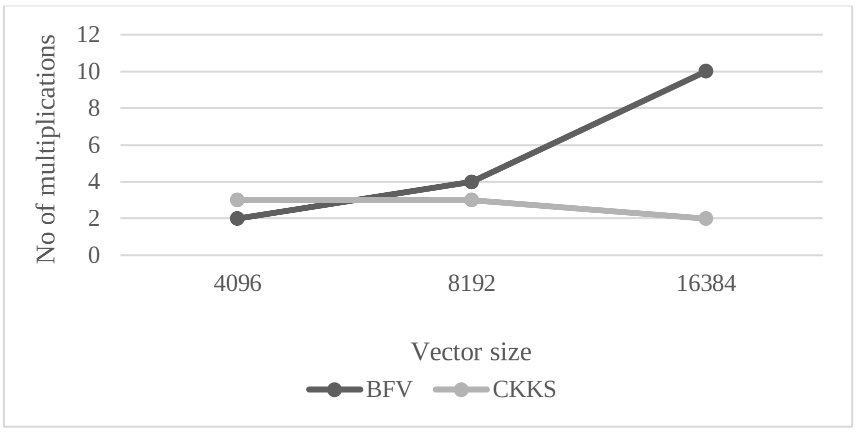


Figure 4. Maximum Number of Sequential Multiplications in BFV and CKKS Scheme

In Figure 4, the number of sequential multiplications that each scheme can handle according to the vector size is shown compared with each other. For 4096 vector size, The BFV scheme handled 2 sequential multiplication times. Meanwhile, CKKS handled 3 times which is better than BFV. In larger vector sizes, such as 8192 and 16384, the CKKS handled fewer number of sequential multiplication compared to the BFV scheme.

4.0 Conclusion and Future Work

This paper studies the performance of two of the Fully homomorphic encryption schemes, BFV and CKKS using the FHE library "Microsoft SEAL" by applying certain arithmetic operations and analyzing its performance by calculating the time consumed and observing the noise loaded especially in sequential operations. The future work is to compare more libraries such as Lattigo and HElib so it can be used with the suitable scheme in machine learning applications. In addition, in the field of computational linguistics, performing secure neural machine translation securely without exposing any text will be translated using homomorphic coding schemes that will be focused on in future work.

References

[1] V. F. Rocha, J. López, and V. Falcão Da Rocha, "An Overview on Homomorphic Encryption Algorithms," 2019.

[2] M. Ogburn, C. Turner, and P. Dahal, "Homomorphic encryption," *Procedia Comput. Sci.*, vol. 20, pp. 502–509, 2013, doi: 10.1016/j.procs.2013.09.310.

[3] S. S. Sathya, P. Vepakomma, R. Raskar, R. Ramachandra, and S. Bhattacharya, "A Review of Homomorphic Encryption Libraries for Secure Computation," no. December, 2018, [Online]. Available: http://arxiv.org/abs/1812.02428.

[4] C. Gentry, "A Fully Homomorphic Encryption Scheme," *Proc. 41st Annu. ACM Symp. Theory Comput.*, no. September, p. 169, 2009, [Online]. Available: http://cs.au.dk/~stm/local-cache/gentry-thesis.pdf.

[5] N. N. Kucherov, M. A. Deryabin, and M. G. Babenko, "Homomorphic Encryption Methods Review," *Proc. 2020 IEEE Conf. Russ. Young Res. Electr. Electron. Eng. EIConRus 2020*, no. June, pp. 370–373, 2020, doi: 10.1109/EIConRus49466.2020.9039110.

[6] L. B. Pulido-Gaytan, A. Tchernykh, J. M. Cortés-Mendoza, M. Babenko, and G. Radchenko, *A*

Survey on Privacy-Preserving Machine Learning with Fully Homomorphic Encryption, vol. 1327. Springer International Publishing, 2021.
[7] V. V. Zvika Brakerski, “Efficient Fullly Homomorphic Encryption From (Standard) LWE,” *Spec. Sect. Fifty-Second IEEE Annu. Symp. Found. Comput. Sci.*, vol. 52, no. 5, pp. 2530–2554, 2014.
[8] J. Fan and F. Vercauteren, “Somewhat Practical Fully Homomorphic Encryption,” *Proc. 15th Int. Conf. Pract. Theory Public Key Cryptogr.*, pp. 1–16, 2012, [Online]. Available: https://eprint.iacr.org/2012/144.
[9] C. Aguilar Melchor, M. O. Kilijian, C. Lefebvre, and T. Ricosset, “A comparison of the homomorphic encryption libraries HElib, SEAL and FV-NFLlib,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11359 LNCS, pp. 425–442, 2019, doi: 10.1007/978-3-030-12942-2_32.
[10] L. Xu, L. Chen, N. Shah, Z. Gao, Y. Lu, and W. Shi, *Financial Cryptography and Data Security*, vol. 10323 LNCS. 2017.
[11] J. Domingo-Ferrer, O. Farràs, J. Ribes-González, and D. Sánchez, “Privacy-preserving cloud computing on sensitive data: A survey of methods, products and challenges,” *Comput. Commun.*, vol. 140–141, no. March, pp. 38–60, 2019, doi: 10.1016/j.comcom.2019.04.011.
[12] R. Gennaro, S. Halevi, and T. Rabin, “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1592, pp. 123–139, 1999, doi: 10.1007/3-540-48910-X_9.
[13] T. Elgamal, “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms,” *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, 1985, doi: 10.1109/TIT.1985.1057074.
[14] D. Catalano and D. Fiore, “Using linearly-homomorphic encryption to evaluate degree-2 functions on encrypted data,” *Proc. ACM Conf. Comput. Commun. Secur.*, vol. 2015-Octob, pp. 1518–1529, 2015, doi: 10.1145/2810103.2813624.
[15] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie, “Private Machine Learning Classification Based on Fully Homomorphic Encryption,” *IEEE Trans. Emerg. Top. Comput.*, vol. 8, no. 2, pp. 352–364, 2020, doi: 10.1109/TETC.2018.2794611.
[16] H. Q. Ahmed Aziz, Ayman Abu Samra, “Using Homomorphic Cryptographic Solutions on E-voting Systems,” *Int. J. Comput. Netw. Inf. Secur.*, no. January, 2018, doi: 10.5815/ijcnis.2018.01.06.
[17] K. Hariss, H. Noura, and A. E. Samhat, “An efficient fully homomorphic symmetric encryption algorithm,” *Multimed. Tools Appl.*, pp. 12139–12164, 2020.
[18] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic Encryption for Arithmetic of Approximate Numbers.”
[19] A. Carey and A. N. Carey, “ScholarWorks @ UARK Computer Science and Computer Engineering On the Explanation and Implementation of Three Open-Source Fully Homomorphic Encryption Libraries On the Explanation and Implementation of Three Open-Source Fully Homomorphic Encryption Librari,” 2020.
[20] J. A. Garay, R. G. Eds, and D. Hutchison, *Advances in Cryptology – Crypto 2014*. 2014.
[21] C. Mouchet, J. Bossuat, and J. Troncoso-pastoriza, *Lattigo : a Multiparty Homomorphic Encryption Library in Go*, vol. 1, no. 1. Association for Computing Machinery, 2020.
[22] G. W. R. Yuriy Polyakov, Kurt Rohloff, “PALISADE Lattice Cryptography Library User Manual,” *Cybersecurity Res. Center, New Jersey Inst. ofTechnology*, 2017.