The Fastest k-means Ever Realized by FHE*

*Note: Sub-titles are not captured in Xplore and should not be used

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—K-means clustering is a significant and classic unsupervised machine learning method used widely in data analysis. Fully homomorphic encryption allows one to operate on ciphertext and get the same result as on plaintext after decryption, making it a vary suitable technique to be used in outsourced computation especially in the bigdata era without the concern of privacy leaks and interaction. In this work, we propose a fully privacy-preserving, effective, and efficient k-means clustering scheme based on CKKS. And to avoid the comparison and division, which are unfriendly operations for FHE schemes, we use polynomial to approximate the compare function and use stabilized method in the "updating centers" step to avoid dividing by a ciphertext. Experimental results show that the accuracy of our proposed k-means clustering algorithm on ciphertext is almost equivalent to that of the classical k-means algorithm on plaintext. And to our best knowledge, our work is the first practical and completely outsourced k-means clustering scheme based on FHE, other works either compeletely outsourced but requiring too much time, or requiring practical time but not compeletely outsourced.**

***Index Terms*—k-means, clustering, Fully Homomorphic Encryption, CKKS, privacy-preserving**

[1]

I. Introduction

[2] [3] Nowadays, computation power becomes a commercial good. It is more and more commen that one party who owns data to utilize the computation power of the other party to analysis the data. In this two-party scenario, data privacy is of important concern. There are many solutions proposed to protect data privacy in this outsourced computation scenario, such as differential privacy technique, secure interactive protocol, and fully homomorphic encryption(FHE). The idea of FHE is that doing operations on ciphertext and obtain the same result as on paintext after decryption. This idea has emerged and been persuited long time ago, since the famous RSA public key encryption system been proposed, which supports homomorphic addition. But one scheme can be called fully homomorphic encryption scheme only when it supports addition and mutiplication simultaniously and has no limit on the times of these two operations can be done. It is until 2008 that Gentry proposed a FHE scheme based on lattice. Afterwards, many FHE schemes are proposed and they are less complicated than the first FHE scheme proposed by Gentry.

Clustering is a kind of unsupervised machine learning method to analysis data. It can reveal intrinsic patterns and characteristics hidden in the data and is widely used in many fields. There are already some privacy-preserving k-means clustering algorithm based on FHE been proposed. Jaschke and Armknecht propoesd a such algorithm based on FHE scheme TFHE, which supports logic operations AND, OR. They construct their algorithm curcuit by plenty of logic gates, which takes too much time to run over. Ali et al. proposed a scheme which supports translation between two different FHE schemes CKKS and TFHE, and they use this translation scheme to run one iteration of k-means clustering algorithm. Recently, Mengyu Zhang et al. proposed a scheme based on CKKS, but did not utilize bootstrapping, so their scheme only supports a few times of iterations of k-means clustering.

Our scheme is based on the CKKS scheme and utilizes the bootstrapping function of CKKS to enable limitless times of iterations of k-means clustering. Comparison and ciphertext division are unsupported operations for the CKKS scheme. So in the "compare distances" step of k-means clustering which is meant to allocate points to their nearest centers by distances to centers, we use ploynomial to approximate the comparison function. And in the "update centers" step, to avoid dividing by the ciphertext of the numbers of points allocated to the

Identify applicable funding agency here. If none, delete this.

[1] Chen: The title is a bit exaggerated. It may not be easy to prove that our method is the best. So “Fastest” may not be suitable here.

[2] Yang: Logic order of Intro: privacy-preserving FHE, application of clustering, previous schemes, our scheme and techniques, compare, summary

[3] Yang: Qinghua’s intro is good, two party and vertically seperated data scenario.

same center, this step is solved by the stabilized method, which let the centers in the last iteration to play a role in the next iteration, so to divide by a plaintext, the whole number number of points. Dividing by a plaintext is a supported operation in CKKS. The batching technique of CKKS which enables multiple values to be encoded and encrypted in one ciphertext and multithreading are utilized to accelerate the execution of our program.

To check the effictiveness of our scheme, we compare the accuracy of our scheme with that of the original k-means clustering on plaintext and the result shows that the accuracy is almost equivalent, difference smaller than 2%. Compare to the work of Jaschke et al which is compeletely outsourced but requiring too much time, our work is 59178× faster. Compared to the work of Ali et al which implements only one iteration of k-means clustering, our work is about 1.3-39.1× faster, the larger the sacle of dataset, the faster of our work. And compared to the work of Mengyu Zhang et al which supports only a few times of iterations of k-means clustering, our work is about [4] . In summary, our pravicy-preserving k-means clustering scheme is completely outsourced, accurate and efficient.

– We use polynomial to approximate comparison funcition and use stabilized method to avoid ciphertext divison, bypassing these two unsupported operations in CKKS scheme. And we utilize bootstrapping to make our scheme a compeletely outsourced privacy-preserving k-means clustering algorithm based on FHE.

– We conduct a series of experiments on various widely used datasets to evaluate and compare our scheme with other works. From the results, it can be concluded that our work is accurate and efficient.

II. Prepare and Building Blocks / Background

A. Original k-means

k-means is one of the classic and well-known clustering algorithms, which needs no prior knowledge about dataset and uses iterative methods to obtain the final clusters.

1) Initialization: decide the "k", which means the number of clusters. then choose $\boldsymbol{k}$ points in dataset, which are named centers.
2) Repeat the following steps until centers change little or stop after a predefined times.
   a) Compute Distances: compute distances between all points in the dataset and all the centers. Here the distance can be l2-distances or other distance metrics.
   b) Compare Distances: reallocate points in dataset to its nearest center. These points that are allocated to the same center consist a cluster. So by the end of this step, we obtain $\boldsymbol{k}$ clusters.
   c) Update Centers: calculate $\boldsymbol{k}$ centers of the $\boldsymbol{k}$ new clusters, and then go back to step a).

[4]Yang: how to compare?

[5]

B. Fully Homomorphic Encryption

Fully homomorphic encryption(FHE) can do corresponding operations on ciphertexts and obtain the same result after decryption as addition or multiplication in plaintexts. There are some different FHE schemes that are accepted and examined by academia, such as BGV [1], BFV [2], [3], CKKS [4], GSW [5], TFHE [6] and FHEW [7]. Among them, GSW, TFHE and FHEW can homomorphically evaluate AND, OR, NOT operations on 0, 1. While BGV and BFV can homomorphically evaluate addition and multiplication on integers module, and CKKS can homomorphically evaluate addition and multiplication on real numbers or complex numbers. It is apperant that real numbers or complex numbers are much more powerful in real modern world, so the CKKS scheme gain much more attention than others.

All the six FHE schemes mentioned above have their security proved based on hard mathematic problem: Learning With Errors (LWE).

CKKS scheme can have many values encrypted in one ciphertext, supportting parallel computation naturally. The exact numbers of values in one ciphertext depends on the setup parameters.

C. The CKKS Scheme

In CKKS, the plaintext space is $\mathcal{M} = \mathbb{Z}[x]/\langle X^N + 1\rangle =: R$ while messages are complex vectors in $\mathbb{C}^\ell$ with $\ell = N/2$, where N is a power-of-two integer. The ciphertext space of CKKS is $C = R/qR$, where q is the *ciphertext modulus*, a large integer. The canonical embedding $\mathbb{R}[X]/\langle X^N+1\rangle \to \mathbb{C}^\ell$ maps $m(X) \in R$ into $\boldsymbol{m} \in \mathbb{C}^\ell$ by evaluating $m(X)$ at the primitive $2N$-roots of unity $\xi_j = \xi^{5^j}$ for $0 \le j < \ell$. The inverse of the canonical embedding encodes a message $\boldsymbol{m}$ as a plaintext $m(X)$. Thus, CKKS naturally support SIMD operations, i.e., performing an operation on a ciphertext corresponds to performing the same operation on $\ell = N/2$ entries of $\boldsymbol{m}$ in parallel. Each entry of the message $\boldsymbol{m} \in \mathbb{C}^\ell$ is called a *plaintext slot.*

For $\boldsymbol{x} = (x_i)_{0\le i<\ell}$ and $\boldsymbol{y} = (y_i)_{0\le i<\ell}$, let ct.$\boldsymbol{x}$ and ct.$\boldsymbol{y}$ be the ciphertext encrypted by CKKS under a same public key. Then CKKS supports the following basic operations:

- $\mathsf{Add}(\mathrm{ct}.\boldsymbol{x}, \mathrm{ct}.\boldsymbol{y}) = \mathsf{Enc}(\boldsymbol{x}+\boldsymbol{y})$.
- $\mathsf{Mul}(\mathrm{ct}.\boldsymbol{x}, \mathrm{ct}.\boldsymbol{x}) = \mathsf{Enc}(\boldsymbol{x}\circ\boldsymbol{y})$, where $\circ$ is for Hadamard product, i.e., component-wise multiplication.
- $\mathsf{CMul}(\boldsymbol{m}, \mathrm{ct}.\boldsymbol{x}) = \mathsf{Enc}(\boldsymbol{m}\circ\boldsymbol{x})$, where $\boldsymbol{m}$ is a message $\mathbb{C}^\ell$; for $m \in \mathbb{C}$, $\mathsf{CMul}(m, \mathrm{ct}.\boldsymbol{x})$ is a special case of $\mathsf{CMul}(\boldsymbol{m}, \mathrm{ct}.\boldsymbol{x})$ with $\boldsymbol{m} = (m, \ldots, m)$.
- $\mathsf{Sl}_{[i,j]}(\mathrm{ct}.\boldsymbol{x})$ convert a ciphertext $\mathrm{ct}.\boldsymbol{x} = \mathsf{Enc}(x_0, \ldots, x_{\ell-1})$ into a ciphertext that encrypts $(\boldsymbol{0}, x_i, x_{i+1}, \ldots, x_j, \boldsymbol{0})$, equivalent to $\mathsf{CMul}(\boldsymbol{m}, \mathrm{ct}.\boldsymbol{x})$ for $\boldsymbol{m} = (\underbrace{0,\ldots,0}_{i-1}, \underbrace{1,1,\ldots,1}_{j-i+1}, 0, \ldots, 0)$.
- $\mathsf{Rot}_k(\mathrm{ct}.\boldsymbol{x})$ convert $\mathrm{ct}.\boldsymbol{x} = \mathsf{Enc}(x_0, \ldots, x_{\ell-1})$ into a new ciphertext $\mathsf{Enc}(x_k, \ldots, x_{\ell-1}, x_0, \ldots, x_{k-1})$.

[5]Chen: I moved the stabilized k-means section here. But we should make it more rigorous. Right?

It is well-known that the computational efficiency of an FHE based algorithm is primarily influenced by two key factors, i.e., the multiplicative depth (including Mul and CMul, and the number of rotations (Rots) on ciphertext. The deeper the multiplication depth, the larger the ciphertext modulus, leading to higher computational costs. In addition, according to our test, for the CKKS scheme implemented in SEAL [8], the efficiency ratio between a ciphertext rotation Rot and a ciphertext multiplications Mul can be as large as $8 : 1$. Thus, in practice, one mainly focuses on optimizing the multiplicative depth and the number of rotations. [6]

III. Building Bolcks

[7]

A. Ciphertexts Comparison Approximation by Polynomial

FHE schemes can do addition or multiplication on ciphertexts, but not comparison on ciphertexts. To compare two ciphertexts is equivalent to evaluate sign function on single ciphertext, since FHE schemes support homomorphic addition.

Then how to construct sign function using addition and multiplication? Polynomial approximation was proposed as a good solution. But a better approximate polynomial needs more times of addition and multiplication, which means more levels consumed.

There have been some researches on polynomial approximation of sign function, aiming to be used in FHE. We adapt the approximation in this paper [9], which proves a theoretically optimal method of polynomial approximation of sign function and gives a general expression.

B. Parallel One-to-one Comparison to Speedup

The most difficult part in k-means realization with FHE is that to reallocate points in dataset to its nearest center, because this requires to find the smallest value in an array. As said above, the comparison operation is demanding in FHE, which needs polynomial approximation and many levels to be consumed. If we practice the find-smallest operation in FHE as the usual sort algorithm does on plaintext, the number of comparison operation needed is at least the length of array minus one. And most frustrating is that these comparison operations have to be done in series but not in parallel, so the pack-acceleration advantage of CKKS scheme can not be utilized.

Can we utilize the pack-acceleration advantage of CKKS scheme to practice the find-smallest operation in parallel? The key is to avoid serial comparison operations. If the size relations of every two values is known, it is enough to determine the biggest or smallest value in an array. Though it may cost additional comparison operations to know the size relations of every two values, the point is that now the comparison operations can be done in parallel. So we can utilize the pack-acceleration advantage of CKKS scheme to save appreciable time.

[6]Chen: We should add more about CKKS BTS and may delete what are not neccessary. We'd better make CKKS BTS an independent subsection.

[7]Yang: building blocks here, containing three parts ?

give an example of onehot encode.

	8	7	9	6	result
8	*	0	1	0	0
7	1	*	1	0	0
9	0	0	*	0	0
6	1	1	1	*	1

TABLE I
An easy example illustrating onehot comparison

We emphasize that this onehot vector encode can be used in other applications where needs to find the biggest or smallest element in an array to save considerable time and levels.

C. Stabilized k-means

[8] FHE schemes support addition and multiplication on ciphertexts, but not division on ciphertexts. This creates a difficulty for the "Update Centers" step in k-means, since this step needs to divide the sum of points in a cluster by the number of points in the cluster to obtain the new center of the cluster.

So to avoid the division, we adapt the stabilized k-means. Instead of divide by the number of points in a cluster, which is encrypted, we divide by the number of all points in the dataset, which is not encrypted. But now the problem is that we are now dividing by something bigger. So to keep the result right, we have also to add something to the sum of points in a cluster before division. The centers in last iteration are added.

IV. Detailed Algorithm

Our protocol consists of two parts, data owner and computation server. Data owner generates secret key $\boldsymbol{sk}$ and evaluation key $\boldsymbol{ek}$ and the size of dataset is of number of points $\boldsymbol{n} \times$ dimension $\boldsymbol{d}$. Before encrypting the dataset, some preprcess need to be done on the dataset so to make the computation steps on the server side can be operated successfully. Then data owner encrypts the preprocessed dataset and send the ciphertext to computation server. Except the ciphertext of dataset, data owner has also to sent some necessary parameters, including dimension $\boldsymbol{d}$, number of points $\boldsymbol{n}$, number of centers $\boldsymbol{m}$, and the times of iterations $\boldsymbol{T}$ want to be done.

After receiving the ciphertext of dataset, the computation server randomly chooses $\boldsymbol{m}$ points as the initial centers. After $\boldsymbol{T}$ iterations, the computation server send the result to the data owner. Data owner decrypts the result and obtain a bool matrix whose size is $\boldsymbol{n} \times \boldsymbol{m}$. Each line contains the information that which class the point should be assigned to. For example, when $\boldsymbol{m}$ is 4, one line could be (0.01, 0.84, 0.02, 0.10), then the point should be assigned to the second class since 0.84 is the max value in this line.

Below we will first give an easy example illustrating the computation steps in the computation server side. Then point

[8]Yang: Stabilized k-means may be viewed as a building block to aviod ciphertext division

out some differences in concrete implementation from this easy illustrative example. And at last, we give an analysis of our protocol.

A. *An Illustrative Example*

[9] [10] In this section, we illustrate our algorithm with an easy example.

Suppose now we have 7 points, (3,3), (3,7), (4,8), (8,8), (7,2), (8,4), (9,1). At initializaiton step, we randomly choose (3,3), (3,7), (7,2), (8,4) as initial centers. Since one ciphertext can hold many values, we allocate values like the manner below. Each line represents a ciphertext. For dimension of 2, 4 ciphertexts are needed, regardless of the number of centers, unless npoints multiplying ncenters outnumbers the capcity of one ciphertext. In this easy example, npoints multiplying ncenters is under the capcity of one ciphertext. And our real experiment is also the case.

All the values are in ciphertext state, unless noted explicitly otherwise.

(3,3,4,8,7,8,9, 3,3,4,8,7,8,9, 3,3,4,8,7,8,9, 3,3,4,8,7,8,9)
(3,7,8,8,2,4,1, 3,7,8,8,2,4,1, 3,7,8,8,2,4,1, 3,7,8,8,2,4,1)

(3,3,3,3,3,3,3, 3,3,3,3,3,3,3, 7,7,7,7,7,7,7, 8,8,8,8,8,8,8)
(3,3,3,3,3,3,3, 7,7,7,7,7,7,7, 2,2,2,2,2,2,2, 4,4,4,4,4,4,4)

TABLE II
INITIALIZATION

Now comes the computing distance step. Sum the suqare of the difference of line 1 and 3, which represents the first dimension, and the suqare of the difference of line 2 and 4, which represents the second dimension. There are four segments, each representing the distances between all the seven points and one center.

(0,16,26,50,17,26,40, 16,0,1,26,41,34,72, 17,41,45,37,0,5,5, 26,34,32,16,5,0,10)

TABLE III
COMPUTE DISTANCE

After computing distances, we need to compare distances to allocate points to its nearest center. Using the matrix comparison method to process the four first values in the four segments, which represent the distances of first point to the four centers. We obtain the result as (1,0,0,0), meaning that the first point should be allocated to the first center.

	0	16	17	26	result
0	*	1	1	1	1
16	0	*	1	0	0
17	0	0	*	1	0
26	0	0	0	*	0

TABLE IV
COMPARE DISTANCE

We record the result also in the four segments format, for convience of following steps.

[9]Chen: I change the title of this section.
[10]Yang: make this example more compact.

(1,-,-,-,-,-,-, 0,-,-,-,-,-,-, 0,-,-,-,-,-,-, 0,-,-,-,-,-,-)

TABLE V
COMPARE DISTANCE

Do the procedure above to other points, obtaining the complete result of comparison.

(1,0,0,0,0,0,0, 0,1,1,0,0,0,0, 0,0,0,0,1,0,1, 0,0,0,1,0,1,0)

TABLE VI
COMPARE DISTANCE

We can now update centers with the comparison result. Process each dimension seperately. When the comparison result is 1, we pick up the value of points, otherwise the old value of centers.

(1,0,0,0,0,0,0, 0,1,1,0,0,0,0, 0,0,0,0,1,0,1, 0,0,0,1,0,1,0)
(3,3,4,8,7,8,9, 3,3,4,8,7,8,9, 3,3,4,8,7,8,9, 3,3,4,8,7,8,9)
(3,3,3,3,3,3,3, 3,3,3,3,3,3,3, 7,7,7,7,7,7,7, 8,8,8,8,8,8,8)

(3,3,3,3,3,3,3, 3,3,4,3,3,3,3, 7,7,7,7,7,7,9, 8,8,8,8,8,8,8)

TABLE VII
UPDATE CENTERS

Do the same for the second dimension.

(1,0,0,0,0,0,0, 0,1,1,0,0,0,0, 0,0,0,0,1,0,1, 0,0,0,1,0,1,0)
(3,7,8,8,2,4,1, 3,7,8,8,2,4,1, 3,7,8,8,2,4,1, 3,7,8,8,2,4,1)
(3,3,3,3,3,3,3, 7,7,7,7,7,7,7, 2,2,2,2,2,2,2, 4,4,4,4,4,4,4)

(3,3,3,3,3,3,3, 7,7,8,7,7,7,7, 2,2,2,2,2,2,1, 4,4,4,8,4,4,4)

TABLE VIII
UPDATE CENTERS

Sum the values for each segment and then divide by the number of points(7, in this example). Since the number of points is never changed and has no threats to information privacy, we store it in plaintext state. Dividing a plaintext is supported in FHE. After the division, we now obtain the new value of centers.

(3,3,3,3,3,3,3, 3.14,3.14,3.14,3.14,3.14,3.14,3.14
7.29,7.29,7.29,7.29,7.29,7.29,7.29 8,8,8,8,8,8,8)
(3,3,3,3,3,3,3, 7.14,7.14,7.14,7.14,7.14,7.14,7.14
1.86,1.86,1.86,1.86,1.86,1.86,1.86 4.57.4.57,4.57.4.57,4.57.4.57,4.57)

TABLE IX
UPDATE CENTERS

By here this iteration is done, things are ready for next iteration. Repeat the steps above again.

B. *Building Blocks / The Concrete Construction*

[11] Though the easy example above captures the majority of key points, there are some differences in concrete implementation.

In the "Comparing distances" step, since the domain of the compare function approximated by polynomial is [-1, 1], the dataset has to be rescaled linearly to make sure that the distances computed is not bigger than 1.

[11]Chen: Title changed, but to be confirmed.

The onehot result of comparison in the example above is exactly 1 or 0, but in the ciphertext computation, since the compare function is approximated by polynomial, the output of the compare function is a float number between 0 and 1.

C. *Our Protocol*

use format language to describe content above. and give a summary here.

a) Correctness:

b) Complexity:

c) Security: Security is obvious. Data owner encrypts dataset and then send this ciphertext plus evaluation key $\boldsymbol{ek}$ and some parameters ($\boldsymbol{n}$, $\boldsymbol{m}$, $\boldsymbol{d}$, $\boldsymbol{T}$). It is apperent that $\boldsymbol{T}$ has no relevence to dataset. The $\boldsymbol{n}$, $\boldsymbol{m}$ and $\boldsymbol{d}$ have some relevance to the dataset, but should not be veiwed as privacy leak. And all the computations done in the computation server side are on ciphertexts. Finally, data owner receives result and uses $\boldsymbol{sk}$ to decrypt it. So during all the process, only data owner has access to $\boldsymbol{sk}$. If the FHE scheme CKKS is secure, then our protocol is secure.

V. Implementation and Experiments

The machine we use is equipped with Intel Xeon Gold 6248R(3.00GHz, 24Core) and 128G(32G*4) memory.

A. *Clustering Accuracy*

dataset	Lloyd's Algorithm	Our Algorithm
Chainlink	65.4%	65.0%
EngyTime	94.9%	94.7%
Hepta		
Lsun	78.8%	80.6%
Tetra	100.0%	100.0%
TwoDiamonds	100.0%	100.0%
WingNut	96.3%	96.2%

TABLE X
Accuracy

B. *Comparison with PEGASUS*

[12] Here a table presents the comparison of time used.

npoints	ncenters	PEGASUS	ours	Speedup
256	2	1.35min	1min26.11s, 36.70GB	
	4	2.33min	1min26.05s, 31.80GB	
	8	4.09min	1min41.16s, 36.49GB	
1024	2	3.66min	1min26.41s, 33.80GB	
	4	7.57min	1min26.40s, 36.31GB	
	8	15.34min	1min41.32s, 36.25GB	
4096	2	13.95min	1min26.99s, 35.21GB	
	4	26.61min	1min26.97s, 35.81GB	
	8	52.04min	1min43.06s, 41.12GB	30×

TABLE XI
Compare with PEGASUS

The machine we use is different form PEGASUS, so we run a baseline example provided in PEGASUS code, to make the time consumed can be compared fairly.

From the table above, it is apperent that our method is much more efficient than PEGASUS especially when the dataset's scale is large.

[12]Yang: Maybe not include the last bts time? For a fair comparison?

C. *Comparison with THREE*

Work	Version	Threads	Run Time(T=10)
Jaschke et al	exact	one	363.90days
Jaschke et al	approximate	one	154.70h
Qinghua	exact	one	1606.36s
Our Work	exact	one	626.59s

TABLE XII
Compare with PEGASUS

D. *Comparison with qinghua*

Here a table presents the comparison of time used.

VI. Discussion

Some discussions here.

References

[1] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc 3rd ITCS (January 8–10, 2012, Cambridge, MA, USA)*, S. Goldwasser, Ed. New York: ACM, 2012, pp. 309–325, https://doi.org/10.1145/2633600.

[2] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in Cryptology – Proc CRYPTO 2012 (August 19–23, 2012, Santa Barbara, CA, USA)*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds. Heidelberg: Springer, 2012, vol. 7417, pp. 868–886, https://doi.org/10.1007/978-3-642-32009-5_50.

[3] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive https://eprint.iacr.org/2012/144, 2012.

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proceedings of ASIACRYPT 2017 – 23rd International Conference on the Theory and Applications of Cryptology and Information Security (December 3-7, 2017, Hong Kong, China), Part I*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds. Heidelberg: Springer, 2017, vol. 10624, pp. 409–437, https://doi.org/10.1007/978-3-319-70694-8_15.

[5] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptolog–Proc CRYPTO 2013 (August 18-22, 2013, Santa Barbara, CA, USA), Part I*, ser. Lecture Notes in Computer Science, R. Canetti and J. A. Garay, Eds. Heidelberg: Springer, 2013, vol. 8042, pp. 75–92, http://dx.doi.org/10.1007/978-3-642-40041-4_5.

[6] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020, https://doi.org/10.1007/s00145-019-09319-x.

[7] L. Ducas and D. Micciancio, "FHEW: Bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology - Proceedings of EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Part I (April 26-30, 2015, Sofia, Bulgaria)*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds. Heidelberg: Springer, 2015, vol. 9056, pp. 617–640, https://doi.org/10.1007/978-3-662-46800-5_24.

[8] Microsoft, "Microsoft SEAL (release 4.1.1)," Accessed in July, 2023, https://github.com/microsoft/SEAL.

[9] J. H. Cheon, D. Kim, and D. Kim, "Efficient homomorphic comparison methods with optimal complexity," in *Advances in Cryptology – ASIACRYPT 2020, Part II (Daejeon, South Korea, December 7–11, 2020)*, ser. Lecture Notes in Computer Science, S. Moriai and H. Wang, Eds. Cham: Springer, 2020, vol. 12492, pp. 221–256, https://doi.org/10.1007/978-3-030-64834-3_8.