


A Comparative Study of BFV and CKKs Schemes to Secure IoT Data Using TenSeal and Pyfhel Homomorphic Encryption Libraries

Yancho B. Wiryen, University of Douala, Cameroon*

Noumsi Woguia Auguste Vigny, University of Douala, Cameroon

Mvogo Ngono Joseph, University of Douala, Cameroon

Fono Louis Aimé, University of Douala, Cameroon

 <https://orcid.org/0000-0002-7315-0427>

ABSTRACT

Internet of things (IoT) devices and applications are on the rise, generating large amounts of sensitive and confidential data that need to be processed securely. Due to resource constraints, the data generated is often stored and processed in the cloud. The drawback of data cloud storage and processing is the fact that it can be hacked, leaked, or sold by cloud companies. Fully homomorphic encryption (FHE) allows computation on encrypted data using basic mathematical operations and has recently been successfully implemented using schemes and libraries with better performance. In this paper, the authors propose a mixture of edge-cloud-based security schemes using FHE to secure IoT data. The authors evaluate the performance of two FHE schemes (BFV and CKKS) based on data: encoding speed, encryption speed, arithmetic operations (addition and multiplication) speed, and decryption decoding speed using two Python libraries (TenSEAL and PyFHEl). The encryption and decryption are done at the edge node using a Raspberry Pi 4, while the processing is done at the cloud node using a laptop.

KEYWORDS

BFV, CKKs, Edge Node, Fully Homomorphic Encryption (FHE), Internet of Things (IoT), Performance, Pyfhel, TenSeal,

1. INTRODUCTION

In our world today, we have billions of connected devices, sensors, actuators, controllers, and applications that are communicating and interacting to form the Internet of Things (IoT). These IoTs help to improve our health, the quality of life in our homes, save time, and make our workplace more

DOI: 10.4018/IJSST.333852

*Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

productive, thereby improving our welfare. By 2030, our world will be revolutionized by the IoT market (Griffiths & Ooi, 2018), and as projected by Gartner, IoT data will arguably become the biggest big data source, possibly overtaking enterprise, media, and entertainment data (Akbar, 2017). Despite the numerous advantages of IoT, they are unable to handle and compute the large amount of data they generate due to resource constraints and because the cost of implementing some computation of massive data on them might outweigh the benefits (Evans & Eysers, 2012). A combination of IoT, machine learning, and cloud computing technology has often been used as a solution to this large amount of data, and even more so due to the prevalence of the COVID-19 pandemic, as everyone is now soliciting for cloud services (Iezzi, 2020).

A security mechanism capable of preserving this data needs to be put in place to ensure that IoT data is not accessed by cloud companies or third parties or does not end up in the wrong hands. This mechanism is difficult to achieve with traditional encryption schemes (Song et al., 2018). For traditional encryption schemes, each time a computation needs to be performed on the encrypted data stored in the cloud, the data is first decrypted. After the decrypted data is processed, it will be re-encrypted and re-uploaded to the cloud. This process often gives the cloud service providers and the model owners' access to the data and is very tedious and time-consuming (Maha et al., 2012). For the users, they want cloud service providers to process the data and extract the valuable information contained while keeping it unknown to other users and third-party services. In other words, there is the desire to manipulate data while ensuring data protection, privacy, and anonymization to ensure that IoT data does not get into the wrong hands..

Homomorphic encryption is capable of handling this challenge and enables computation on encrypted data without decryption. In 2009, there was a remarkable breakthrough when Gentry (Gentry, 2009) successfully demonstrated that fully homomorphic encryption (FHE) was possible, even though it had difficulties in implementation and was time-consuming. FHE refers to a specific class of encryption scheme that allows computing directly (a large number of different types of mathematical operations) on encrypted data without having to decrypt it first. The result of the ciphertext when decrypted is the same as the output of the mathematical operations on the corresponding plaintext.

Several FHE schemes and libraries have been published that allow even those who are not good at cryptography to apply FHE in various domains ranging from data science (Iezzi, 2020), healthcare (Wood et al., 2020), IoT (Song et al., 2018), (Alabdulatif et al., 2019), (Butpheng et al., 2020), (Ramesh & Govindarasu, 2020), and banking (Ren et al., 2021) to enhance data security and privacy. We will evaluate the performance of the two most successful FHE schemes: Brakerski/Fan-Vercauteren (BFV) (Fan & Vercauteren, 2012) and Cheon, Kim, Kim, and Son (CKKS) (Cheon et al., 2017), used in TenSEAL (Benaissa et al., 2021) and PyFHEI (Ibarrondo & Viand, 2021) python base libraries that we have considered in this paper. Their main feature is the use of the residue number system (RNS) for performing operations (Babenko et al., 2020). This is done by determining the execution time of the main functions (encoding, encryption, addition/multiplication operations, decryption, and decoding) in the scheme, thereby determining the most productive scheme.

The majority of the FHE-based IoT data privacy and security models that are currently in use are based on the cloud. There is a need to extend these schemes and models to incorporate edge computing because there is always a possibility for data to be a compromise between the IoT device and the cloud (Ma et al., 2020). Secondly, the overall performance of the BFV and CKKs schemes is affected by certain parameters, which tend to determine and influence the degree of required security, speed and number of mathematical operations done in each scheme (Fawaz et al., 2021). Limited information exists on quantitative comparison as concerns the variation of these parameters according to the various schemes and libraries. This makes different FHE schemes have distinctive advantages (Jiang & Ju, 2022). It is therefore necessary to implement and modify these parameters to determine which schema performs well in a particular scenario.

The remaining sections are structured as follows: Section 2 provides an overview of related works on Fully Homomorphic Encryption (FHE) and introduces two libraries and schemes utilized in this

study. Section 3 delves into the methods, results, and analysis of the schemes and libraries' privacy performance. Finally, Section 4 presents concluding observations.

2. LITERATURE REVIEW AND DESCRIPTION OF USEFUL CONCEPTS

2.1 Literature Review

The original concept behind the cryptography technique was to ensure secure communication between multiple parties. In this process, one party encrypts a message and sends it to another party, who then decrypts it (Wood et al., 2020). Among the various methods for data security and preservation, Craig Gentry's groundbreaking work (Gentry, 2009) demonstrated the feasibility of Fully Homomorphic Encryption (FHE). However, implementing FHE posed challenges due to its enormous computational time. FHE refers to a specific class of encryption schemes that enable computation directly on encrypted data without the need for prior decryption. Each time the ciphertext is decrypted, the result should match the output of the mathematical operations performed on the corresponding plaintext. Numerous research works focus on preserving data security and privacy using Homomorphic Encryption schemes. Below, we outline some of the related work we reviewed regarding the enhancement of data security using HE.

Hossein et al. (2017) proposed Pilatus, a data protection platform that stores only encrypted data in the cloud. It supports certain queries like range and sum and was implemented in two mobile applications (Fitbit and Ava). However, its efficiency was low as it relied on Partially Homomorphic Encryption (PHE). Wei-Tao Song et al. (2018) improved the bootstrapping technique of Halevi and Shoup by introducing SIMD homomorphic computation techniques, thereby enhancing the efficiency of re-encryption. Goiuri et al. (2019) demonstrated the enhancement of data privacy and security in the cloud by combining Network Coding (NC) with HE technology. Practical implementation issues arose due to platform differences. Abdulatif et al. (2021) introduced the Edge of Things (EoT) paradigm, a secure smart healthcare surveillance framework utilizing FHE for data privacy. The proposed framework was evaluated using patient bio-signal data. However, the Brakerski-Gentry-Vaikuntanathan (BGV) framework used in this system, limited to computation over integers, presented implementation challenges. Ahmed et al. (2019) introduced an efficient and symmetric verifiable FHE scheme based on a noise-free mathematical structure, employing simple matrix operations for homomorphic computations. The implementation utilized the JAVA programming language and Microsoft Azure as a cloud computing environment without utilizing edge computing for encryption. Jiasen Liu et al. (2021) employed the TenSEAL library and the CKKS scheme to encrypt user data, implementing a secure KNN classification scheme (CKKSKNNC) in cloud servers for Cyberspace. Shereen et al. (2021) conducted a comparative study of the BFV and CKKS FHE schemes using the Microsoft SEAL library in C++ to preserve data. Encryption and multiplication processes were performed in the cloud, without edge encryption. The scheme utilized simple matrix operations for homomorphic computations, with the encryption process taking only a few milliseconds in a cloud environment.

These research works have demonstrated the ability of Fully Homomorphic Encryption (FHE) to maintain the privacy of sensitive data throughout the computation process. Various solutions have been suggested to enhance the security of IoT data, particularly at the cloud level. Further research is required to practically implement FHE for securing IoT data, as well as to examine the performance of different schemes at the edge when dealing with growing data volumes.

In the next subsection, we will provide a brief description of the Brakerski-Fan-Vercauteren Homomorphic Encryption scheme (BFV scheme) and the Cheon-Kim-Song Homomorphic Encryption Scheme (CKKS scheme), as well as some useful devices.

2.2 Description of the Two Useful Schemes (BFV and CKKS)

A good FHE Schemes should be capable of supporting two main homomorphic operations (Luka & Vuletić, n.d.):

1. Additive Homomorphic Encryption;

$$\text{Enc}(m1 + m2) = \text{Enc}(m1) + \text{Enc}(m2); \forall m1, m2 \in M.$$

2. Multiplicative Homomorphic Encryption.

$$\text{Enc}(m1 * m2) = \text{Enc}(m1) * \text{Enc}(m2); \forall m1, m2 \in M$$

2.2.1 Brakerski-Fan-Vercauteren Scheme (BFV Scheme)

In 2012, Fan and Vercauteren (2012) made modifications to Brakerski's Fully Homomorphic Encryption (FHE) scheme based on Learning With Errors (LWE) to work under the security assumption of RLWE, resulting in the BFV scheme. BFV exhibits powerful Single Instruction Multiple Data (SIMD) parallelism, making it efficient for handling large data. Theoretically, BFV supports bootstrapping, a technique for noise removal by applying a circuit representing the decryption algorithm of an FHE scheme to a ciphertext and an encrypted private key (Sathya et al., 2018). However, the bootstrapping process is slow and rarely used in practice. The BFV scheme operates on integers using modular arithmetic and allows for addition and multiplication operations on encrypted data while preserving the privacy of the underlying plaintext. It is defined over a polynomial ring modulo a prime number, with plaintext represented as an element of this ring and ciphertext as a pair of polynomials. The BFV scheme involves multiple steps, as mentioned below.

- Key Generation:
 - Parameters: Choose appropriate security parameters, such as the modulus “q” and plaintext space “R”.
 - Secret Key (sk) Generation: Select a secret key “s” uniformly at random from the set Z_q and compute the public key (pk) corresponding to “s”.
- Encryption:
 - Plaintext Mapping: Represent the plaintext message “m” as an integer in the plaintext space “R”.
 - Error Generation: Generate a small random error “e” from a distribution centered around 0.
 - Encryption: Compute the ciphertext “c” by encrypting the plaintext “m” with the public key (pk) using the encryption function Enc:

$$c = \text{Enc}(pk, m, e) = (q * m + 2 * e + s) \bmod q.$$

- Homomorphic Operations:
 - Homomorphic Addition: Given two ciphertexts “c1” and “c2” representing encrypted messages “m1” and “m2”, homomorphic addition is performed by adding the ciphertexts together:

$$c_{\text{add}} = c1 + c2 \bmod q.$$

- Homomorphic Multiplication: Given two ciphertexts “c1” and “c2” representing encrypted messages “m1” and “m2”, homomorphic multiplication is performed by multiplying the ciphertexts together modulo q^2 :

$$c_{\text{mul}} = c1 * c2 \bmod q^2.$$

- Decryption:
 - Using the secret key (sk), decrypt the ciphertext “c” to obtain the original plaintext message “m” using the decryption function Dec:

$$m = \text{Dec}(\text{sk}, c) = (c \bmod q) \bmod R.$$

The BFV scheme incorporates techniques such as modulus switching and noise management to ensure the correctness and security of the computations. These techniques enable secure computation on encrypted data while preserving the privacy of the underlying plaintext.

2.2.2 Cheon-Kim-Kim-Song Scheme (CKKS Scheme)

CKKS, proposed by Cheon, Kim, Kim, and Song (2017) in 2017, is an approximate number arithmetic scheme that enables computations on vectors containing both real and complex values. This gives CKKS an advantage over BFV, which only supports computations over integers. During computation, CKKS supports homomorphic rounding-off and treats noise as a part of the numerical error in its approximation. The Cheon-Kim-Kim-Song (CKKS) scheme is a FHE scheme that operates on complex numbers using approximate arithmetic. It is defined over a cyclotomic ring of characteristic zero, where the plaintext is an element of this ring and the ciphertext is a vector of complex numbers. The CKKS scheme involves multiple steps:

- Key Generation:
 - Parameters: Choose appropriate parameters, such as the degree of the polynomial “n”, modulus “q”, and scaling factor “s”.
 - Secret Key (sk) Generation: Randomly generate a secret key “s” and compute the public key (pk) corresponding to “s”.
- Encryption:
 - Plaintext Mapping: Represent the plaintext message “m” as a polynomial with complex coefficients.
 - Error Generation: Generate a small random error polynomial with complex coefficients.
 - Encryption: Compute the ciphertext “c” by encrypting the plaintext polynomial “m” with the public key (pk) using the encryption function Enc:

$$c = \text{Enc}(\text{pk}, m, e) = (m + e) \bmod q.$$

- Homomorphic Operations:
 - Homomorphic Addition: Given two ciphertexts “c1” and “c2” representing encrypted polynomials “m1” and “m2”, homomorphic addition is performed by adding the ciphertexts together:

$$c_{\text{add}} = c1 + c2 \bmod q.$$

- Homomorphic Multiplication: Given two ciphertexts “c1” and “c2” representing encrypted polynomials “m1” and “m2”, homomorphic multiplication is performed by multiplying the ciphertexts together modulo q:

$$c_mul = c1 * c2 \bmod q.$$

- Decryption:
 - Decryption: Using the secret key (sk), decrypt the ciphertext “c” to obtain the original plaintext polynomial “m” using the decryption function Dec:

$$m = \text{Dec}(sk, c) = c \bmod q.$$

The CKKS scheme incorporates rescaling to manage noise accumulation during homomorphic operations, ensuring accurate computations on encrypted data. By multiplying the ciphertext with a power of the scaling factor, it handles this process. CKKS utilizes complex numbers and approximate arithmetic, enhancing efficiency while introducing minor errors in the results.

This paper focuses on the CKKS and BFV homomorphic encryption schemes due to their practicality, extensive research, and real-world applicability. CKKS and BFV are fully homomorphic encryption schemes, allowing arbitrary computations on encrypted data. They offer versatility, with CKKS suitable for real or complex number computations and BFV excelling in computations on encrypted integers. These schemes are widely used, well-documented, and supported by libraries, making them comprehensive and accessible options for various applications.

In the present day, various computing technologies such as Edge, Fog, and Cloud computing have emerged. These technologies can be leveraged in the realm of IoT to tackle certain challenges by offering flexible resources and services to end users at the network's edge. In this Subsection, we conclude by elucidating these three computing technologies within the context of IoT.

The edge encompasses various components such as sensors, controllers, actuators, tag and tag readers, communication elements, gateways, and physical devices. Its primary function is to minimize the required communication bandwidth between sensors and the central data center by conducting analytics and generating insights at or near the data source. Additionally, data encryption can also be implemented at this specific node or level.

Fog computing is viewed as an expansion of the cloud computing model that extends from the core of the network to its edge. It operates as a highly virtualized platform, offering fundamental computation, encryption, storage, and networking services between end devices and conventional cloud servers.

A Cloud refers to a centralized online system that offers services such as enabling immediate, on-demand provision of computing infrastructure, databases, storage, and applications required for processing and analyzing data points generated by numerous IoT devices.

Throughout this paper, we present a security scheme that focuses on preserving privacy by utilizing FHE to safeguard IoT data from unauthorized access by attackers and cloud companies. We will compare different libraries and schemes in order to achieve this objective. In pursuit of this goal, we will proceed to:

1. Encrypt IoT plaintext data at edge node using rasp berry pi 4
2. Send the FHE data to a computer at the level of cloud
3. Perform two arithmetic operations (addition and/or multiplication) on the FHE data.
4. Send back the FHE data to the edge node (rasp berry pi 4) for decryption.

5. Measure, record and compare the time for each round operation for both PyFHEI and TensSEAL library under CKKs and BFV schemes.

3. METHODS, RESULTS AND DISCUSSIONS

3.1 Methods

Python is a flexible and currently one of the world's leading programming language when it comes to data science and machine learning since it enables rapid prototyping. To begin with the implementation of FHE, we choose two schemes and two Python libraries from Table 1 that utilize Python, in contrast to the majority of FHE encryption libraries that are based on C++. The choice of these libraries are guided by the simplicity and readability of their language, making it ideal for prototyping.

TenSEAL is an open-source library that leverages Microsoft SEAL to perform Fully Homomorphic Encryption operations on tensors. While it utilizes a Python API, the majority of its efficient operations are implemented in C++. TenSEAL seamlessly integrates with popular machine learning frameworks, simplifying the process of implementing tensor operations on encrypted data. TenSEAL relies on the implementation of CKKS and BFV schemes as in Microsoft SEAL.

Python for Homomorphic Encryption Libraries (Pyfhe) by Alberto Ibarrondo and Alexander Viand is an easy to use python library for FHE schemes, which also currently includes the Brakerski-FanVercauteren (BFV) scheme, the Cheon-Kim-Kim-Song (CKKS) scheme, and bootstrapping for CKKS. Pyfhe implements functionalities of multiple Homomorphic Encryption libraries such as addition, multiplication, exponentiation or scalar product in Python and Cython on top of C++. This library is useful both for simple Homomorphic Encryption Demos as well as for complex problems such as Machine Learning algorithms.

We propose a combined edge and cloud-based architecture to enhance the security and privacy of data within our network. Figure 1 illustrates the edge architecture, and we have identified the application of Fully Homomorphic Encryption (FHE) on both the edge and cloud as a potential solution for improving the security and privacy of IoT data and applications.

Within this new architecture, we encrypt the data generated by the IoT device at the edge using Raspberry Pi 4 before transmitting it to the cloud. Encrypting the data at the edge is chosen due to its robust preprocessing and encryption capabilities, which prevent privacy breaches or attacks before the data reaches the cloud. Additionally, it helps prevent cloud controllers from accessing critical data.

Thus, our IoT system generates data from customer devices, which is encrypted at the edge level using Raspberry Pi 4 and a public key. The public key can be shared with anyone, while the private key remains exclusively shared with the customer to unlock the data. The encrypted data is then sent to the cloud for processing, which involves operations such as addition and multiplication of

Table 1. FHE libraries

Name	Programming Language	Schemes		
		CKKs	BFV	BGV
TenSEAL	C++, Python	✓	✓	✓
PyFHEI	C++, Python	✓	✓	✓
SEAL	C++, .NET	✓	✓	✓
PALISADE	C, C++	✓	✓	✓
HElib	C, C++	✓		✓
HEAAN	C++	✓		
Lattigo	C++	✓	✓	

encrypted data. The resulting outcomes are also encrypted and transmitted back to the customer at the edge node (Raspberry Pi 4). The customer can then use the private key to unlock and decrypt the results, which are subsequently provided to the user in an unencrypted format, as depicted in Figure 2.

3.2 Results and Discussions

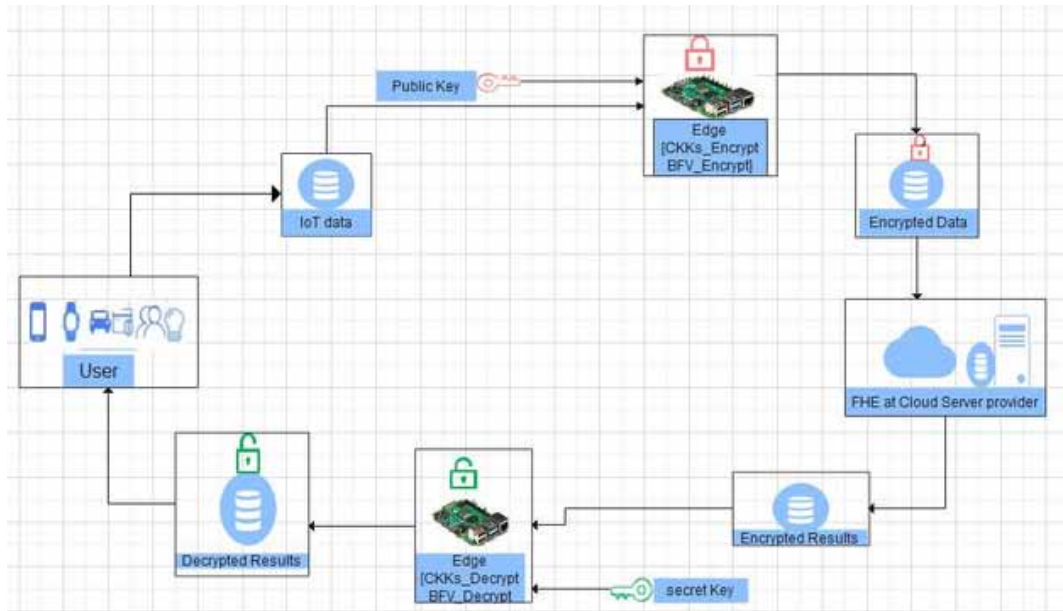
In this Subsection, we will present a comparative evaluation of addition and multiplication operations using the TenSEAL and PyFHEI library. The encryption and decryption was conducted on Raspberry pi 4 at the edge while the mathematical operations took place in a computer (with the characteristics: Windows10 operating system, Intel(R), Core(TM) i5-4310U CPU @, 2.00GHz and 4 GB RAM). Using Anaconda 3-2019.07, Spider 5.0.5 and python IDE to call TenSEAL-0.3.5, PyFHEI-2.3.1 libraries, we implement both CKKS and BFV encryption scheme. By applying each scheme (of each associated library) on two linear homomorphic operations (addition and multiplication), a comparative analysis with respect to the time consumed by each computation is done. We give runtimes for addition and multiplication in BFV and CKKS schemes using TenSEAL and PyFHEI libraries under given parameters as in Table 2,3,4,5. There are three mean HE parameters used by the schemes:

- **n (polynomial degree):** It determines the number of slots of the plaintext vectors (n in BFV and $n/2$ in CKKS).

Figure 1. The edge architecture



Figure 2. System architecture



- **p (plaintext modulus):** It determines the modulus of the plaintext space in BFV, which determines how large encrypted values can get before wrap-around occurs. This module also does not affect security
- **q (plain text size or cyphertext modulus):** It determines how much noise can accumulate before decryption fails. This module has no effect on the degree of security. Instead of providing a ciphertext modulus Q , users working with CKKS must provide a modulus chain of prime sizes example ($q = [40, 40, 40, 40]$) (Fan & Vercauteren, 2012).

The computation process often begins with the KeyGen algorithm to set up the various keys which can be implemented in python using Pyfhel library as shown below:

```
from Pyfhel import PyCtxt, Pyfhel, PyPtxt
HE = Pyfhel ()
HE. contextGen (scheme ='BFV ', n=4096, p= 65537)
HE. keyGen ()
from Pyfhel import PyCtxt, Pyfhel, PyPtxt
HE = Pyfhel ()
HE. contextGen (scheme ='CKKS ', n=4096, q_s=[40, 40, 40, 40 ])
HE. keyGen ()
```

Next, we need to encode and encrypt the message using the encryption algorithm. In a similar manner after decryption, the plaintext will need to be decoded as shown below.

```
import numpy as np
np_array = np. array ([[11,20,30,75,5,16,7,80,60,10]], dtype =np.
int64)
array_ptxt = HE. encode (np_array)
array_ctxt = HE. encrypt (array_ptxt)
array_dec = HE. decrypt (array_ctxt, decode = True)
```

In Table 2, we perform TenSEAL BFV and CKKS vector addition using two vectors recording the time in millisecond (ms) for each of the five operations (Encoding, Encryption, Addition, Decryption and Decoding). When we increase the ciphertext dimension n while keeping constant p and q , we notice an increase in the time for each operation for both schemes. We also observed that;

- The encryption operation takes a great deal of time compared to the addition, encoding, decrypting and decoding operations.
- CKKS encryption takes more time than BFV encryption
- Addition operation is faster for CKKS than BFV but slower for CKK when the value of “ n ” is increased up to $2^{16}(32768)$.

We record the time for TenSEAL BFV and CKKS Multiplication with the same parameters and operations (Encoding, Encryption, Addition, Decryption and Decoding) like those of Table 2. From the results of Table 3 we observe that;

- Multiplication operation consumed a larger amount of time for the same parameters (n , p , q) than the addition operation in Table 2.
- The multiplication operation consumed more time than the encryption or decryption operations.

We perform Pyfhel BFV and CKKS vector addition using two vectors recording the time in millisecond (ms) for each of the five operations (Encoding, Encryption, Addition, Decryption and Decoding). The results in Table 4 were similar to those of Table 2. The following observations were made;

- The encryption procedure took longer compared to the addition operation, which required less time.
- CKKS encryption takes more time than BFV encryption
- For values of $n < 16384$ BFV addition is more performant than CKKs and for values of $n \geq 16384$ BFV addition is less performant than CKKs.

In Table 5 we present the time for the multiplication computation process for Pyfhel library and observed as follows;

- The Pyfhel BFV multiplication and encryption operations took approximately the same amount of time, whereas the Pyfhel CKKS multiplication operations required significantly less time compared to encoding and encryption.
- For values of $n < 16384$ the total time for BFV multiplication is faster than CKKs and for values of $n \geq 16384$ BFV multiplication is slower than CKKs.

Upon comparing the results presented in Tables 2, 3, 4, and 5, we observed variations in the timings of the two Python FHE libraries (TenSEAL and Pyfhel) across different schemes (BFV and CKKS). Figures 3 and 4 provide a comparative analysis of the implementation of these libraries and schemes across various operations. Figure 3 specifically demonstrates the timing disparity between the two libraries and schemes during the addition process. Based on the results, it is evident that the TenSEAL library outperforms the Pyfhel library for values of $n < 32768$, while it lags behind for values of $n \geq 32768$. Furthermore, across all polynomial degrees, the BFV schemes consistently exhibit faster performance compared to the CKKS schemes. When examining Figure 4, it becomes apparent that the total time required for Pyfhel Multiplication exceeds that of TenSEAL multiplication

Table 2. TenSEAL BFV and CKKS addition

HE Parameter		TenSEAL BFV Vector Addition Time(ms)					TenSEAL CKKs Vector Addition Time(ms)							
n(poly-nomial degree)		Encode	Encrypt	Addition	Decrypt	Decode	Total Time	Encode	Encrypt	Addition	Decrypt	Decode	Total Time	
	4096		0.10	15.64	0.00	0.90	0.01	16.65	8.90	31.24	0.00	0.95	9.00	50.09
	8196		0.20	46.85	0.23	3.00	0.20	50.32	18.09	57.71	0.07	1.11	16.01	93.15
	16384		1.20	122.01	1.91	11.10	0.70	136.91	37.80	100.11	1.90	3.01	40.00	181.83
	32768		3.00	374.92	15.61	31.26	3.00	444.79	65.00	100.22	31.00	5.02	60.00	261.24
p(plaintext Modulus)		65537	/											
q(plain text size)		[40,40,40,40]	/											
			[40,40,40,40]											

Table 3. TenSEAL BFV and CKKS multiplication

HE Parameter		TenSEAL BFV Vector Multiplication Time(ms)						TenSEAL CKKs Vector Multiplication Time(ms)					
n(polyn-omial degree)		Encode	Encrypt	Multiplication	DeCrypt	Decode	Total Time	Encode	Encrypt	Multiplication	Decrypt	Decode	Total Time
	4096	0.10	15.64	15.62	0.90	0.01	32.22	8.90	31.24	2.50	0.95	9.00	50.09
	8192	0.20	46.85	31.27	3.00	0.20	81.52	18.09	57.71	5.07	1.11	16.01	93.15
	16384	1.20	122	156.00	11.10	0.70	291.0	37.80	100.11	15.71	10.01	50.00	211.6
	32768	3.00	374.92	734.00	31.26	3.35	1146	65.00	100.22	626.00	5.02	60.00	856.22
p(plain text Modulus)	65537	/						/					
q(plain text size)	[40,40,40,40]	/						[40,40,40,40]					

Table 4. Pyfhel BFV and CKKS addition

HE Parameter		Pyfhel BFV Vector Addition Time(ms)						Pyfhel CKKs Vector Addition Time(ms)						
n(polyn-omial degree)		Encode	Encrypt	Addition	Decrypt	Decode	Total Time	En-code	Encrypt	Addition	Decrypt	Decode	Total Time	
		4096	0.12	16.41	0.00	1.00	0.09	17.26	8.90	31.25	0.00	2.05	9.00	51.2
		8196	0.29	47.00	0.61	3.90	0.20	52.00	18.09	57.71	0.18	3.11	16.01	95.1
		16384	2.78	122.0	4.01	19.19	1.90	149.8	37.80	100.11	0.90	13.01	50.00	201.8
		32768	3.00	421	4.09	31.00	3.03	462	68.01	200.01	2.0	20.20	80.0	368
p(plain text Modulus)		65537												
q(plain text size)		/											[40,40,40,40]	

Table 5. Pyfhel BFV and CKKS multiplication

HE Parameter		Pyfhel BFV Vector Multiplication Time(ms)						Pyfhel CKKS Vector Multiplication Time(ms)							
n(polyn-omial degree)		Encode	Encrypt	Multiplication	Decrypt	Decode	Total Time	Encode	Encrypt	Multiplication	Decrypt	Decode	Total Time		
	4096		0.12	16.41	25.33	1.00	0.09	42.97		8.90	31.25	4.00	2.05	9.00	55.20
	8196		0.29	47.00	45.60	3.90	0.20	96.99		18.09	57.71	10.07	3.11	16.01	105.0
	16364		2.78	122.0	171.20	19.19	1.90	317.0		37.80	100.11	31.25	13.01	50.00	232.1
	32768		3.00	421	562.00	31.0	3.0	1017		68.01	200.22	150.02	30.0	80.00	530.0
p(plain text Modulus)		65537													
q(plain text size)		/													
		[40,40,40,40]													

when $n < 32768$, but the situation is reversed when $n \geq 32768$, regardless of the specific operations performed on the vectors.

Furthermore, upon examining Figure 3 and Figure 4, it becomes evident that in the CKKS scheme, ciphertext addition demonstrates superior performance for polynomial degrees or ciphertext dimensions (n) up to 16384, regardless of whether the TenSeal or Pyfhel libraries are used. On the other hand, for values of $n > 16384$, both the TenSeal and Pyfhel libraries exhibit better performance with the BFV scheme.

In ciphertext addition, TenSeal library is more performant than Pyfhel library for both CKK and BFV schemes for all values of ciphertext dimension n .

Figure 3. Addition time for TenSEAL BFV vs. CKKS

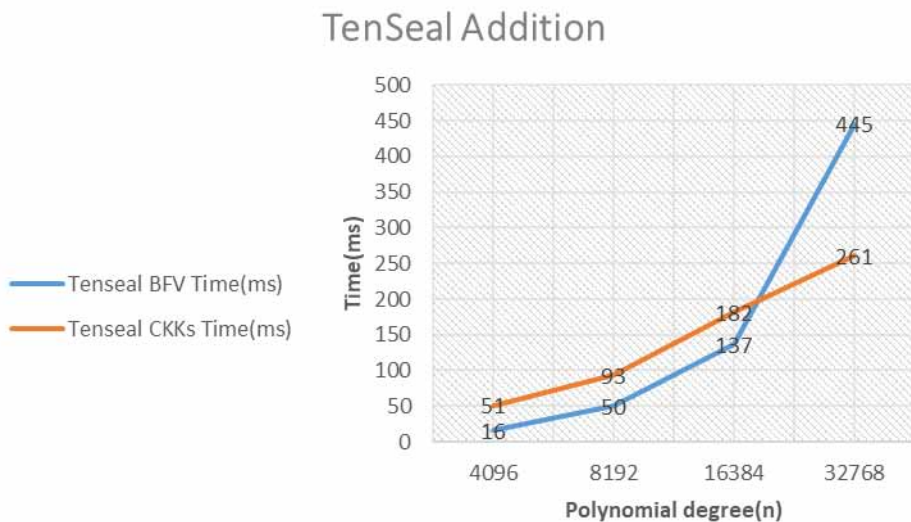
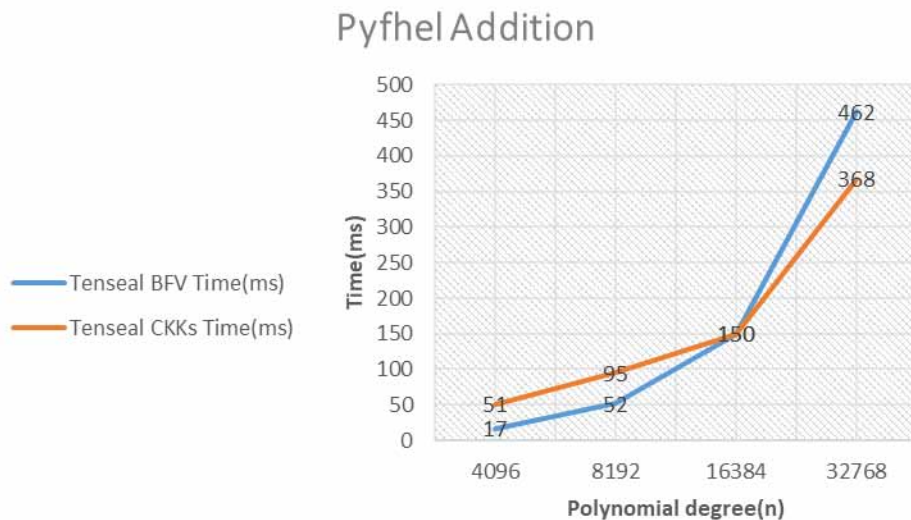


Figure 4. Addition time for Pyfhel BFV vs. CKKS



We observe from Figures 3-6 that ciphertext multiplication is much more complex and more time consuming than ciphertext addition.

Based on the observations made in Figure 5 and Figure 6, it is apparent that the CKKS scheme demonstrates superior performance for ciphertext multiplication when the ciphertext dimension $n \geq 8192$. This holds true for implementations using both the TenSeal and Pyfhel libraries. However, when the dimension is lower, specifically for $n < 8192$, the results indicate slightly better performance using the BFV scheme for both libraries.

For polynomial degrees (n) up to 16384, the TenSeal library demonstrates significantly better performance in ciphertext multiplication. However, when dealing with higher polynomial degrees ($n > 16384$), the Pyfhel library yields better results in terms of performance.

Figure 5. Multiplication time in TenSEAL (BFV and CKKS)

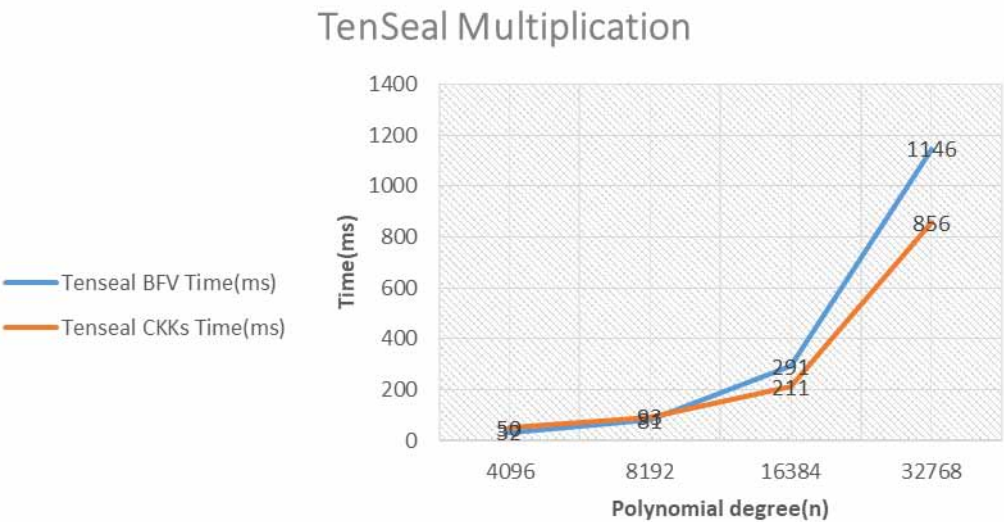
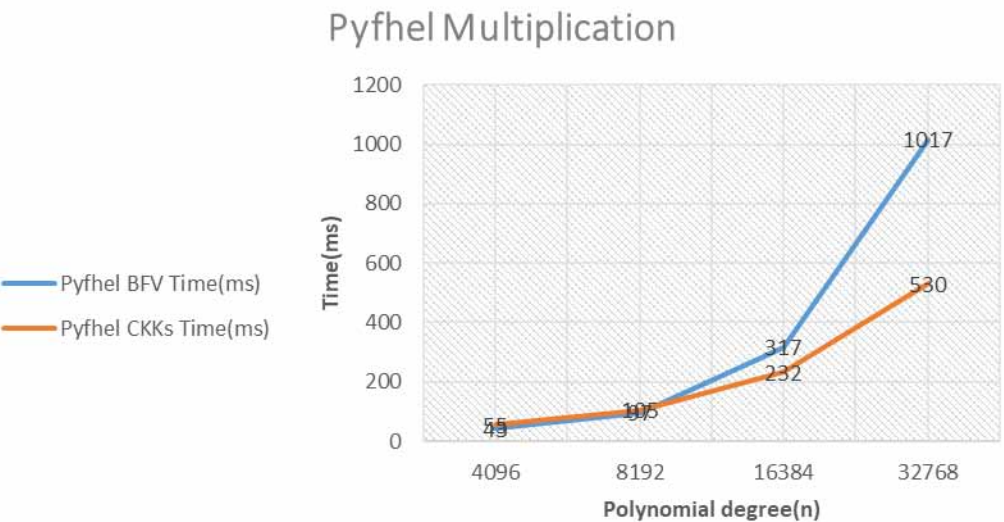


Figure 6. Multiplication time for Pyfhel BFV vs. CKKS



4. CONCLUDING REMARKS

In this paper, we present an edge-cloud base FHE and we analyze its ability to enhance data security and privacy with practical implementation of FHE models using two python libraries TenSEAL and PyFHEI which are based on two schemes Brakerski-Fan-Vercauteren scheme (BFV) and Cheon, Kim, Kim, and Song (CKKS). The proposed process has two phases:

- The Encryption/decryption is done at the edge node (edge computing) using Raspberry pi 4,
- The computation (addition and multiplication) is done in the cloud using laptop.

The results obtained from implementing and testing FHE in real-world IoT data scenarios showcase the feasibility of deploying FHE at the edge node. The computational time for basic operations such as multiplication and addition exhibits significant differences only when dealing with very large vector sizes (polynomial degree of $n \geq 2^{15}$). However, this challenge can be addressed by utilizing GPU hardware that supports FHE computations.

The FHE Python libraries, namely TenSEAL and Pyfhel, along with their BFV and CKKS schemes, offer user-friendly interfaces and modifiability, making FHE accessible to a broader audience. Consequently, even an untrusted party can securely perform computations on IoT encrypted data, yielding results consistent with those obtained from unencrypted data or ciphertexts.

This solution allows us to identify the most performant FHE scheme for different vector sizes of IoT data, considering their propensity to generate substantial amounts of data.

In our future work, we aim to expand our research by comparing more intricate mathematical operations, utilizing larger datasets to test various operations, and exploring additional libraries. Moreover, we plan to implement machine learning algorithms such as Support Vector Machine, regression, and Neural Networks on this IoT encrypted data. We anticipate that the performance can be further enhanced by implementing FHE on Graphics Processing Units (GPUs).

ACKNOWLEDGMENT

This work was done under the research grant FR 21-333 RG/MATHS/AF/ AC_G-FR 3240319514 from Unesco-TWAS and the Swedish International Development Cooperation Agency (SIDA). The views expressed herein do not necessary represent those of UNESCO-TWAS, Sida or its Board of Governors.

REFERENCES

- Akbar, A. (2017). *Extracting Knowledge from Raw IoT Data Streams*. University of Surrey.
- Alabdulatif, A., Khalil, I., Yi, X., & Guizani, M. (2019). Secure edge of things for smart healthcare surveillance framework. *IEEE Access : Practical Innovations, Open Solutions*, 7, 31010–31021. doi:10.1109/ACCESS.2019.2899323
- Babenko, M. G. E., Golimblevskaia, E. I., & Shiriaev, E. M. (2020). Comparative analysis of homomorphic encryption algorithms based on learning with errors. *Труды института системного программирования РАН*, 32(2), 37–51.
- Butpheng, C., Yeh, K. H., & Xiong, H. (2020). Security and privacy in IoT-cloud-based e-health systems-A comprehensive review. *Symmetry*, 12(7), 1191. doi:10.3390/sym12071191
- Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017, December). Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 409–437). Springer. doi:10.1007/978-3-319-70694-8_15
- El-Yahyaoui, A., & Dafir Ech-Cherif El Kettani, M. (2019). A verifiable fully homomorphic encryption scheme for cloud computing security. *Technologies*, 7(1), 21. doi:10.3390/technologies7010021
- Evans, D., & Eysers, D. M. (2012, November). Efficient data tagging for managing privacy in the internet of things. In *2012 IEEE International Conference on Green Computing and Communications* (pp. 244–248). IEEE. doi:10.1109/GreenCom.2012.45
- Fan, J., & Vercauteren, F. (2012). *Somewhat practical fully homomorphic encryption*. Cryptology ePrint Archive.
- Fawaz, S. M., Belal, N., ElRefaey, A., & Fakhr, M. W. (2021, December). A Comparative Study of Homomorphic Encryption Schemes Using Microsoft SEAL. *Journal of Physics: Conference Series*, 2128(1), 012021. doi:10.1088/1742-6596/2128/1/012021
- Gentry, C. (2009). *A fully homomorphic encryption scheme*. Stanford University.
- Griffiths, F., & Ooi, M. (2018). The fourth industrial revolution-Industry 4.0 and IoT. *IEEE Instrumentation & Measurement Magazine*, 21(6), 29–43. doi:10.1109/MIM.2018.8573590
- Ibarrondo, A., & Viand, A. (2021, November). Pyfhel: Python for homomorphic encryption libraries. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (pp. 11–16). doi:10.1145/3474366.3486923
- Iezzi, M. (2020, December). Practical privacy-preserving data science with homomorphic encryption: An overview. In *2020 IEEE International Conference on Big Data (Big Data)* (pp. 3979–3988). IEEE.
- Liu, J., Wang, C., Tu, Z., Wang, X. A., Lin, C., & Li, Z. (2021). Secure KNN Classification Scheme Based on Homomorphic Encryption for Cyberspace. *Security and Communication Networks*, 2021, 2021. doi:10.1155/2021/8759922
- Luka, U. A. B., & Vuletić, P. V. (n.d.). *Performance comparison of homomorphic encryption scheme implementations*. Academic Press.
- Ma, Z., Ma, J., Miao, Y., Liu, X., Choo, K. K. R., Yang, R., & Wang, X. (2020). Lightweight privacy-preserving medical diagnosis in edge computing. *IEEE Transactions on Services Computing*, 1.
- Maha, T. E. B. A. A., Saïd, E., & Abdellatif, E. (2012). Homomorphic encryption applied to the cloud computing security. In *Proceedings of the World Congress on Engineering (Vol. 1, pp. 4–6)*. Academic Press.
- Peralta, G., Garrido, P., Bilbao, J., Agüero, R., & Crespo, P. M. (2019). On the combination of multi-cloud and network coding for cost-efficient storage in industrial applications. *Sensors (Basel)*, 19(7), 1673. doi:10.3390/s19071673 PMID:30965629
- Ramesh, S., & Govindarasu, M. (2020). An efficient framework for privacy-preserving computations on encrypted IoT data. *IEEE Internet of Things Journal*, 7(9), 8700–8708. doi:10.1109/JIOT.2020.2998109

- Ren, W., Tong, X., Du, J., Wang, N., Li, S. C., Min, G., & Bashir, A. K. (2021). Privacy-preserving using homomorphic encryption in Mobile IoT systems. *Computer Communications*, 165, 105–111. doi:10.1016/j.comcom.2020.10.022
- Shafagh, H., Hithnawi, A., Burkhalter, L., Fischli, P., & Duquennoy, S. (2017, November). Secure sharing of partially homomorphic encrypted IoT data. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems* (pp. 1-14). doi:10.1145/3131672.3131697
- Song, W. T., Hu, B., & Zhao, X. F. (2018). Privacy protection of IoT based on fully homomorphic encryption. *Wireless Communications and Mobile Computing*, 2018, 2018. doi:10.1155/2018/5787930
- Ullah, A., Azeem, M., Ashraf, H., Alaboudi, A. A., Humayun, M., & Jhanjhi, N. Z. (2021). Secure healthcare data aggregation and transmission in IoT-A survey. *IEEE Access : Practical Innovations, Open Solutions*, 9, 16849–16865. doi:10.1109/ACCESS.2021.3052850
- Wood, A., Najarian, K., & Kahrobaei, D. (2020). Homomorphic encryption for machine learning in medicine and bioinformatics. *ACM Computing Surveys*, 53(4), 1–35. doi:10.1145/3394658

Yancho B. Wiryen is a PhD Student and Researcher, laboratory of Applied Computer science, University of Douala.

Noumsi Woguia Auguste Vigny Enseignant à l'université de Douala, Cameroun. PhD obtenu à l'Université de Rennes 1 et à l'université de Douala depuis 2010.

Mvogo Ngono Joseph is a research and senior lecturer in the field of signal and image processing at the University of Douala, he holds a PHD in image processing, his work focuses on data coding and compression.

Fono Louis Aimé was born in July 1969 in Douala-Cameroon, former student of the Higher Teacher Training College of the University of Yaounde I-Cameroon and, Ph.D. holder in Applied Mathematics for Social Science of the Laboratory of MASS of the same University, M. Louis Aimé FONO is Associate Professor in the Department of Mathematics and Computer Science at the University of Douala-Cameroon and he chairs the Laboratory of Mathematics of the University. His areas of interest are: Fuzzy Mathematics, Preference Modeling, Social Choice, Financial Mathematics, Mathematics for Life Insurance and Mathematics for Supply Chain Management.