Comparison of FHE Schemes and Libraries for Efficient Cryptographic Processing

Arisa Tsuji
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku, Tokyo, 112-8610, Japan
arisa-t@ogl.is.ocha.ac.jp

Masato Oguchi
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku, Tokyo, 112-8610, Japan
oguchi@is.ocha.ac.jp

***Abstract*—In recent years, opportunities to collect and analyze big data in the cloud have increased. Cryptographic processing is imperative for protecting data on cloud servers, and homomorphic encryption, which can perform calculations in an encrypted state, is highly expected for this purpose. Fully Homomorphic Encryption (FHE) is an encryption scheme that allows for addition and multiplication operations any number of times in an encrypted state. In the pursuit of practical applications of FHE, multiple encryption schemes have been proposed, and several libraries are available for executing these schemes.**

In this study, we first conducted a comparison to help select the appropriate FHE encryption scheme and library based on the execution environment and processing requirements of the application. Specifically, we organize the time-space complexity and compatible operations for BFV, BGV, CKKS, and Zama's variant of TFHE schemes implemented in OpenFHE, Lattigo, and TFHEpp libraries. For achieving 128-bit security, it was found that BGV, BFV, and CKKS, in that order, are the fastest. Additionally, the memory usage varied depending on the library, with OpenFHE requiring less memory than Lattigo. It is worth noting the differences in the encryption processes of BFV, BGV and CKKS schemes compared with Zama's variant of TFHE scheme. In the comparison between CKKS and Zama's variant of TFHE, Zama's variant of TFHE was more compatible with multiplications between arbitrary values, but CKKS was more compatible with vector inner products.

A common challenge for all FHE encryption schemes is their immense time-space complexity. Therefore, as a second consideration, we compared the execution times and Solid State Drive (SSD) bandwidths between OpenFHE and TFHEpp for Zama's variant of TFHE in environments with limited DRAM, such as in the cloud. It was found that TFHEpp is faster when DRAM is limited. This is because the gate key generation time in OpenFHE significantly increases owing to a lack of memory required for arithmetic processing.

***Index Terms*—(Torus) Fully Homomorphic Encryption, SSD**

I. Introduction

In recent years, businesses and individual users have increasingly had the opportunity to operate large-scale systems to collect and analyze big data using cloud services. When traditional encryption techniques are employed to operate in the cloud, data must be decrypted during computation, which poses a high risk of personal information leakage and unauthorized use by cloud attackers. Fully Homomorphic Encryption (FHE) [1] has attracted attention because it enables the arbitrary computation of encrypted data.

In FHE, multiple encryption schemes such as BFV [2], BGV [3], CKKS [4], and Zama's variant of TFHE [5] have been proposed, each with different internal encryption processes. In addition, several libraries are available to execute these encryption schemes, each with unique implementation details. While it is necessary to select an appropriate encryption scheme and library depending on the execution environment and operations to be performed, it is not easy for people other than cryptography researchers to set appropriate parameters within FHE libraries and compare libraries or schemes.

In this paper, we first compare the features of multiple cryptographic libraries and schemes when they are actually run and aim to provide a reference for practitioners building systems using FHE. Specifically, we organize the time-space complexities during multiplication and proper operations for BFV, BGV, CKKS, and Zama's variant of TFHE schemes implemented in OpenFHE [6], Lattigo [7], and TFHEpp [8] libraries. From our experiments, we found that when processing integers cryptographically, BGV in Lattigo was the fastest. In CKKS, which is capable of cryptographic processing of floating points, Lattigo was faster when there were fewer consecutive multiplication counts, which is called the multiplicative depth, whereas OpenFHE was faster when the multiplicative depth was large. The memory usage depends on the library, and we discovered that OpenFHE requires less memory than Lattigo for all encryption schemes. It is important to note that BFV, BGV, CKKS and Zama's variant of TFHE schemes have different fundamental units of computation and bootstrapping mechanisms; therefore, the choice of the appropriate scheme depends on the type of operations performed in FHE. In the comparison between CKKS and Zama's variant of TFHE schemes, it was observed that Zama's variant of TFHE has a lower time-space complexity when performing multiple multiplications between arbitrary values, but CKKS is more efficient in vector inner products.

However, a common challenge for all FHE encryption schemes when considering practical applications is the immense time-space complexity. Particularly in cloud environments, where resources are shared among multiple VMs or containers, the efficiency of DRAM usage is low, necessitating the utilization of storage [9]. Therefore, as a second aspect, we focus on Zama's variant of TFHE scheme and compare the execution time, SSD bandwidth, and implementation efficiency of OpenFHE and TFHEpp when the DRAM is limited. The purpose is to select an appropriate library based on the amount

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

of DRAM available. Experiments showed that TFHEpp was faster than OpenFHE because memory limitations cause a lack of memory required for calculation processing in OpenFHE gate key creation.

The contributions of this study are as follows:

・Measuring the time-space complexity of FHE encryption schemes (BFV, BGV, CKKS, Zama's variant of TFHE) and libraries (OpenFHE, Lattigo, TFHEpp) and make a comparison to select the appropriate one according to the execution environment and the operations using FHE.

・Comparing the Zama's variant of TFHE implementations between OpenFHE and TFHEpp under DRAM constraints, focusing on execution times, storage bandwidth, and implementation efficiency. This evaluation aims to assist in choosing an appropriate library based on the amount of DRAM available.

II. Background

A. Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is an encryption scheme that allows for addition and multiplication operations any number of times in an encrypted state. FHE always satisfies (1) and (2).

Fully Homomorphic Encryption

$$Encrypt(m) \oplus Encrypt(n) = Encrypt(m+n) ... (1)$$
$$Encrypt(m) \otimes Encrypt(n) = Encrypt(m \times n) ... (2)$$

Figure 1 shows the processing flow of FHE. The flow of FHE process is as follows:

- **Encoding**: Convert a message m you want to encrypt into a plaintext $\mathcal{M}$. In BFV, BGV, and CKKS schemes (see 2.B), the plaintext is a circular polynomial of degree $N-1$, expressed as $R_t = Z_t[x]/(x^N+1)$. Multiple messages ms can be packed into one plaintext $\mathcal{M}$. In Zama's variant of TFHE scheme (see 2.B), the plaintext $\mathcal{M}$ is a torus, and one message m is converted to one plaintext $\mathcal{M}$.
- **Key generation**: Generate a private key and public key to convert the plaintext $\mathcal{M}$ to a ciphertext $\mathcal{C}$.
- **Encryption**: Generate the ciphertext $\mathcal{C}$ from the plaintext $\mathcal{M}$. In BFV, BGV, and CKKS, the ciphertext $\mathcal{C}$ is expressed by the polynomial $R_q = Z_q[x]/(x^N+1)$. The plaintext Rt is mapped to the ciphertext Rq using noise, scaling parameters, and public key. In Zama's variant of TFHE, a ciphertext is expressed by an $(n+1)$-order vector in which each element is a torus.
- **Operation**: Perform homomorphic addition and homomorphic multiplication.
- **Modulus switching or rescale**: Because it is encrypted by adding noise to the message m based on the (R)LWE problem, noise accumulates within the ciphertext $\mathcal{C}$ during calculations. It is reduced by converting the ciphertext modulus from q to q' ($q' < q$). The number of executions must be determined before starting FHE processing. This operation is called modulus switching in BFV and BGV and rescale in CKKS.

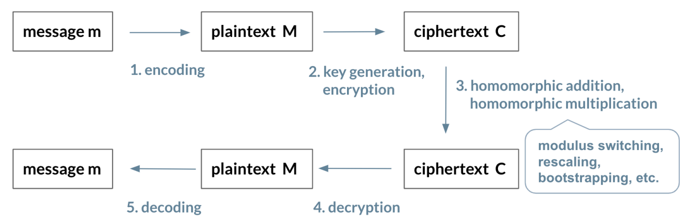


Fig. 1. Processing flow of FHE.

- **Bootstrapping**: Reduce the noise accumulated in the ciphertext $\mathcal{C}$. Noise can be reduced by performing bootstrapping on ciphertext with noise below a certain level. By performing bootstrapping at an appropriate timing, it is possible to multiply continuously any number of times.
- **Decryption**: Decrypt the ciphertext $\mathcal{C}$ using the private key and output a plaintext Rt'.
- **Decoding**: Convert the plaintext Rt' to a message m' using the inverse procedure of encoding.

B. FHE encryption schemes and libraries

Multiple encryption schemes exist for FHE, including BFV, BGV, CKKS, TFHE [10], and Zama's variant of TFHE, and the encryption processing differs for each. In addition, multiple libraries that can execute each encryption scheme have been released.

1) Overview of FHE encryption schemes: Table 1 lists the features of the FHE encryption schemes BFV, BGV, CKKS, TFHE, and Zama's variant of TFHE. BFV, BGV, and CKKS share the same basic cryptographic processing, such as the theory of bootstrapping. Bootstrapping in these schemes reduces noise by decrypting the ciphertext once in a secure state and then encrypting the decryption result again. These schemes execute ciphertexts using arithmetic operations. BFV and BGV can evaluate an integer using the RLWE problem [11] and store N messages in one plaintext. By constrast, CKKS can evaluate a floating point using a Fourier transform and store $N/2$ messages in one plaintext.

TFHE differs from BFV, BGV, and CKKS in basic cryptographic processing. In TFHE, each bit constituting the ciphertext is evaluated using a logic circuit. When performing arithmetic operations, such as addition and multiplication on ciphertext, it is necessary to combine the logic gates. Bootstrapping is executed at each logic gate constituting the circuit. For bootstrapping, it uses a test vector called LUT, whose coefficients are ciphertexts with a certain amount of noise reduced. It is possible to evaluate binary values.

Zama's variant of TFHE is an extension of the TFHE scheme, with an implementation called programmable bootstrapping that allows bootstrapping of not only logical operations but also univariate functions. In addition, it is possible to evaluate a floating point by implementing an encoder. One of the future challenges is to reduce drift errors [5] during calculations. In this study, we compared BFV, BGV, CKKS, and Zama's variant of TFHE schemes.

TABLE I
FEATURES OF FHE SCHEMES

FHE scheme	feature			
	arithmetic operation	logical operation	domains	packing size
BFV	✓		integer	N
BGV	✓		integer	N
CKKS	✓		floating-point	$N/2$
TFHE		✓	binary	1
Zama's variant	✓	✓	floating-point	1

TABLE II
LIBRARIES IN WHICH THE FHE ENCRYPTION SCHEMES ARE IMPLEMENTED

FHE scheme	library		
	OpenFHE	Lattigo	TFHEpp
BFV	✓	✓	
BGV	✓	✓	
CKKS	✓	✓	
TFHE	✓		✓
Zama's variant	✓		✓

2) Overview of FHE libraries: Table 2 lists the libraries in which the FHE encryption schemes are implemented. OpenFHE is an open source library and is used inside the transpiler [12] announced by Google in 2021. TFHE and Zama's variant of TFHE schemes are implemented using libraries such as OpenFHE or TFHEpp. TFHEpp is also an open-source C++ library and is approximately 10% faster than the original TFHE implementation.

C. Storage

In a cloud environment, one resource is shared using VMs and containers; therefore, the DRAM usage efficiency is low, and it is necessary to utilize storage. An SSD, which has been actively researched [13], is a type of storage which features fast read/write speeds and low power consumption. Figure 2 shows the flow of control signals and data among the CPUs, DRAMs, and SSDs that configure systems. The CPU reads, decodes, and executes instructions and outputs the resulting data. The instructions or data to be executed are loaded/stored using the DRAM via internal buses. The data in SSDs are used by the CPU via a DRAM page cache or buffer cache. If the DRAM capacity is insufficient, the cache is released, and the frequency of accessing the SSDs increases. In addition, swap out, which copies the contents of unused memory to the SSD, and swap in, which transfers the contents of the SSD to memory, are performed. This acts as if there is more memory, which is called virtual memory, than the actual DRAM capacity, which is called physical memory.

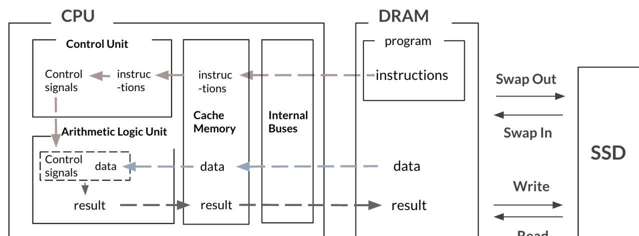


Fig. 2. Control signals and data flow in a system using CPUs, DRAMs, and SSDs.

III. RELATED WORK

[14] classifies multiple FHE schemes and [15] compares the operations that can be performed for each FHE library. However, these studies did not include a runtime evaluation. This study compares and considers the time-space complexity of multiple FHE libraries and schemes when they are actually run. [16] created a benchmark called Terminator 2 Benchmark Suite to compare the computational domains and applications that can be processed at high speed between multiple libraries. However, different schemes are used to compare libraries. In this study, we separately discussed the effects of the implementation of the library and scheme used. [17] uses the Microsoft SEAL library to compare the execution times of BFV, BGV, and CKKS for each cryptographic process. However, high-speed schemes differ depending on the library implementation. In this study, we compared the schemes using multiple libraries. [18] evaluates the cryptographic theory of SHE [19], LHE [20], and FHE and the execution time of some schemes that are not publicly available. In this study, we compared the execution times of the latest open-source libraries. [21] performs a practical evaluation of multiple schemes and libraries. In this evaluation, they compared the execution time between libraries when varying the degree N of the ciphertext polynomial in each scheme; however, in this study, we compared the time-space complexity when varying the multiplicative depth.

IV. EVALUATION OF FHE SCHEMES AND LIBRARIES

We compared BFV, BGV, CKKS, and Zama's variant of TFHE schemes implemented in OpenFHE, Lattigo, and TFHEpp. The evaluation was conducted in the following three steps:

- BFV, BGV, and CKKS that perform evaluation using arithmetic operations and can pack messages into a plaintext. We will compare the time-space complexities of BFV, BGV, and CKKS implemented in Lattigo and OpenFHE, respectively, while varying the multiplicative depth.
- Zama's variant of TFHE that uses logical operations to evaluate each bit that composes a ciphertext. We will compare the time-space complexities of OpenFHE and TFHEpp.
- CKKS and Zama's variant of TFHE. We will use OpenFHE for CKKS and TFHEpp for Zama's variant of TFHE to examine the proper operations of these schemes.

A. Evaluation environment

Table 3 lists the parameters used to compare BFV, BGV, and CKKS schemes. Table 4 lists the parameters used to evaluate

TABLE III
PARAMETERS USED TO COMPARE BFV, BGV, AND CKKS SCHEMES

(a) Parameters

scheme	t	N	batch
BFV	65537	32768	-
BGV	65537	32768	-
CKKS	65537	65536	32768

(b) Ciphertext modulus q

scheme	library	mult depth				
		2	4	6	8	10
BFV	Lattigo	118	177	235	294	352
	OpenFHE	118	177	236	295	354
BGV	Lattigo	118	235	352	410	583
	OpenFHE	118	236	354	472	590
CKKS	Lattigo	146	236	326	416	506
	OpenFHE	145	235	325	415	505

TABLE IV
PARAMETERS USED TO EVALUATE ZAMA'S VARIANT OF TFHE SCHEME

library	N	n	q	gadgetBase	baseSK
OpenFHE	2048	512	2^{27}	2^7	2^7
TFHEpp	2048	500	2^{32}	2^6	2^4

Zama's variant of TFHE scheme. In Zama's variant of TFHE, the ciphertext is represented in two forms: a vector based on the LWE problem [22] and a polynomial ring based on the RLWE problem. gadgetBase is a parameter used to reduce noise in an operation called decomposition for multiplication between ciphertexts based on RLWE. baseSK is used in an operation called key switching to convert ciphertext based on the RLWE problem into ciphertext based on LWE. Table 5 lists the parameters used in the evaluation of CKKS when comparing the computational strengths of CKKS and Zama's variant of TFHE in Equation (1) and Equation (2) (see 4.B).

TABLE V
PARAMETERS USED FOR THE EVALUATION OF CKKS SCHEME IN EQUATION (1) AND EQUATION (2)

n	t	N	batch	q	mult depth
Equation (1)					
2	65537	8192	4096	183	2
20	65537	65536	32768	955	20
40	65537	65536	32768	1475	26
60	65537	65536	32768	1640	29
80	65537	65536	32768	1750	31
100	65537	65536	32768	1805	32
Equation (2)					
2	65537	4096	2	78	1
20	65537	4096	32	78	1
40	65537	4096	64	78	1
60	65537	4096	64	78	1
80	65537	4096	128	78	1
100	65537	4096	128	78	1

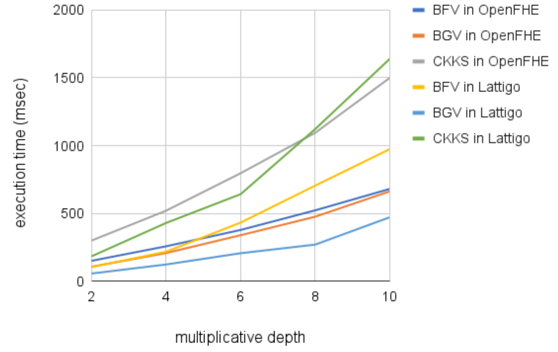


Fig. 3. Comparison of the execution time of BFV, BGV, and CKKS schemes using openFHE and Lattigo libraries.

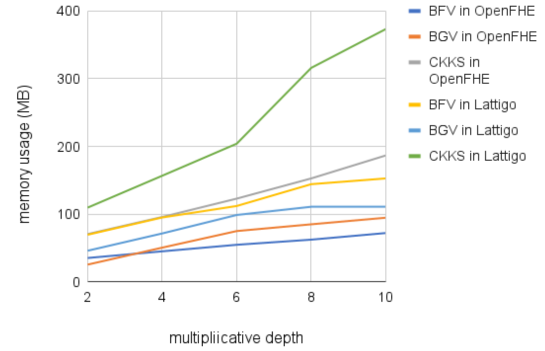


Fig. 4. Comparison of memory usage of BFV, BGV, and CKKS schemes using OpenFHE and Lattigo libraries.

For each evaluation, we used the fastest parameter that satisfied 128-bit security. In addition, the CPU used was an AMD Ryzen 7 5700G@3.80GHz, with eight cores and 16 logical CPUs. The DRAM capacity was 16GB, and the storage used was an SK hynix PC711 SSD with a capacity of 238GB.

B. Experimental result

1) Comparison of time-space complexities of BFV, BGV and CKKS: Figure 3 shows the execution time when changing the multiplicative depth of BFV, BGV, and CKKS with OpenFHE and Lattigo. Comparing the schemes, BGV, BFV, and CKKS were faster at all multiplicative depths, in that order. Among all schemes and libraries, Lattigo's BGV is the fastest in terms of all multiplicative depths. For BFV and CKKS, Lattigo was faster when the multiplicative depth was small, and OpenFHE was faster when it was large. Figure 4 shows the memory usage when changing the multiplicative depth of BFV, BGV, and CKKS with OpenFHE and Lattigo. CKKS tended to use more memory than BFV and BGV. OpenFHE can be executed using less memory for all schemes and number of multiplications.

For all schemes and libraries, the execution time and memory usage increased linearly with the multiplicative depth or

TABLE VI
COMPARISON OF THE EXECUTION TIME OF ZAMA'S VARIANT BETWEEN OPENFHE AND TFHEPP

library	generating sk (ms)	generating gk (ms)	encryption (ms)	bootstrapping (ms)	decryption (ms)
OpenFHE	0.801	3652.380	0.040	463.729	0.007
TFHEpp	0.041	6442.310	0.018	26.865	0.003

degree N of the ciphertext polynomial changes.

2) Time-space complexity of Zama's variant of TFHE: Table 6 compares the execution time of Zama's variant between OpenFHE and TFHEpp. It takes time to generate a gate key and bootstrapping. For gate key creation, OpenFHE was 1.764 times faster than TFHEpp. Conversely, in bootstrapping, TFHEpp was 17.261 times faster than OpenFHE. Therefore, if bootstrapping is executed several times, TFHEpp is faster. For both libraries, memory usage increases linearly during the gate key creation and reaches around 3.3GB upon completion. In the following bootstrapping, memory usage remains stable at approximately 3.3GB. A detailed evaluation is presented in Section 5.

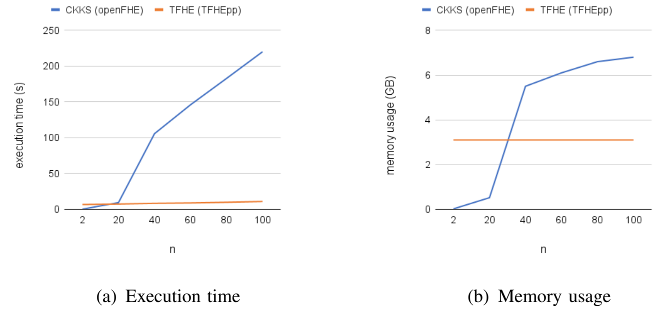
3) Comparison of time and space complexity of CKKS and Zama's variant of TFHE: In this experiment, we compared the time-space complexity when using CKKS in OpenFHE and Zama's variant of TFHE in TFHEpp for multiplications between arbitrary values in (1) and the vector inner product in (2). For, $x \in \mathbb{R}$, $\boldsymbol{a} = (a_0, a_1, ..., a_{n-1})$, and $\boldsymbol{b} = (b_0, b_1, ..., b_{n-1})$, define (1) and (2) as

$$f(x) = \prod_{i=0}^{n-1} x_i \tag{1}$$

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=0}^{n-1} a_i b_i \tag{2}$$

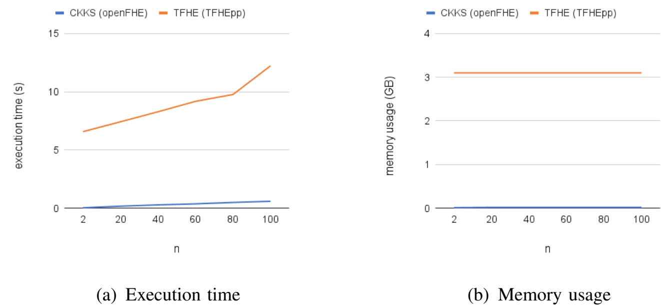
Evaluation of Equation (1). When executing (1), CKKS first performs an operation called rotation on a ciphertext polynomial packed with n messages and creates a new ciphertext by shifting the plaintext vectors in the ciphertext one by one. They are then multiplied to complete the multiplication of n elements in the ciphertext. Because multiplication is performed continuously, bootstrapping must be performed every time the multiplicative depth is consumed. In contrast, Zama's variant of TFHE creates n ciphertexts corresponding to each message and multiplies them. It executes a programmable bootstrapping after one multiplication.

Figure 5 shows the time-space complexity associated with changes in the number of data n when executing Equation (1) using CKKS and Zama's variant of TFHE. When the number of data n is less than 20, CKKS has a low time-space complexity, and when it is 40 or more, Zama's variant of TFHE has a low time-space complexity. In CKKS, it is necessary to rotate and multiply the packed ciphertext n times. As the number of multiplications increased, the number of bootstrappings required also increased. When the number of data n is 0, 20, 40, 60, 80, or 100, bootstrapping is required 0, 2, 3, 4, or 5 times, respectively. Because the amount of time-space complexity required for one bootstrapping is large, the efficiency decreases significantly compared to Zama's variant of TFHE as the number of required bootstrappings increases. In Zama's variant of TFHE, multiplication and bootstrapping are required for each number of data n, and the amount of time-space complexity increases in proportion to the number of data n. By contrast, because the time-space complexity required for multiplication and bootstrapping is small, there is no significant increase.



(a) Execution time (b) Memory usage

Fig. 5. Comparison of the execution status of Equation (1) between CKKS and Zama's variant of TFHE.

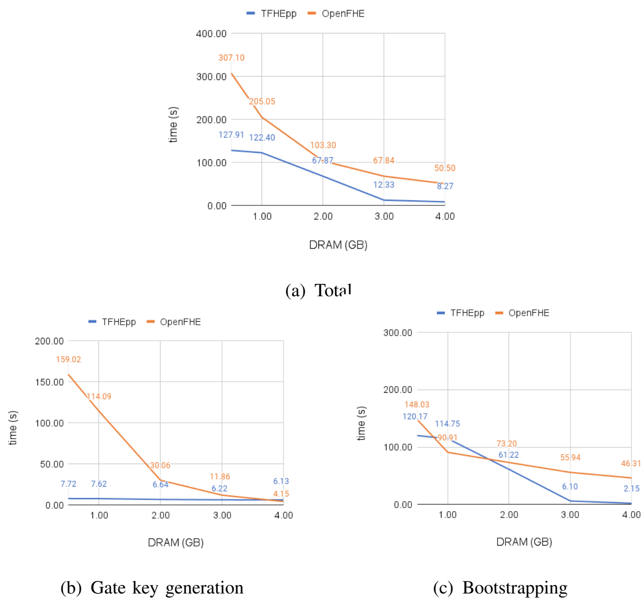


(a) Execution time (b) Memory usage

Fig. 6. Comparison of the execution status of Equation (2) between CKKS and Zama's variant of TFHE.

Evaluation of Equation (2). When executing (2), CKKS multiplies two ciphertexts, each storing n messages. Next, addition is performed on the elements in the vector of the multiplication result (i.e., ciphertext). By contrast, in Zama's variant of TFHE, it is necessary to perform n-time multiplication and bootstrapping for each set of vector elements. After n multiplications, the n operation results are added.

Figure 6 shows the time-space complexity associated with changes in the number of data n when executing Equation (2) using CKKS and Zama's variant of TFHE. For all n, CKKS requires less time-space complexity than Zama's variant of TFHE. In CKKS, the multiplication result of n elements can be obtained by only one multiplication between two ciphertexts, and bootstrapping is not required. In Zama's variant of TFHE, it is necessary to perform multiplication and bootstrapping n times each in the same way as in (1).



(a) Total

(b) Gate key generation

(c) Bootstrapping

Fig. 7. Comparison of the execution time when the DRAM is limited in Zama's variant of TFHE with TFHEpp and OpenFHE.

TABLE VII
COMPARISON OF SSD BANDWIDTH CHANGES WHEN THE DRAM IS LIMITED IN ZAMA'S VARIANT OF TFHE BETWEEN TFHEPP AND OPENFHE

scheme	library	memory constraint (GB)				
		0.5	**1**	**2**	**3**	**unlimited**
gate key generation						
OpenFHE	read	41.902	47.347	64.349	31.816	0.166
	write	39.723	45.412	99.876	41.086	0.179
TFHEpp	read	0.635	0.103	0.138	0.053	0.034
	write	317.097	254.194	160.888	0.179	0.104
bootstrapping						
OpenFHE	read	31.358	19.946	17.116	7.985	0.005
	write	2.811	10.077	14.553	8.008	0.006
TFHEpp	read	33.020	28.433	30.074	38.378	0.724
	write	6.318	10.995	33.147	44.998	0.077

V. EVALUATION OF ZAMA'S VARIANT OF TFHE SCHEME WHEN DRAM IS LIMITED

FHE has a large time-space complexity; however, in practice, it is often used on the cloud and executed without sufficient DRAM allocation. Therefore, in this section, we compare the execution time, SSD bandwidth, and implementation efficiency of OpenFHE and TFHEpp, which can implement Zama's variant of TFHE scheme when the DRAM is limited. In this study, we evaluated a series of cryptographic processing (key generation, encryption, bootstrapping, and decryption) for 100 messages. We performed an experiment to scale up the actual execution environment where FHE runs in the cloud, showing the results when the DRAM was set to 0.5GB, 1GB, 2GB, 3GB, and no limit.

A. Evaluation environment

The CPU used was an AMD Ryzen 7 5700G @ 3.8 GHz and had 16 logical cores. In our experiments, we use Docker containers to limit DRAM. We used SK hynix PC711 with a maximum of 238GB available for swap destination SSD. Table 4 lists the parameters used in OpenFHE and TFHEpp.

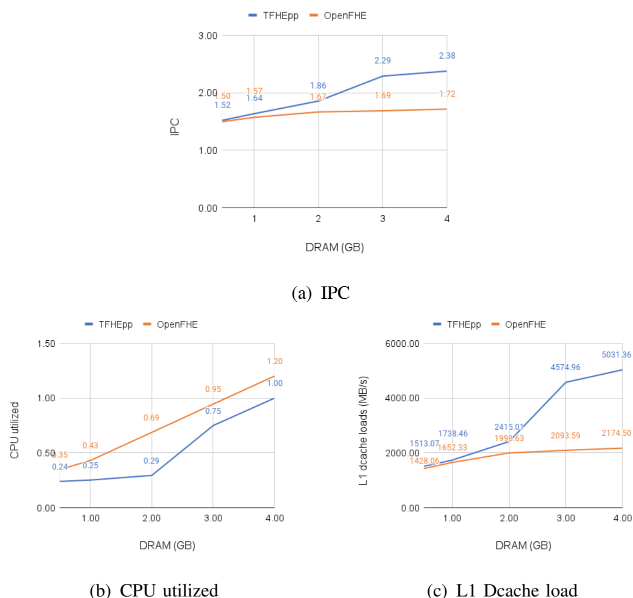
B. Experimental result

Execution time and SSD bandwidth. Figure 7 shows the changes in the total execution time, gate key generation time, and bootstrapping execution time when the DRAM is limited in Zama's variant of TFHE with TFHEpp and OpenFHE. OpenFHE generates the gate key faster if memory is not limited; however, OpenFHE significantly increases the gate key generation time owing to memory limitations. Table 7 shows a comparison of SSD bandwidth changes when the DRAM is limited in Zama's variant of TFHE between TFHEpp and OpenFHE. When generating a gate key with OpenFHE, there is no increase in the bandwidth owing to DRAM limitations. This is because there is a tendency not to access a large amount of data during the operation, and the DRAM limit does not increase the frequency of read/write to the SSD. However, with DRAM being limited, there is not enough memory to perform arithmetic processing, which increases the execution time. When generating a gate key in TFHEpp, SSD write bandwidth increases significantly owing to the DRAM limitations. TFHEpp accesses a large amount of data during the operation, and it is thought the memory used for data access will be insufficient owing to DRAM limitations.

Regarding bootstrapping, TFHEpp is faster when there is no memory limit, but TFHEpp is more susceptible to memory limits than OpenFHE, and the execution time increases rapidly below 3GB. There is no increase in bandwidth for either encryption scheme, and it can be seen that the memory required to perform arithmetic processing in bootstrapping is insufficient due to DRAM limitations. In terms of the total execution time, TFHEpp was faster at all DRAM limit values because of the increase in OpenFHE's gate key creation time.

Implementation efficiency. Figure 8 shows a comparison of changes in implementation efficiency when the DRAM is limited in Zama's variant of TFHE between TFHEpp and OpenFHE. If there is no DRAM limit, TFHEpp has a higher IPC and better computational efficiency. However, owing to the DRAM limitation, the IPC of TFHEpp is significantly reduced, reaching to 1.5, which is comparable to OpenFHE with 0.5GB of DRAM. In addition, the maximum value of the CPU utilized is 16; however, this was a low value for both libraries, indicating that the parallelism of processing was extremely low. By default, TFHEpp performs the entire process in a single thread, and OpenFHE uses 16 threads for key generation and one thread for bootstrapping.

The number of L1D cache loads was 2.5 times higher in TFHEpp. Owing to the DRAM limit, TFHEpp's L1D cache load number had decreased to 1513 (MB/sec), which was approximately the same as OpenFHE. It can be seen that the L1D cache is not utilized owing to the deterioration in the



(a) IPC

(b) CPU utilized

(c) L1 Dcache load

Fig. 8. Comparison of changes in implementation efficiency when the DRAM is limited in Zama's variant of TFHE between TFHEpp and OpenFHE.

computing performance of the CPU.

VI. Conclusion

In this study, we first compared the time-space complexities of multiple cryptographic libraries (OpenFHE, Lattigo, TFHEpp) and schemes (BFV, BGV, CKKS, Zama's variant of TFHE). When cryptographically multiplying integers, BGV in Lattigo is the fastest among BFV, BGV, and CKKS. For CKKS which can evaluate floating points, Lattigo is faster when the multiplicative depth is small, and OpenFHE is faster when there are many. Among all the encryption schemes, OpenFHE can be executed with less memory than Lattigo.

In addition, CKKS (,BFV and BGV) and Zama's variant of TFHE schemes have different basic calculation units and bootstrapping mechanisms; therefore, the operations at which they are good are different. When multiplying arbitrary values 40 or more times, Zama's variant of TFHE requires less time-space complexity. However, when calculating the inner product of vectors, CKKS requires less time-space complexity.

Second, we compared the execution time, SSD bandwidth, and implementation efficiency when limiting DRAM with multiple values between OpenFHE and TFHEpp, which implement Zama's variant of TFHE. TFHEpp is faster than OpenFHE when memory is limited. This is because the gate key ganeration time in OpenFHE increases significantly, owing to the lack of memory required for arithmetic processing.

References

[1] Craig Gentry. 2009. A FULLY HOMOMORPHIC ENCRYPTION SCHEME. Stanford University. https://crypto.stanford.edu/craig/craig-thesis.pdf

[2] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144. http://eprint.iacr.org/2012/144.pdf

[3] Zvika Brakerski, Craig Gentry and Vinod Vaikuntanathan. 2012. (leveled) fully homomorphic encryption without bootstrapping. In Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS ' 12), 309–325. Association for Computing Machinery. https://doi.org/10.1145/2090236.2090262

[4] Jung Hee Cheon, Andrey Kim, Miran Kim and Yong-soo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt ' 17), 409–437.Cryptology ePrint Archive, Report 2017/421. https://eprint.iacr.org/2016/421.pdf

[5] Ilaria Chillotti, Marc Joye and Pascal Paillier. 2021. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. Cyber Security, Cryptology, and Machine Learning (CSCML ' 21), 1–19. Cryptology ePrint Archive, Report 2021/091. https://eprint.iacr.org/2021/091.pdf

[6] OpenFHE, https://github.com/openfheorg/openfhe-development

[7] Lattigo, https://github.com/tuneinsight/lattigo

[8] TFHEpp, https://github.com/kenmaro3/TFHEpp

[9] Muhammad Tirmazi et al. Borg: the Next Generation. 2020. Fifteenth European Conference on Computer Systems (EuroSys ' 20), 1–4. Association for Computing Machinery. https://doi.org/10.1145/3342195.3387517

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: fast fully homomorphic encryption over the torus. Journal of Cryptology, Volume 33, Number 1, 34–91. Cryptology ePrint Archive, Report 2018/421. https://eprint.iacr.org/2018/421.pdf

[11] Zvika Brakerski, Craig Gentry and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS ' 12), 309–325. Association for Computing Machinery. https://doi.org/10.1145/2090236.2090262

[12] google, https://github.com/google/fully-homomorphic-encryption

[13] Jaeyoung Do, Sudipta Sengupta and Steven Swanson. 2019. Programmable solid state storage in future cloud datacenters. Communications of the ACM, Volume 62, Issue 6, 54–62. Association for Computing Machinery. https://doi.org/10.1145/3286588

[14] Sain Sri Sathya, Praneeth Vepakomma, Ramesh Raskar, Ranjan Ramachandra and Santanu Bhattacharya. 2018. A Review of Homomorphic Encryption Libraries for Secure Computation. https://arxiv.org/pdf/1812.02428.pdf

[15] Ijcsis Editor and Ahmed El-Yahyaoui. 2016. Fully Homomorphic Encryption: State of Art and Comparison. International Journal of Computer Science and Information Security (IJCSIS ' 16), Volume 14, Number 4.

[16] Charles Gouert, Dimitris Mouris and Nektarios Georgios Tsoutsos. 2022. SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks. Proceedings on Privacy Enhancing Technologies (PoPETs ' 23), Issue 3, 154–172. Cryptology ePrint Archive, Report 2022/425. https://eprint.iacr.org/2022/425.pdf

[17] Shereen Mohamed Fawaz, Nahla Belal, Adel Elrefaey and Mohamed Waleed Fakhr. 2021. A Comparative Study of Homomorphic Encryption Schemes Using Microsoft SEAL. Journal of Physics Conference Series, Volume 2128, 012021. https://iopscience.iop.org/article/10.1088/1742-6596/2128/1/012021/pdf

[18] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac and Mauro Conti. 2018. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. ACM Computing Surveys, Volume 51, Issue 4, Number 79, 1–35. Association for Computing Machinery. https://doi.org/10.1145/3214303

[19] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac and Mauro Conti. 2018. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. ACM Computing Surveys, Volume 51, Issue 4, Number 79, 1–35. Association for Computing Machinery. https://doi.org/10.1145/3214303

[20] Frederik Armknecht et al. 2015. A Guide to Fully Homomorphic Encryption. International Association for Cryptologic Research (IACR), Report 1192, 1–35. Cryptology ePrint Archive, Report 2015/1192. https://eprint.iacr.org/2015/1192.pdf

[21] Thi Van Thao Doan, Mohamed-Lamine Messai, Gérald Gavin and Jérôme Darmont. 2023. A survey on implementations of homomorphic encryption schemes . The Journal of Supercomputing, Volume 79, 15098–15139. https://doi.org/10.1007/s11227-023-05233-z

[22] Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM),Volume 56, Issue 6, 1–40. Association for Computing Machinery. https://doi.org/10.1145/1568318.1568324