Practical Aspects of Using Fully Homomorphic Encryption Systems to Protect Cloud Computing

Anna Ilyenko[1], Sergii Ilyenko[1], Olena Prokopenko[1], Hennadii Hulak[2,3], and Iryna Melnyk[2]

[1] National Aviation University, 1 Liubomyra Huzara ave., Kyiv, 03058, Ukraine
[2] Borys Grinchenko Kyiv University, 18/2 Bulvarno-Kudriavska str., Kyiv, 04053, Ukraine
[3] National Academy of the Security Service of Ukraine, 22 Mykhaila Maksymovycha str., Kyiv, 03022, Ukraine

Abstract

Fully homomorphic encryption schemes are the most promising area of cryptographic information security, particularly in cloud computing. Over the last ten years, Fully Homomorphic Encryption (FHE) has moved from a theoretical idea to practical implementation in real-world cryptographic applications. The concept of homomorphic encryption is ideal for providing secure cloud computing, where user data will never be in plaintext at any stage of its processing. However, there are still many problems related to the performance and complexity of computing that need to be overcome. To confirm the effectiveness of homomorphic encryption in the cloud, a cryptographic cloud computing protection application based on homomorphic encryption was developed. Based on a detailed analysis of existing FHEs, studying their mathematical apparatus, and classifying them according to various criteria, two schemes were selected for implementation in the application—CKKS and BFV, which allow to performance of homomorphic processing of encrypted data. The proposed solution demonstrates a new approach to the design of FHE applications, where the user independently chooses the parameters for implementing the FHE scheme, according to his requirements. The proposed test local server allows to testing of selected scheme parameters by combining the execution of various homomorphic computations. Based on the tests, it is possible to customize the proposed application according to one's tasks, sacrificing performance and security for the ability to perform more complex homomorphic computations, or vice versa, or even to maintain a balance between them.

Keywords

Fully homomorphic encryption, BFV schema, CKKS schema, cloud computing.

1. Introduction

By using cloud services, the user assigns the task of ensuring the integrity, availability, and confidentiality of data to the cloud provider. At the same time, ensuring data confidentiality using standard encryption methods is ineffective, because the server must first decrypt the data to perform data processing [1–3]. This necessity creates the problem of key distribution, as well as the problem of possible theft of data decrypted by the server and processed in plaintext.

An effective solution to the shortcomings of standard cryptosystems is the use of homomorphic schemes that allow the cloud server to process encrypted client data while obtaining the result that would be obtained by performing the same operations on open data. Thus, the data is not in the open form at all stages of processing.

This paper aims to demonstrate how fully homomorphic encryption can be used to ensure data privacy in cloud computing. The development of the author's cryptographic application for the protection of cloud computing based on homomorphic encryption can

CPITS-2023-II: Cybersecurity Providing in Information and Telecommunication Systems, October 26, 2023, Kyiv, Ukraine
EMAIL: ilyenko.a.v@nau.edu.ua (A. Ilyenko); serhii.ilienko@npp.nau.edu.ua (S. Ilyenko) bortnik.olena.v@nau.edu.ua (O. Prokopenko); h.hulak@kubg.edu.ua (H. Hulak); iy.melnyk@kubg.edu.ua (I. Melnyk)
ORCID: 0000-0001-8565-1117 (A. Ilyenko); 0000-0002-0437-0995 (S. Ilyenko); 0000-0001-9895-888X (O. Prokopenko); 0000-0001-9131-9233 (H. Hulak); 0000-0001-6041-6145 (I. Melnyk)

show how exactly it is possible to implement a fully homomorphic encryption scheme in the model of client-server interaction, analyze the performance of encrypted data processing in comparison with conventional encryption methods, and also track the correctness of the results of the calculations performed on the encrypted data when decrypting them.

The work aims to study and test an approach to cryptographic protection of cloud computing based on homomorphic encryption. The practical value is the creation of an author's cryptographic application for cloud computing protection based on fully homomorphic encryption using CKKS and BFV schemes. This is an application with a windowed interface written in the Python programming language using the capabilities of the Microsoft SEAL cryptographic library, which allows the user to independently select the parameters of the homomorphic scheme implementation and investigate its performance and the correctness of homomorphic computations over encrypted data on a local cloud server.

2. Theoretical Approaches to Security based on Fully Homomorphic Computing

The National Institute of Standards and Technology (NIST) defines cloud computing as a model for providing convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications, and services) that can be quickly provisioned and released with minimal effort by the administrator or service provider.

Security risk analysis in cloud computing should consider the risks of storing data in different locations and the risks of data distribution between employees, and the level of risk significance may depend on the cloud architecture model under consideration [4–5].

Based on the ENISA report, the following categories of cloud computing security risks exist organizational risks: loss of management, loss of business reputation, compliance issues, stoppage or failure of a cloud service, etc.; technical risks: data protection risks, resource exhaustion (under- or over-utilization), isolation failure, malicious insider (abuse of high privileges), data interception, data leakage, DDoS, conflict between client procedures.

Using cloud services allows users to access computing resources anywhere and anytime. Considering the obvious advantages of using cloud services, there are also obvious disadvantages.

The main disadvantage of using cloud services is that the user provides their data to the cloud provider and relies on the service provider to ensure the appropriate level of confidentiality, integrity, and authenticity of the user's data through encryption, hashing, and other cryptographic mechanisms. However, most cloud services require user data to be decrypted before it is processed. Therefore, if the user does not trust the security mechanisms offered by the provider, he encrypts the data before transferring it to the cloud and must transfer the private key along with the encrypted data to the cloud server, which in turn uses this key to decrypt and further process the decrypted data. In this scheme of interaction between the client and the server, the secret key can be stolen during its transmission. Data that has been decrypted by the server for processing can also be stolen [6–8].

The use of homomorphic encryption eliminates the possibility of compromising the private encryption key since it does not need to be transferred to the server, as well as the possibility of stealing open data during processing because the data is not decrypted during processing and is never stored on the cloud server in plaintext. That is why this technology is the most promising in the field of cloud computing security [9].

Homomorphic encryption allows the cloud service provider to perform certain computational functions on the data even when it is encrypted. With traditional encryption schemes, customers must compromise the security of their data by providing a private decryption key to the service provider to use cloud services, as traditional schemes do not allow the provider to work with encrypted data [10–14].

The use of homomorphic cryptosystems allows to exclude from the algorithm of interaction between the user and the cloud server the stages of transferring the secret key to the cloud provider with the subsequent decryption of data on the cloud server for processing, so the interaction algorithm shows in Fig. 1 looks like this: The user creates a

message m that must be sent and processed on the cloud server. The user generates a key pair—a public key Pk and a private key Sk using some asymmetric encryption algorithm. The user encrypts the plaintext message m using the public key Pk. The user sends the received cryptogram c to a remote cloud server. The server processes the encrypted text c. The server sends the processed encrypted text c to the user. The user decrypts the processing result received from the server with a private key.

As we can see, the use of homomorphic cryptosystems has made it possible to reduce the algorithm of interaction between the user and the cloud server by two stages: there is no longer a need to send a private key to the server, which eliminates the possibility of its compromise, and to decrypt data on the server, which eliminates the possibility of its theft.

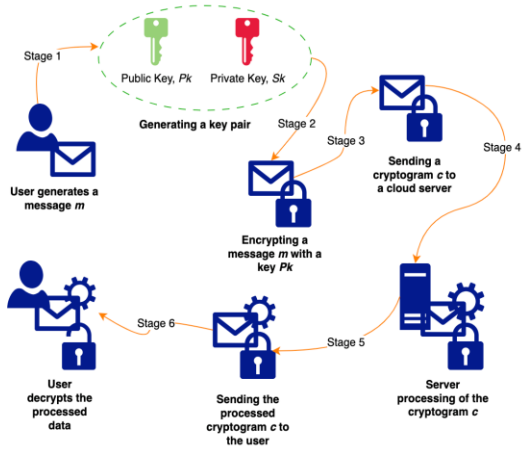

Figure 1: General algorithm of interaction between a client and a cloud server using homomorphic encryption schemes

Modern fully homomorphic encryption schemes (hereinafter referred to as FHE, i.e. Fully Homomorphic Encryption) date back to 2009, when Craig Gentry presented the first possible FHE design. Later, Gentry developed the idea of fully homomorphic encryption and now there are many modern schemes [15–18].

Any homomorphic encryption scheme consists of four algorithms, namely:

1. Key generation algorithm (KeyGen): accepts as input some parameters that depend on the encryption scheme, and as output receives a pair—a public key (Pk) and a private key (Sk).
2. Encryption algorithm (Enc): accepts as input the plaintext m and the public encryption key Pk. The output is a cryptogram (1):

$$c=\mathrm{Enc}(m) \quad (1)$$

3. Decryption algorithm (Dec): accepts a cryptogram c and a private decryption key Sk as input. The output is a decrypted message (2):

$$\mathrm{Dec}(c)=m \quad (2)$$

4. Evaluation algorithm (Eval): accepts a pair of ciphers (c_1, c_2) as input and evaluates the f() function over the ciphers to get the result (3):

$$f(\mathrm{Enc}(m_1,m_2))=f(\mathrm{Dec}(c_1,c_2)=f(m_1,m_2) \quad (3)$$

The paper focuses on practical schemes for complete homomorphic encryption systems, namely: Brakerski-Fon-Vercauteren, BFV, and Cheon-Kim-Kim-Song, CKKS. These schemes work based on levels where there is a noise parameter that allows only a limited number of multiplication operations to be performed until correct decryption is impossible. After all, in non-binary schemes, the problem of multiplication depth is now acute, which is solved differently in different schemes—linearization and modules switching in BFV, change of scale in CKKS, etc. Despite all the existing problems, at this stage, homomorphic schemes can provide an appropriate level of security and a sufficient level of computing performance, offering to solve various problems with binary and non-binary schemes, integer and floating point schemes, etc. [7, 19–21].

To compare the homomorphic schemes, the sizes of the public and private keys, the size of the ciphertext, and three parameters are given for each of them: λ is security parameter, n is grid size, and p is a power of two.

The sizes of the public and private keys, as well as the ciphertext for each of the schemes under consideration are shown in Table 1.

Table 1
Comparison of key pair and cryptogram sizes of the considered FHEs

Scheme	Public key size	Private key size	Ciphertext size
Gentry	n^7	n^3	$n^{1.5}$
DGHV	$\vartheta(\lambda^{10})$	$\vartheta(\lambda^2)$	$\vartheta(\lambda^5)$
BGV	2pn log q	2p log q	2p log q
BFV	2p log q	p	2p log q
CKKS	$\vartheta(n)$	$\vartheta(n^2)$	$\vartheta(n \log n)$
GSW	$\vartheta(n^2\log^2 q)$	$(n+1)$ log q	$(n+1)^2\log^3 q$

3. Practical Aspects of Implementing Fully Homomorphic Systems

For practical testing of homomorphic cryptographic systems for cloud computing, a cloud server was deployed, a leveled fully homomorphic BFV and CKKS schemes were selected, and the Microsoft SEAL library was selected. The designed cryptographic module uses Pyfhel, which provides a Python shell for the Microsoft SEAL library that can be extended with other C++ libraries and goes beyond simply exposing the core API by adding a carefully crafted abstraction layer that is easy to use in Python [22–24]. The proposed solution demonstrates a new approach to the design of FHE applications, where the user independently chooses the parameters for implementing the FHE scheme, according to their requirements.

To evaluate the performance of the implemented BFV and CKKS schemes, performance tests of homomorphic addition and multiplication for three values of n were performed, followed by decryption, decoding, and verification of the correctness of homomorphic addition and multiplication.

Based on the practical tests, a comparative Table 2 was formed, which demonstrates the performance of the BFV scheme concerning the addition and multiplication operations.

So, changing the value of n does not affect the performance of the encoding operation, and the encryption operation slows down by 70% for each increase in *n*.

Table 2
Performance of homomorphic addition of the BFV scheme at different values of *n*

Module *n*	**8192**	**16384**	**32768**
Encoding, s	0.001	0.001	0.001
Encryption, s	0.015	0.049	0.167
Addition, s	0.002	0.010	0.046
Decryption, s	0.003	0.009	0.040
Decoding, s	0.002	0.004	0.008
Total time, s	0.023	0.073	0.262

The decoding speed decreases by 50%, and all other operations by about 80% with each increase in the value of the polynomial module. The homomorphic calculations were performed correctly, and the decrypted result coincides with the manually calculated one, so the noise level was greater than 0.

As with the homomorphic addition operation, the results of the homomorphic multiplication performance testing for the BFV scheme are presented in Table 3.

Table 3
Performance of homomorphic multiplication of the BFV scheme at different values of *n*

Module *n*	**8192**	**16384**	**32768**
Encoding, s	0.001	0.001	0.001
Encryption, s	0.015	0.049	0.167
Addition, s	0.009	0.049	0.253
Decryption, s	0.002	0.009	0.040
Decoding, s	0.003	0.004	0.008
Total time, s	0.030	0.112	0.469

According to the results of testing the speed of operations during homomorphic multiplication, the decoding speed decreases by 50%, and all other operations by about 80% with each increase in the value of the polynomial module. The homomorphic calculations were performed correctly, and the decrypted result coincides with the manually calculated one, so the noise level was greater than 0.

The percentage reduction in performance is approximately the same for both multiplication and addition operations in the BFV scheme.

However, the absolute speed of multiplication operations is lower than that of addition, due to the need to perform relinearization to reduce the polynomial degree of the encrypted text and module switching operations to reduce noise. Thus, the speed of addition and multiplication operations when n is equal to 8192 is almost the same, but if the value of n increases, the performance of the multiplication operation drops evenly by 70%.

So, the most computationally complex operations are homomorphic operations, which take much longer than encryption, decryption, encoding, and decoding operations. It is obvious that the noise level increases much faster when performing homomorphic multiplication, so with several consecutive multiplication operations or large number multiplication operations, the noise level can drop to zero and correct decryption will be impossible. However, to perform the calculations required by the test, even the minimum value of n equal to 8192 is enough to ensure that the results of addition and multiplication are correct.

Based on the practical tests, comparative Table 4 is formed, which demonstrates the performance of the CKKS scheme concerning the addition operation.

Table 4
Performance of homomorphic addition of the CKKS scheme at different values of *n*

Module *n*	8192	16384	32768
Encoding, s	0.001	0.007	0.030
Encryption, s	0.014	0.046	0.190
Addition, s	0.006	0.010	0.045
Decryption, s	0.001	0.002	0.006
Decoding, s	0.003	0.013	0.075
Total time, s	0.025	0.078	0.350

From the performed tests, it can be seen that when moving from n equal to 8192 to *n* equal to 16384, the encoding speed drops by 86% and the encryption speed by 70%, with a further transition to the value of n equal to 32768, both operations reduce their performance by about 75%. At the same time, the decryption speed steadily decreases by 50% in proportion to the increase in the value of the polynomial module n. Decoding operation is 80–85% slower on average. Homomorphic addition, when changing the value from n equal to 8192 to n equal to 16384, reduces the efficiency of calculations by 70%, and with further growth of n, the performance drops by half, i.e. to 80%. All three values of n for the CKKS scheme provide a sufficient noise budget for the tested addition operations, so the operations are performed correctly.

As with the homomorphic addition operation, the results of testing the performance of homomorphic multiplication for the CKKS scheme are presented in Table 5.

Table 5
Performance of homomorphic multiplication of the CKKS scheme at different values of n

Module n	**8192**	**16384**	**32768**
Encoding, s	0.001	0.007	0.030
Encryption, s	0.014	0.046	0.190
Addition, s	0.009	0.046	0.289
Decryption, s	0.001	0.001	0.006
Decoding, s	0.004	0.013	0.063
Total time, s	0.029	0.113	0.578

With homomorphic multiplication in the CKKS scheme, decryption at the first two values of the polynomial modulus n: 8192 and 16384 does not change, but performance drops sharply by 85% when *n* is equal to 32768. At the same time, with an increase in the value of *n*, the speed of homomorphic multiplication decreases by 83% on average, and decoding—by 75%.

Increasing the value of n has a more significant effect on the decrease in multiplication performance than addition. This is primarily because multiplication operations take more time than addition. After all, homomorphic multiplication additionally requires a linearization operation to reduce the degree of the ciphertext polynomial, as well as a scaling operation of the scale factor in the CKKS scheme to reduce noise. The linearization operation and scale switching of the scale factor are computationally expensive, which affects the performance of homomorphic multiplication.

However, what these two operations have in common is that when moving from n equal to 8192 to n equal to 16384, there is a uniform drop in the performance of all operations, but when *n* is equal to 32768, there is a sharp jump in the speed of all operations, of course, except for encryption and encoding operations.

It is seen that, as in the BFV scheme, the performance of multiplication and addition operations in CKKS when *n* is equal to 8192 is almost the same, but with the growth of n, the multiplication performance gradually decreases, first by 75%, and then by another 80%, which is more significant than in the BFV scheme. The graphs clearly show that when performing the cycle of homomorphic addition operations, the BFV scheme is more productive, where the total time for performing the sequence of operations: encryption, encoding, addition, decryption, and decoding is less than that of CKKS. With n equal to 8192 and n equal to 16384, the performance of addition in CKKS and BFV is almost the same, but with n equal to 32768, homomorphic addition in BFV is 20% faster.

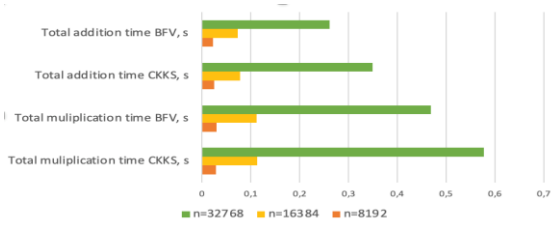

Figure 2: Performance of homomorphic schemes BFV and CKKS

As we can see, the situation is similar to the one when comparing the performance of homomorphic multiplication, the overall performance of the BFV scheme is higher than that of CKKS. With n equal to 8192 and n equal to 16384, the multiplication performance of CKKS and BFV is almost the same, but with n equal to 32768, the homomorphic multiplication in BFV is 20% faster.

At the same time, the speed of the addition operation is much faster for both schemes than the multiplication operation. As explained earlier, this is because the multiplication operation is much more complex than the addition operation. After all, there is a concept of multiplication depth and noise level, which grows much faster in homomorphic multiplication than in addition. Therefore, to perform homomorphic multiplication in both schemes, additional operations are performed, such as linearization, which reduces the degree of the ciphertext polynomial, module switching, to reduce the noise level in the BFV scheme, and the scaling operation in CKKS, to reduce the degree of the scale factor and noise level.

The testing proved that even with n equal to 8192, the noise level is sufficient to perform at least one addition and multiplication operation. That is, the larger the value of n, the more consecutive multiplications the scheme supports, but the lower its performance. It is also worth taking into account the peculiarities of the implementation of the schemes—if the noise level reaches zero, then correct decryption of the cryptogram using the BFV scheme will be impossible, while at the same time since the CKKS scheme is based on "approximate calculations," a high noise level will significantly affect the accuracy of calculations.

In terms of performance, testing has shown that the BFV scheme is faster when performing homomorphic multiplication and addition operations than CKKS. As for the accuracy of calculations, the very fact that the CKKS scheme is based on the concept of approximate calculations makes it necessary to use the BFV scheme to perform accurate calculations. However, it is worth noting that CKKS is the only option for working with floating-point numbers.

Thus, the choice of the CKKS and BFV schemes in most cases depends on the type of data to be processed: integers or floating point numbers, after testing the performance of homomorphic calculations, we can say that if you need to ensure a small level of multiplication depth, then for values of n below 32768, the performance of these schemes is the same.

4. Conclusions

The practical implementation of the designed cryptographic application of cloud computing protection based on homomorphic encryption can provide a comprehensive solution for protecting cloud infrastructure while maintaining a balance between the required level of security and computing performance, as well as the number and complexity of homomorphic computing. These capabilities exist due to the use of the modern SEAL cryptographic library as the basis of the application architecture, which provides tools for flexible implementation and customization of CKKS and BFV schemes:

- At $n = 8192$ and $n = 16384$, the addition performance in CKKS and BFV is the same, at $n = 32768$, homomorphic addition in BFV is 20% faster.
- At $n = 8192$ and $n = 16384$, the multiplication performance of CKKS and BFV is the same, at $n = 32768$, homomorphic multiplication in BFV is 20% faster.
- At the same time, the speed of the addition operation is much faster for both schemes than the multiplication operation. This is because to perform homomorphic multiplication, both schemes perform additional operations, such as linearization, which reduces the degree of the ciphertext polynomial, module switching to reduce the noise level in the BFV scheme, as well as the CKKS scaling operation to reduce the degree of the scale factor and noise level.

The flexibility of customizing the program module to meet the specific needs of the user is provided by a user-friendly graphical interface, where one can choose: the required FHE scheme, depending on the type of data that the user will work with; a polynomial module n, which is chosen small if maximum performance is required and large if there is a need to

perform complex homomorphic calculations; a security parameter that affects the level of security but does not affect the performance and complexity of homomorphic computing.

Along with the advantages, there are also obvious disadvantages and unrealized opportunities that allow the application to be enhanced in the future. Among these, it is possible to highlight the inability to see and control the noise level—the user learns that the noise level has reached the limit after which correct decryption is impossible only after decryption is performed and the incorrectness of the calculations is assessed.

References

[1] A. Bessalov, et al., Multifunctional CRS Encryption Scheme on Isogenies of Non-Supersingular Edwards Curves, in: Workshop on Classic, Quantum, and Post-Quantum Cryptography, vol. 3504 (2023) 12–25.

[2] V. Sokolov, P. Skladannyi, H. Hulak, Stability Verification of Self-Organized Wireless Networks with Block Encryption, in: 5th International Workshop on Computer Modeling and Intelligent Systems, vol. 3137 (2022) 227–237.

[3] A. Bessalov, et al., Implementation of the CSIDH Algorithm Model on Supersingular Twisted and Quadratic Edwards Curves, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, vol. 3187, no. 1 (2022) 302–309.

[4] N. Roy, R. Jain, Cloud Computing: Architecture and Concept of Virtualization, J. Sci. Technol. Manag. 4 (2015) 2394–1537.

[5] Z. Balogh, M. Turčáni, Modeling of Data Security in Cloud Computing, IEEE Systems Conference (2016) 1–6. doi: 10.1109/syscon.2016.7490658.

[6] D. Chen, H. Zhao. Data Security and Privacy Protection Issues in Cloud Computing, IEEE Int. Conf. Comput. Sci. Electron. Eng. 1 (2012) 641–651. doi: 10.1109/iccsee.2012.193.

[7] J. Cheon, et al. Homomorphic Encryption for Arithmetic of Approximate Numbers, 23rd Int. Conf. Theory Appl. Cryptol. Inf. Secur. (2017) 409–437. doi: 10.1007/978-3-319-70694-8_15.

[8] D. Meng, Data Security in Cloud Computing, IEEE Int. Conf. Comput. Sci. Educ. (2013) 810–813. doi: 10.1007/978-1-4614-3872-4_103.

[9] P. Anakhov, et al., Evaluation Method of the Physical Compatibility of Equipment in a Hybrid Information Transmission Network, Journal of Theoretical and Applied Information Technology 100(22) (2022) 6635–6644.

[10] B. Schneier, Applied Cryptography: Protocols, Algorithms and Source Code in C, 2nd Edition, Wiley (2015).

[11] C. Gentry, A Fully Homomorphic Encryption Scheme, Stanford University (2009).

[12] A. Acar, et al. A Survey on Homomorphic Encryption Schemes: Theory and Implementation, ACM Computing Surveys 51(4) (2018) 1–35.

[13] A. Ilyenko, S. Ilyenko, Program Module of Cryptographic Protection Critically Important Information of Civil Aviation Channels, Int. Conf. Comput. Sci. Eng. Educ. Appl. (2022) 235–247.

[14] S. Kazmirchuk, et al., Improved Gentry's Fully Homomorphic Encryption Scheme: Design, Implementation and Performance Evaluation, CybHyg (2019) 72–83.

[15] C. Gentry, Fully Homomorphic Encryption Using Ideal Lattices, 41st ACM Symposium on Theory of Computing (2009) 169–178. doi: 10.1145/1536414.1536440.

[16] C. Gentry, Toward Basing Fully Homomorphic Encryption on Worst-Case Hardness, Annual Cryptology Conference (2010) 116–137. doi: 10.1007/978-3-642-14623-7_7.

[17] [C. Gentry, S. Halevi, Implementing Gentry's Fully-Homomorphic Encryption Scheme, Annual Int. Conf. Theory Appl. Cryptogr. Techniques (2011) 129–148. doi: 10.1007/978-3-642-20465-4_9.

[18] C. Gentry, S. Halevi, Implementing Gentry's Fully-Homomorphic Encryption Scheme, Cryptology ePrint Archive, Report (2010).

[19] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (Leveled) Fully Homomorphic

Encryption Without Bootstrapping, ACM Transactions on Computation Theory (TOCT) 6(3) (2014) 1–36. doi: 10.1145/2633600.
[20] X. Sun, et al., Private Machine Learning Classification based on Fully Homomorphic Encryption, IEEE Transactions on Emerging Topics in Computing 8(2) (2018) 352–364.
[21] D. Stehlé, R. Steinfeld, Faster Fully Homomorphic Encryption, Int. Conf. Theory Appl. Cryptol. Inf. Secur. (2010) 377–394. doi: 10.1007/978-3-642-17373-8_22.
[22] A. Titus, et al., PySEAL: A Python Wrapper Implementation of the SEAL Homomorphic Encryption Library, arXiv (2018).
[23] Y. Polyakov, K. Rohloff, G. Ryan. PALISADE Lattice Cryptography Library User Manual, Tech. Rep. (2019).
[24] S. Erabelli, pyFHE-a Python Library for Fully Homomorphic Encryption, Ph. D. Dissertation, Massachusetts Institute of Technology (2020).