Performance comparison of homomorphic encryption scheme implementations

Goran Đorđević, *AET Europe The Netherlands, ETF Beograd, Milan Marković, Panevropski Univerzitet Apeiron Banja Luka, Pavle V. Vuletić, ETF Beograd*

***Abstract* — Homomorphic Encryption allows third party to receive encrypted data and perform arbitrarily computations on that data while it remains encrypted, despite not having the secret decryption key. This enables many new secure applications in cloud environments. For a long time, a key issue with the homomorphic encryption was its low performance which made it unusable in production environments. Advances in the last ten years in the field of homomorphic encryption resulted in several new schemes and software libraries which implement them. These homomorphic schemes have improved performance, but there is still a question whether the improvements would justify their use in production environments. In this paper we evaluated features and performances of several new homomorphic encryption mechanisms: BGV, BFV and CKKS.**

***Keywords* — Homomorphic Encryption; Performance; Secure Multiparty Computation.**

I. Introduction

Homomorphic encryption allows computations on ciphertext without the knowledge of the secret key, or more precisely it allows performing computations on the encrypted data, without decrypting them [1]. Homomorphic encryption allows a third party (e.g., cloud, service provider) to perform certain computable functions on the encrypted data while preserving the features of the function and format of the encrypted data and without being able to see its content. Homomorphism of the first asymmetric encryption algorithms (RSA) over some mathematical operations (e.g. multiplication) was known since these algorithms were invented almost fifty years ago. Such schemes which support partial set of mathematical operations are known as partially homomorphic. Cryptographic mechanisms that support arbitrary level of computations on ciphertext (multiplication, addition, rotation) without the knowledge of the secret keys are known as Fully Homomorphic Encryption (FHE) systems. The increased popularity of cloud-based services on one side and the need to preserve data privacy led to the new interest in homomorphic encryption research which would enable secure multiparty computation in the cloud environment. One could imagine the use of AI or machine learning algorithms on the data which is encrypted and invisible to the AI system provider, thus preserving data privacy only for the data owner. An example of such a scenario where homomorphic encryption mechanisms are deployed is given in Figure 1. In this example the user sends and stores the data in the encrypted form on the cloud server. The data is processed on the server in the encrypted form, and the results which remain in the encrypted form are sent back to the user who can decrypt the data and use the result.

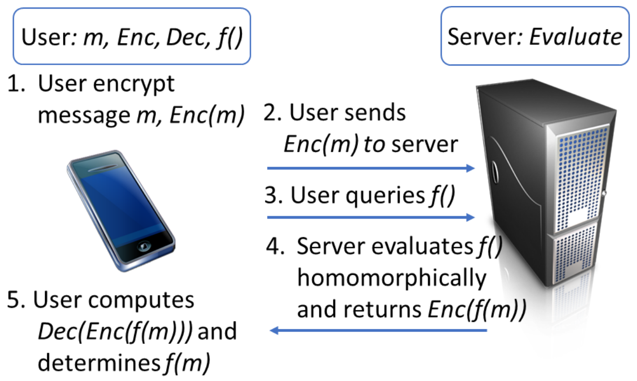
User: *m, Enc, Dec, f()*
Server: *Evaluate*
1. User encrypt message *m, Enc(m)*
2. User sends *Enc(m)* to server
3. User queries *f()*
4. Server evaluates *f()* homomorphically and returns *Enc(f(m))*
5. User computes *Dec(Enc(f(m)))* and determines *f(m)*

Fig. 1. An example of client-server HE scenario

The biggest obstacle for the use of homomorphic encryption schemes was the fact that there were no FHE mechanisms which had reasonable performance. Computations were by several orders of magnitude slower than the operation on unencrypted data which made any such solution too resource expensive. However, in the last ten years a breakthrough happened in the area and a set of new homomorphic encryption schemes emerged. Modern fully homomorphic encryption schemes use complex algorithms on lattice structures and Ring-LWE (Ring Learning With Errors) mechanism [2]. In addition to the homomorphic property, it is believed that these algorithms are resistant to quantum computer attacks because nowadays there are no known algorithms that would use the properties of quantum computers to break these algorithms in polynomial time. Following the appearance of new fully homomorphic encryption schemes, a set of programming APIs and libraries which implement different schemes emerged as well. In this paper we are assessing the set of capabilities and performance of three homomorphic encryption schemes (BGV, BFV, CKKS) and are discussing the suitability and constraints of these schemes for use in the cloud-based environments for secure multiparty computations. Performance assessment of the new FHE schemes has not been explored a lot in the literature. We believe that this paper will provide a better insight into the current state of the work on FHE and its suitability for real use case scenario deployments.

The paper is organized as follows. Section II gives an overview of the related work in the field of the performance evaluation of the FHE schemes. The most important properties of homomorphic encryption and the classification of the homomorphic encryption schemes are presented in Section III. The main features and description of modern HE schemes: BGV [3], BVF [4] and CKKS [5] are elaborated in Section IV. In Section V is shown results of experimental analysis. Conclusions are given in Section VI.

II. RELATED WORK

Related work about the FHE schemes is spread across the papers in the relevant sections, while this section contains only those papers which were dedicated to FHE performance evaluation. Experimental results related to BGV scheme with value of ciphertext modulus q=130 are given in [1]. Viand et al. in [6] compare the features of Palisade, Microsoft SEAL, and HELib homomorphic encryption libraries. In addition, this paper gives statistical compiler tests of BVF scheme implemented in the SEAL library in a graphical form without presenting precise numerical values. Melchor et al. [7] compared the performance of three libraries HELib, SEAL and FV-NFLlib for large plaintext moduli of up to 2048 bits. Finally, Lepoint et al. [8] compare the performance of two older homomorphic schemes. Unlike the previous work, in this paper we give experimental results for BGV with broader range of values ciphertext modulus q and results for other modern homomorphic schemes: BFV and CKKS that are not covered in [1].

III. PROPERTIES OF HOMOMORPHIC ENCRYPTION

There are four main types of homomorphic schemes [1]:

- Partially Homomorphic Encryption (PHE). The PHE scheme enables either any number of addition or any number of multiplication operations over encrypted data.
- Somewhat Homomorphic Encryption (SHE) allows both addition and multiplication, but it can perform a limited number of operations. “Somewhat” means it works for some functions *f*.
- Fully Homomorphic Encryption. The scheme allows any number of addition or multiplication operations. “Fully” means it works for all functions *f*. An FHE scheme can evaluate unbounded depth.
- Levelled Homomorphic Encryptions (LHE). This scheme can evaluate arbitrary polynomial-size circuits.

Homomorphic Encryption should support two main homomorphic operations:

- Additive Homomorphic Encryption;
- Multiplicative Homomorphic Encryption.

Homomorphic encryption is additive, if [9]:

$$\text{Enc}\,(m_1 + m_2) = \text{Enc}\,(m_1) + \text{Enc}\,(m_2);\ \forall m_1, m_2 \in M.$$

Homomorphic encryption is multiplicative, if [9]:

$$\text{Enc}\,(m_1 * m_2) = \text{Enc}\,(m_1) * \text{Enc}\,(m_2);\ \forall m_1, m_2 \in M.$$

The most popular classes of homomorphic schemes are (given with their main properties):

- Boolean circuit (Fastest Homomorphic Encryption in the West (FHEW) [10] and Fast Fully Homomorphic Encryption over the Torus (TFHE) [11]):
  - Plaintext data are coded as bits;
  - Computations are performed by using Boolean circuits.
- Modular integer arithmetic (BGV, BFV):
  - Plaintext data are coded as integer modulo a plaintext;
  - Computations are expressed as integer modulo arithmetic.
- Approximate number arithmetic (CKKS):
  - Plaintext data are coded as real (or complex) numbers;
  - Computations are performed in a way similar to floating-point arithmetic but dealing with fixed-point numbers.

Modern HE mechanisms are based on usage of lattice cryptography with errors LWE [12]. Lattices have an important role in modern cryptography, especially in the context of the research on post-quantum cryptography. It is known that the factoring problem which was discovered to be solvable in polynomial time on a quantum computer by Shor can be applied to the widely used asymmetric cryptographic schemes (RSA, DH). At the moment of writing this paper there was no report in the literature which claimed that it can break lattice-based cryptographic algorithms using quantum computer algorithms.

The newest HE algorithms are applied structured lattices i.e. Ring-LWE mechanism [2]. The Ring-LWE reduces key length and computation time. The ring implementation is based on power-of-two cyclotomic rings:

$$\text{R}_q = \mathbb{Z}_q \,/\, \langle \text{x}^n + 1 \rangle$$

The optimized Residue Number System (RNS) variants of algorithms show significant performance gain compared to their earlier respective implementations [13]. The RNS works with native (machine-word size) integers because it is faster than multi-precision integer arithmetic. It breaks rings of large bit-width integers into a parallel set of rings (<64-bit residues) allowing very efficient computation on 64-bit CPU architecture.

Large modulus q is represented as product of integers:

$$q = \prod_{i=1}^{k} qi$$

Modulus q is a functional parameter that determines how many computations are allowed without the appliance bootstrapping procedure [14].

One of the properties of the homomorphic encryption schemes is that they add noise to a ciphertext in the encryption process. Homomorphic operations (especially multiplication) increase the noise. If the noise becomes too large, the resultant ciphertext can become undecryptable. Noise budget is the total amount of noise that can be added until the decryption fails [15]. The bootstrapping is the procedure of "refreshing" a ciphertext by running the decryption function on it homomorphically, resulting in a reduced noise.

All considered homomorphic encryption schemes support the following homomorphic operations:

- Addition;
- Multiplication;
- Rotation.

IV. Homomorphic schemes

The BGV scheme was proposed [3]. BGV is a levelled HE scheme, meaning that the parameters of the scheme depend on the multiplicative depth that the scheme is capable to evaluate. Multiplicative depth determines how many sequential multiplications can be performed.

The BFV scheme [4] is a homomorphic cryptographic scheme based on the Ring-LWE problem in a lattice.

The CKKS scheme [5] is known as Homomorphic Encryption for Arithmetic of Approximate Numbers (HEAAN). Supported operations in the scheme are shown in Figure 2. The CKKS scheme enables computations on vectors of complex values.

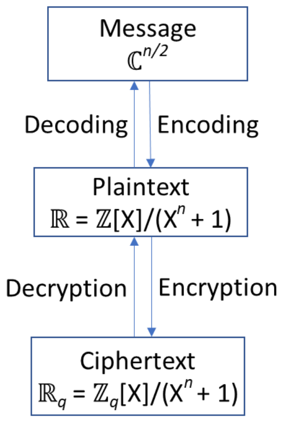
Message
$\mathbb{C}^{n/2}$
Decoding
Encoding
Plaintext
$\mathbb{R} = \mathbb{Z}[X]/(X^n + 1)$
Decryption
Encryption
Ciphertext
$\mathbb{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$

Fig. 2. Operations in CKKS

The CKKS is an approximate homomorphic encryption scheme with the following features:

- Dec (Enc(m)) ≈ m;
- Dec (ct_1 * ct_2) ≈ Dec (ct_1) * Dec (ct_2);
- Noise bounds are determined by the parameter set.

In the CKKS scheme noise is considered as a part of numerical error in approximate computation. It supports homomorphic rounding-off.

In all above-mentioned schemes the following homomorphic operations are implemented [16]:

- Public key encryption:
  $$PubEncrypt(pk, M) \rightarrow C$$
  The public encryption algorithm takes as input the public key (pk) of the scheme and any message M from the message space. The algorithm outputs a ciphertext C.
- Decryption:
  $$Decrypt(sk, C) \rightarrow M$$
  The decryption algorithm takes as input the secret key of the scheme (sk), and a ciphertext C. It outputs a message M from the message space.
- Homomorphic addition:
  $$EvalAdd(Params, ek, C_1, C_2) \rightarrow C_3$$
  EvalAdd is an algorithm that takes as input the system parameters *Params*, the evaluation key (ek), two ciphertexts C_1 and C_2, and outputs a ciphertext C_3.
- Homomorphic multiplication:
  $$EvalMult(Params, ek, C_1, C_2) \rightarrow C_3$$
  EvalMult is an algorithm that takes as input the system parameters *Params*, the evaluation key ek, two ciphertexts C_1 and C_2, and outputs a ciphertext C_3.

The evaluation key is needed to perform homomorphic operations over the ciphertexts. The evaluation key is used in in the following homomorphic operations: relinearization (multiplication) and rotation. Any entity that has only the evaluation key cannot learn anything about the messages from the ciphertexts only [16].

An example of homomorphic encryption with asymmetric key cryptography by using BGV [3], BVF [4], and CKKS [5] schemes is shown in Figure 3.

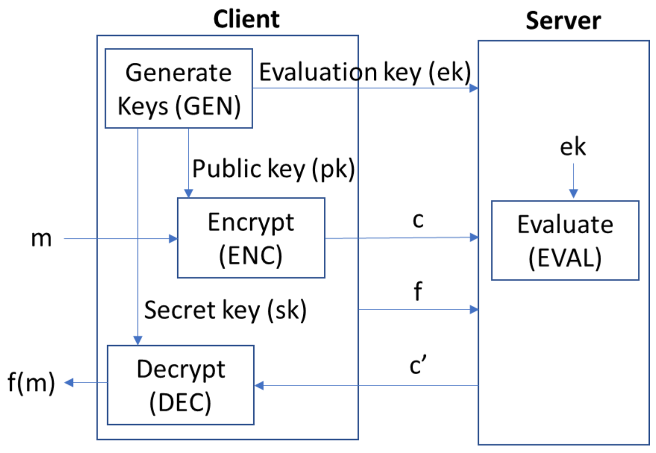

Fig. 3. Homomorphic encryption with asymmetric keys

V. Experimental analysis

In the experimental analysis we evaluated the time needed for execution of the following homomorphic operations: Public key encryption (Table II), Decryption (Table III), Homomorphic addition (Figure 4), and Homomorphic multiplication (Figure 5). Homomorphic encryption libraries implement the above-mentioned cryptographic operations of a scheme and expose a higher-level API. We evaluated the use of the following homomorphic schemes:

- BGV,
- BVF and
- CKKS;

that are implemented in the following open-source libraries respectively:

- Microsoft SEAL [17];
- Palisade [14];
- HELib [18] [19].

HELib is a C++ open source library that implements both the BGV [3] and CKKS [5] homomorphic encryption schemes. HELib library, published in 2013 by Halevi and Shoup, was the first homomorphic encryption library.

Palisade [14] is multi-threaded library written in C++ 11. It uses the NTL library [20] to accelerate underlying mathematical operations. Palisade supports more schemes, including BFV, BGV, CKKS. It also supports multi-party extensions of certain schemes and other cryptographic primitives like Proxy Re-Encryption (PRE) and digital signatures [6].

Microsoft Simple Encrypted Arithmetic Library (SEAL) [17] is a homomorphic encryption library that allows additions and multiplications to be performed on encrypted integers or real numbers. Microsoft SEAL is written in C++11 and contains a .NET wrapper library for the public API. The latest available version 3.6.2 is developed in C++17.

Table I gives an overview of the publicly available open-source libraries with implemented HE algorithms. Palisade implements Boolean circuits Fully Homomorphic Encryption (FHE) schemes: FHEW and TFHE. In the FHE mechanisms it uses bootstrapping procedure [14] (noise refreshing procedure) with the application of the appropriate bootstrapping keys. The FHEW and TFHE schemes are not implemented in the HELib and Microsoft SEAL libraries.

TABLE I
HE ALGORITHMS IN OPEN-SOURCE LIBRARIES

Library/ HE scheme	Palisade	HELib	SEAL
BGV	√	√	
BFV	√		√
CKKS	√	√	√
FHEW	√		
Threshold FHE	√		

The homomorphic encryption code was executed on a PC with:

- 2194.84 MHz 8-core CPU;
- 16 GB RAM;
- Ubuntu 20.04 LTS.

Tables II and III and Figures 4 and 5 show the results of encryption, decryption, HE addition and HE multiplication tests respectively, where:

- Times in the last three columns (HE Library) are expressed in microsecond (μs);
- Each operation was executed 1000 times and the times presented are the times to execute 1000 iterations;
- We used 128-bit homomorphic encryption security level;
- Ciphertext dimension is *n*;
- Ciphertext modulus is *q*.

Ciphertext dimension *n* shall be chosen on basis of desired security level and value of ciphertext modulus *q*. If ciphertext modulus *q* is bigger than noise budget it enables implementation more complex homomorphic evaluation function *f* i.e. implementation the function with bigger depth.

Palisade library implements modular arithmetic schemes: BGV and BVF with 128-bit security level beginning from ciphertext dimension $n = 2048$.

The public key encryption operation in BFV scheme has the best performance when the SEAL library is used. Performance difference depends on the ciphertext dimension: while the SEAL encryption is three times faster for the ciphertext dimension of 2048, when the ciphertext dimension is 32768, this factor is 1.3 times. The encryption operation has the best performance in BGV scheme when the Palisade library is used. Performance difference ratio decreases with the increase of the ciphertext dimension. The encryption operation in CKKS scheme for ciphertext dimension $n \geq 8192$ has the best performance when the HELib library is used, whereas in case of lower dimension *n* the best results are achieved by using SEAL library.

TABLE II
PUBLIC KEY ENCRYPTION

HE scheme	HE parameters		HE library		
	n	$\log_2 q$	Palisade	HELib	SEAL
BFV	1,024	27	-	-	272
BGV	1,024	27	-	1,783	-
CKKS	1,024	27	585	482	257
BFV	2,048	54	1,557	-	506
BGV	2,048	54	1,560	3,608	-
CKKS	2,048	54	1,173	997	479
BFV	4,096	109	3,519	-	1,687
BGV	4,096	109	3,493	7,833	-
CKKS	4,096	109	2,753	2,288	1,926
BFV	8,192	218	7,773	-	4,838
BGV	8,192	218	8,116	17,817	-
CKKS	8,192	218	7,538	4,664	5,688
BFV	16,384	438	24,050	-	16,252
BGV	16,384	438	25,926	44,796	-
CKKS	16,384	438	23,183	12,581	19,344
BFV	32,768	881	77,553	-	59,457
BGV	32,768	881	78,639	109,340	-
CKKS	32,768	881	76,406	39,890	71,373

TABLE III
SECRET KEY DECRYPTION

HE scheme	HE parameters		HE library		
	n	$\log_2 q$	Palisade	HELib	SEAL
BFV	1,024	27	-	-	63
BGV	1,024	27	-	13,047	-
CKKS	1,024	27	415	3,159	10
BFV	2,048	54	159	-	127
BGV	2,048	54	133	49,096	-
CKKS	2,048	54	809	5,104	19
BFV	4,096	109	420	-	416
BGV	4,096	109	353	192,351	-
CKKS	4,096	109	1,432	14,279	72
BFV	8,192	218	940	-	1,484
BGV	8,192	218	1,012	763,178	-
CKKS	8,192	218	6,038	48,960	290
BFV	16,384	438	2,370	-	5,904
BGV	16,384	438	3,690	3,033,690	-
CKKS	16,384	438	13,776	183,254	1,166
BFV	32,768	881	7,330	-	24,919
BGV	32,768	881	14,941	12,003,497	-
CKKS	32,768	881	51,960	701,913	4,826

The decryption operation in CKKS scheme has the best performance by using SEAL library. The decryption operation in CKKS scheme when using SEAL is approximately 10 times faster than when Palisade is used and more than 100 times faster than when HELib is used.

The secret key decryption operation in BGV scheme performs better by several orders of magnitude in the Palisade than in the HELib library.

The decryption operation in BFV scheme for ciphertext dimension $n \geq 8192$ has better performance when Palisade library is used, whereas in case of lower dimension n better results are achieved by using SEAL library.

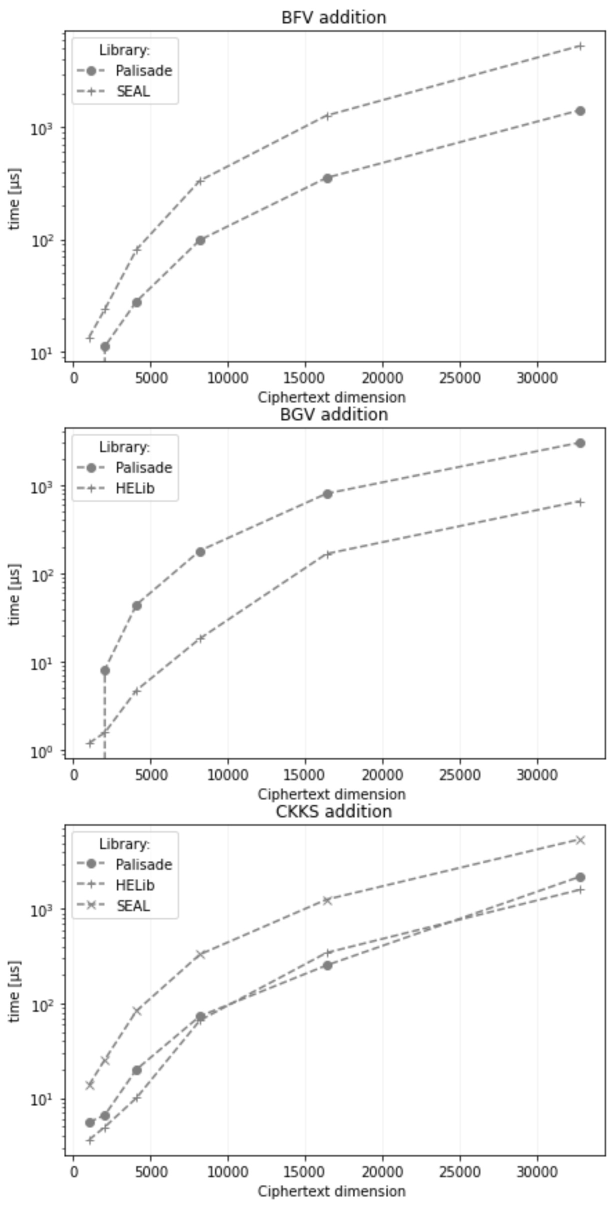

Fig. 4. Homomorphic encryption – addition operation time

The ciphertext addition in CKKS scheme has the best performance in the HELib library. The ciphertext addition in CKKS scheme has better performance in the Palisade than in the SEAL library, but the differences are generally smaller than for the decryption operation.

The ciphertext addition in BFV scheme has significantly better performance (more than 2 times faster) in the Palisade than in the SEAL library.

The ciphertext addition in BGV scheme has significantly better performance (more than 4 times faster) in the Palisade than in the SEAL library.

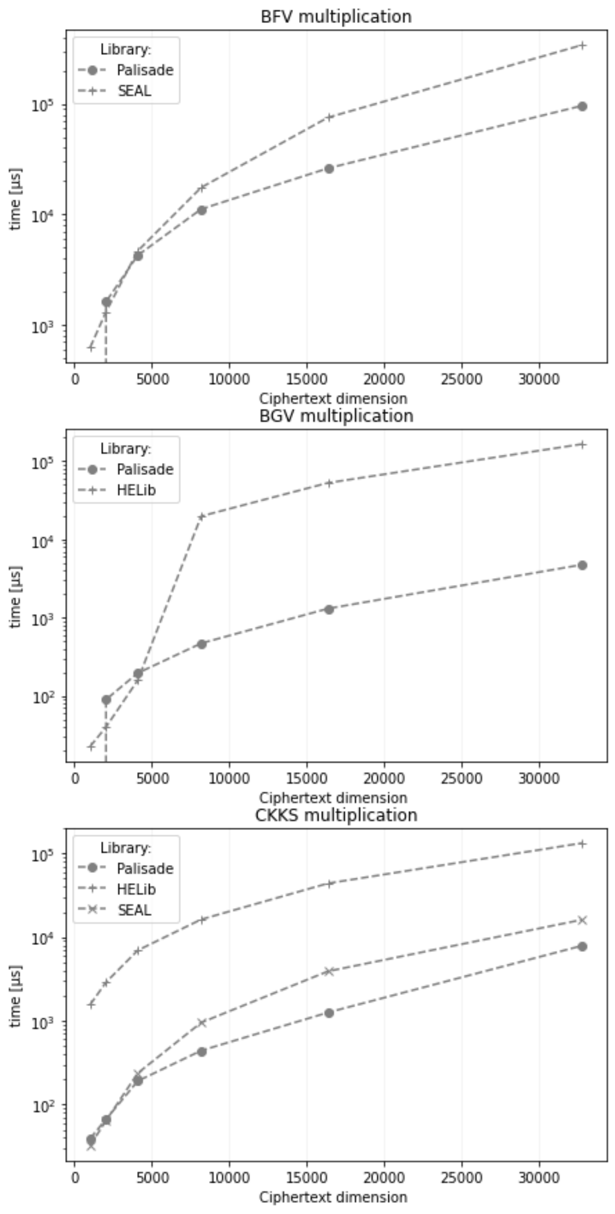

Fig. 5. Homomorphic encryption – multiplication operation time

The ciphertext multiplication is much more complex and more time consuming than ciphertext addition. Figure 4 presents the time needed for performing homomorphic multiplication without relinearization procedure.

The cyphertext multiplication in CKKS scheme for ciphertext dimension $n \geq 8192$ has the best performance when implemented in the Palisade library whereas in case of lower dimension n the better results are achieved using SEAL library.

The cyphertext multiplication in BGV scheme for ciphertext dimension $n \geq 8192$ has significantly better performance (more than 3 times faster) when implemented in the Palisade library than in the HELib whereas for lower ciphertext dimensions better results are achieved by using HELib library.

The cyphertext multiplication in BFV scheme for ciphertext dimension $n \geq 4096$ has better performance in the Palisade than in the SEAL whereas for lower ciphertext dimensions slightly better results are achieved by using SEAL library.

In addition, we compared execution time of homomorphic operations with no security level versus operations with 128-bit security level. We have measured execution time of homomorphic operations in CKKS (approximate arithmetic) and BGV (integer modulo arithmetic) schemes that are implemented in the Palisade library.

In the experiments we have got similar ratio of results for both schemes, so we present only results related to CKKS scheme.

In the tests we have performed homomorphic operations by using following scenarios:

1. No security level with ciphertext dimension n=512;
2. 128-bit security level with ciphertext dimension n=32768.

Each operation was executed 1000 times. In both scenarios it is used same value of ciphertext modulus q.

We have got following results of homomorphic operations (CKKS scheme):

- Public key encryption operation is about 69 times faster in scenario 1;
- Private key decryption operation is about 46 times faster in scenario 1;
- Homomorphic addition operation is about 45 times faster in scenario 1;
- Homomorphic multiplication operation is about 48 times faster in scenario 1.

VI. Conclusions

Homomorphic encryption allows performing computations on the encrypted data, without decrypting them. The paper compares the time needed to execute homomorphic operations, like, public key encryption, secret key decryption, addition and multiplication implemented in the open-source libraries: Microsoft SEAL, Palisade, and HELib. The operations are compared for BGV, BFV and CKKS homomorphic encryption schemes implemented in the libraries.

Homomorphic operations that are performed at client side: public key encryption and secret key decryption if it is used BGV scheme (integer arithmetic) have the best performance when using methods that are implemented Palisade.

Homomorphic operations that are performed at the server side: addition and multiplication are fastest when Palisade library is used for all three tested schemes, except for BGV addition and higher ciphertext dimensions in which cases HELib has slightly better performance.

Execution time of homomorphic operations with no security level versus operations with 128-bit security level was performed and showed that all the operations are still by two orders of magnitude slower than when no security is used which presents an issue when complex machine learning or AI calculations are required.

The performance of current fully homomorphic encryption schemes, especially for large parameters, can still be improved. Further improvement can be achieved by implementation low-level homomorphic operations in an assembly language which is executed on a hardware platform. Also it can be achieved better performance if homomorphic operations are implemented in hardware platforms like Graphics Processing Unit (GPU), Application-Specific Integrated Circuit (ASIC), and Field-Programmable Gate Array (FPGA).

Literature

[1] A. Acar, H. Aksu, A. Selcuk, and M. Conti, "A Survey on Homomorphic Encryption Schemes: Theory and Implementation," ACM Comput. Surv. 1, 1, Article 1, http://dx.doi.org/10.1145/3214303, 2018.

[2] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," Journal of the ACM (JACM) 60, no. 6, 2013.

[3] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully Homomorphic Encryption without Bootstrapping," Cryptology ePrint Archive, Report 2011/277. https://eprint.iacr.org/2011/277, 2011.

[4] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," IACR Cryptology ePrint Archive, 2012:144, 2012.

[5] J.H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," Cryptology ePrint Archive, Report 2016/421, https://eprint.iacr.org/2016/421, 2016.

[6] A. Viand, P. Jattke, A. Hithnawi, "SoK: Fully Homomorphic Encryption Compilers", IEEE Symposium on Security and Privacy 2021.

[7] C. Aguilar Melchor, M. Kilijian, C. Lefebvre, T. Ricosset, "A Comparison of the Homomorphic Encryption Libraries HElib, SEAL and FV-NFLlib," in: Lanet JL., Toma C. (eds) Innovative Security Solutions for Information Technology and Communications. SECITC 2018. Lecture Notes in Computer Science, vol 11359. Springer, Cham. https://doi.org/10.1007/978-3-030-12942-2_32, 2019.

[8] T. Lepoint, M. Naehrig, "A Comparison of the Homomorphic Encryption Schemes FV and YASHE," in: Pointcheval D., Vergnaud D. (eds) Progress in Cryptology – AFRICACRYPT 2014. AFRICACRYPT 2014. Lecture Notes in Computer Science, vol 8469. Springer, Cham. https://doi.org/10.1007/978-3-319-06734-6_20, 2014.

[9] T. Maha, S. Hajji, and A. Ghazi, "Homomorphic encryption applied to the cloud computing security," in Proceedings of the World Congress Engineering, vol. 1, pp. 4-6, 2012.

[10] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in E. Oswald and M. Fischlin, editors, Advances in Cryptology – EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, So_a, Bulgaria, April 26-30, 2015, Proceedings, Part I, volume 9056 of Lecture Notes in Computer Science, pages 617-640. Springer, 2015.

[11] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster packed homomorphic operations and e_cient circuit bootstrapping for tfhe," in Advances in Cryptology-ASIACRYPT 2017: 23rd International Conference on the Theory and Application of Cryptology and Information Security, pages 377-408. Springer, 2017.

[12] O. Regev, "The learning with errors problem," in Blavatnik School of Computer Science, Tel Aviv University Invited survey in CCC, 2010.

[13] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full rns variant of approximate homomorphic encryption," Cryptology ePrint Archive,Report 2018/931, https://eprint.iacr.org/2018/931, 2018.

[14] Y. Polyakov, K. Rohloff, G.W. Ryan, and D. Cousins, "PALISADE Lattice Cryptography Library User Manual (v1.10.6)", 2020.

[15] S. Sathya, P. Vepakomma, R. Raskar, R. Ramachandra, and S. Bhattacharya, "A Review of Homomorphic Encryption Libraries for Secure Computation," http://arxiv.org/abs/1812.024, 2018.

[16] M. Albrecht, M. Chase, H. Chen and others, "Homomorphic encryption standardization," homomorphicencryption.org, 2018.

[17] K. Laine, "Simple Encrypted Arithmetic Library 2.3.1," 2017.

[18] S. Halevi and V. Shoup, "Algorithms in Helib," in Advances in Cryptology – CRYPTO 2014, J. A. Garay and R. Gennaro, Eds, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 554–571, 2014.

[19] S.Halevi, V. Shoup, "HElib design principles," 2020.

[20] V. Shoup and others, "NTL: A library for doing number theory," http://www.shoup.net/ntl.