Hindawi
Security and Communication Networks
Volume 2019, Article ID 3648671, 14 pages
https://doi.org/10.1155/2019/3648671

Research Article

A Bitwise Design and Implementation for Privacy-Preserving Data Mining: From Atomic Operations to Advanced Algorithms

Baek Kyung Song, Joon Soo Yoo, Miyeon Hong, and Ji Won Yoon

Korea University, Seoul, Republic of Korea

Correspondence should be addressed to Ji Won Yoon; jiwon_yoon@korea.ac.kr

Received 7 April 2019; Accepted 14 August 2019; Published 16 October 2019

Academic Editor: Stelvio Cimato

Homomorphic encryption (HE) is considered as one of the most powerful solutions to securely protect clients' data from malicious users and even severs in the cloud computing. However, though it is known that HE can protect the data in theory, it has not been well utilized because many operations of HE are too slow, especially multiplication. In addition, existing data mining research studies using encrypted data focus on implementing only specific algorithms without addressing the fundamental problem of HE. In this paper, we propose a fundamental design and implementation of data mining algorithm through logical gates. In order to do this, we design various logic of atomic operations in encrypted domain and finally apply these logic to well-known data mining algorithms. We also analyze the execution time of atomic and advanced algorithms.

1. Introduction

With the progress of storage in the cloud server, advanced data process and analysis using machine learning and data mining techniques are developed to extract valuable information. However, the concern about the data privacy and security issues has occurred in storing and managing information in cloud servers. This is because the server must decrypt the data in order to process the data encrypted in conventional cryptosystems such as AES and DES, even though the client transmits the data to the server in encrypted form. Eventually, users must share the decryption key with the cloud, which can lead to data infringement by a malicious server.

Homomorphic encryption (HE) [1, 2] is mentioned as one of the most powerful solutions to the data security problem in the cloud, since the data can be processed in the encrypted domain without decryption. However, data analysis with HE is not so popular in real world although it is highly recommended for providing the proper security to the cloud. The major reason is the fact that it is difficult to link HE and machine learning. As known, HE is a new cryptosystem which uses profound, mathematical property with lattice, which makes it difficult for the data scientists to understand and use.

In addition, a few well-known HE algorithms support only very simple operations such as addition and multiplication between integers. Although Gentry [3] presented fully homomorphic encryption (FHE) which allows all operations on the ciphertext to be theoretically unlimited, it had many limitations in adapting to the real cloud model [4]. Since the implementation and development of the encryption algorithm are not main interest to theoretical cryptographers, the practical usage and implementation are rarely developed compared to the theoretical progress in FHE. Therefore, to date, FHE has been limited to be applied only to specific algorithms without solving the fundamental problems of FHE [5–11].

From this point of view, we propose a FHE computation method that can be applied more generally by using bitwise logical circuits, rather than algorithms that operate only under certain conditions. By designing the basic operations necessary for machine learning, we make a universal link between HE and machine learning. People who are studying FHE can easily apply machine learning with homomorphic operations. Furthermore, machine learning researchers will be able to run data-driven data analysis algorithms with encrypted data although they do not have the knowledge about FHE at all.

Our contribution of this paper is threefold:

(i) In order to build simple data mining techniques with FHE, we design various atomic operations including absolute value operation, multiplication, comparison, and sorting through the gate operation provided by the TFHE library

(ii) In contrast to the integer-based FHE scheme in which possible operations are limited, all the operations including division and log can be designed in the bit-based FHE scheme

(iii) We finally demonstrate the applicability of the several well-known data mining techniques using our proposed bitwise FHE schemes: the linear regression, the logistic regression, k-NN classifier, and k-means clustering

2. Background

2.1. Homomorphic Encryption. Homomorphic encryption (HE) [1, 2] is a cryptosystem in which the result of operations between ciphertexts is equal to the result of operations between plaintexts when decrypted. The operations on the ciphertexts of a and b can be expressed as $a \circ b = \mathbb{D}[\mathbb{E}[a] \bullet \mathbb{E}[b]]$ where $\mathbb{E}[\cdot]$ and $\mathbb{D}[\cdot]$ denote encryption and decryption, respectively.

The concept of HE was first presented in 1978 by Rivest et al. [12]. Many HE schemes have been introduced since then, and the most popular one was the Paillier cryptosystem, proposed by Paillier [13] in 1999. However, they were partial HE with a limited number of operations since the encryption noise is amplified each time the operation is performed. The solution to this noise accumulation problem was the fully homomorphic encryption (FHE) of Gentry [3] in 2009. Gentry [3] proposed a bootstrapping algorithm that removes accumulated noise, thereby eliminating the limit on the number of operations. However, this Gentry [3] technique had to encrypt each plaintext bit by bit. It was a heavy burden on memory because the size of the ciphertext was so large. In addition, the bootstrapping operation was performed with a very complicated algorithm, so it took dozens of minutes to bootstrap a bit. For these reasons, many FHE libraries now use integer-based schemes, but this also has the disadvantage that the possible operations are very limited.

2.1.1. TFHE Library for FHE. In 2017, Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène proposed TFHE [14] library which is an improved version of FHEW [15] library. It has the bit-by-bit encryption scheme similar to Gentry's initial FHE [3]. However, unlike [3], TFHE has constructed operations in a more fundamental way than addition and multiplication between ciphertexts. It is the binary circuit that was used for the encrypted bits operation. In other words, TFHE supports NOT, AND, OR, NAND, NOR, XOR, and XNOR gate operations between encrypted bits, allowing users to construct encrypted circuits using these logical operations. Another advantage of TFHE is that it efficiently solves the bootstrapping problem, which was the biggest obstacle to using FHE. This is designed to perform a bootstrapping function automatically whenever a single operation is performed, unlike the conventional FHE, in which a direct bootstrapping must be performed to remove noise each time a certain number of operations are performed. In other words, it is possible to perform computation without limitations. Here, the bootstrapping algorithm is performed with a time of less than one 0.1 second and has the fastest performance among all of the preceding FHE schemes.

In addition, through supporting the multiplexer function, convenience of implementation and speed of circuit are more improved. In the below function, a is a multiplexer factor and outputs either b or c depending on the value:

$$\mathrm{MUX}(a,b,c) = \begin{cases} b, & (\text{if } a = 0), \\ c, & (\text{if } a = 1). \end{cases} \quad (1)$$

2.2. Data Mining and Machine Learning Algorithms

2.2.1. Linear Regression. Linear regression is the most popular model for predicting target value of y. It is the method that estimates the coefficients of the linear equation, involving one or more independent variables. Several types of process exist to optimize the values of the coefficients. We focus on gradient descent, iteratively minimizing the error of the training data.

2.2.2. Logistic Regression. Logistic regression is a special case of generalized linear model in which the target variable is binary such as pass or fail, live or death, etc. In general, logistic regression makes an inference on parameters of sigmoid function which determines classification of modeling binary or categorical dependent variables.

2.2.3. k-Nearest Neighbors (kNN) Classification. In data mining, the k-nearest neighbors algorithm is one of the most well-known and useful supervised methods for classifying a dataset. Given the classified data with several classes, the kNN determines the class of new input data based on its neighbors. At this time, the label of the input data is set to the largest number of labels of the closest k data. In addition, there are many ways to calculate the distance between data, typically Euclidean distance. Depending on which distance measurement method is used, different results may be obtained.

2.2.4. k-Means Clustering. Unlike kNN, the k-means clustering algorithm grasps the relationship of unlabeled data and clusters them into k clusters. The k-means clustering sets the representative value of each cluster and assigns each data to the cluster with the closest representative value. After forming clusters for k representative values initially set arbitrarily, the mean of each cluster is newly representative of each cluster. This process is repeated until the cluster converges, and finally, the data are clustered into k clusters. The result of the k-means is affected by the distance measurement method as well as the kNN.

3. Problems

3.1. FHE for Machine Learning. Although machine learning and FHE have long history, their research has been conducted separately for a long time. Recently, as the era of cloud computing comes, privacy-preserving machine learning and data mining have been introduced as a hot topic. There have been several studies on connecting FHE and machine learning [6, 7, 9, 10, 16].

However, as mentioned in Section 2, there is a limitation that it is difficult to apply FHE to machine learning algorithms because it is only possible to perform limited operations such as addition and multiplication in most libraries. Accordingly, existing machine learning studies using encrypted data have focused on implementing specific algorithms such as Naïve Bayes classifier [9] or linear regression [16]. In addition, since FHE requires complex theoretical knowledge, it is difficult for general machine learning engineers to understand its concept. Worse, in order to use the FHE scheme, we need a technique to replace all operations on plaintext with homomorphic operations.

In this paper, we focused on how to efficiently implement basic atomic operations and universal application to various machine learning algorithms. These studies will be a good mediator between FHE and machine learning.

3.2. Integer-Level Encryption vs. Bitwise Encryption. The FHE, which operates in integer space, takes scalar integers or polynomials with integer coefficients as input and then performs an operation on an integer basis. Therefore, additional integer encoding is required for real data that are not integers. Previous research studies about FHE application have used the rounding function to convert real numbers to integers for the encoding and decoding processes. Most of them used the scaling constant k before rounding to preserve the original number. In order to recover the encoded value, k must be divided from the decrypted result as follows:

(1) Encoding: $\lfloor k \times a \rceil$ for a plaintext $a \in \mathbb{R}$

(2) Encryption: $c = \mathbb{E}[\lfloor k \times a \rceil]$

(3) Decryption and decoding: $a \approx (1/k) \cdot D[c]$

However, there is a problem with this method, which is to use an approximation rather than an accurate data. The approximation accuracy of the data is determined by the scaling constant, and the user must also determine this constant.

On the other hand, bitwise encryption does not require encoding process to an integer because all real-valued data can be represented in bits. In addition, since the computer stores and processes data on a bit-by-bit basis, a generalized encryption scheme can be easily applied to any data.

In this paper, we introduce the logic of various operations for the bitwise encryption scheme using the TFHE library. We present a method for constructing atomic operations using the circuit operation for each bit after converting integer data into bits. Table 1 shows the logical operators used in this study and their notation.

4. Designing Homomorphic Atomic Operations

Our method uses the TFHE library, so we perform all operations on a bit-by-bit basis. This is similar to the way that binary data in a plaintext are processed by a computer using AND/OR/NAND/NOR/XOR/XNOR/NOT gates. However, since we do not know actual values to be computed, the operations should be differently designed from algorithms in the plaintext, such as using ciphertext in if-statement (for example, "If ciphertext $= \mathbb{E}[0]$, then follow below command"; in this case, we cannot compare ciphertext and $\mathbb{E}[0]$ typically). Considering these characteristics, we introduce a new design of the atomic operations in this section. The atomic operations include Addition, 2's Complement, Subtraction, Equivalent Comparison, Large and Small Comparison, Shift, Absolute, Multiplication, and Division. Note that Addition, Subtraction, and Multiplication among these atomic operations have already been introduced in the literature [14, 17]. However, other atomic operations have rarely been studied although they are highly significant for numerical computation. We demonstrate the description and algorithms of both already and rarely studied atomic operations in this section because they are separately classified as homomorphic atomic operations from the advanced homomorphic data mining algorithms in Section 4.

All algorithms introduced in this paper are implemented and evaluated with Intel i7-7700 3.60 GHz, 8.0 GB RAM, and Ubuntu 16.04.4 LTS.

4.1. Addition Operation. Addition is one of the most basic operations. There are many ways to implement full adder circuit with basic gates such as 9 NAND gates and 7 NOR and 5 NOT gates. However, since the number of basic gates is relative to speed of the circuit in the TFHE library, addition can be more efficiently designed by using only 2 XOR, 2 AND, and 1 OR gates. More details are described in Figure 1.

In the circuit diagram of Figure 1, the least significant bit (lsb) of a and b is input to the upper bit input, and c_0, which is the lsb of the carry, is initialized to $\mathbb{E}[0]$. s_i passing through the circuit is the sum of the corresponding bits, and c_{i+1} is the carry of the next bit.

4.2. 2's Complement Operation. It is necessary to express a negative number in order to perform an integer binary data operation. There are two ways to represent negative numbers in a computer, mainly the 1's complement method and 2's complement method. The 1's complement method has a simpler advantage than the 2's complement method when representing a negative number. The desired number is operated through a XOR gate with a single bit 1. The process can be replaced to taking the NOT gate for every bit of the desired number. This is because the NOT gate is significantly faster than the XOR gate. However, the 1's complement

Table 1: The notations of the logical operators.

Operator	NOT	AND	OR	NAND	NOR	XOR	XNOR
Notation		$\wedge$	$\vee$	$\overline{\wedge}$	$\overline{\vee}$	$\oplus$	$\odot$

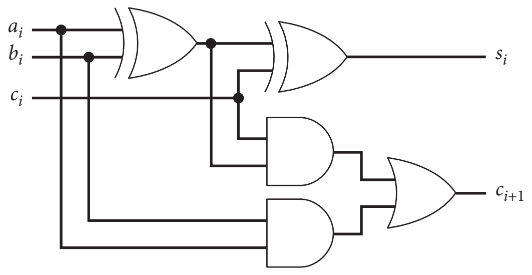


Figure 1: The circuit design for the binary full adder in the FHE scheme.

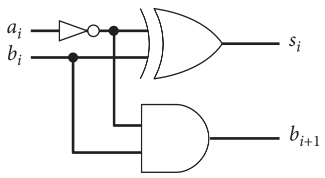


Figure 2: The circuit design for 2's complement in the FHE scheme.

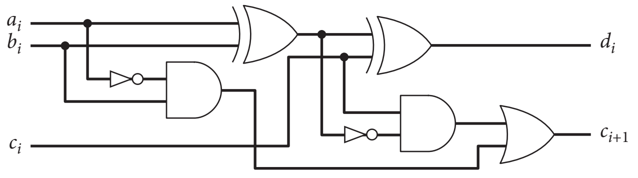


Figure 3: The circuit design for subtraction.

method has two ways of representing 0, and it is necessary to use a logic different from the plaintext to perform operations such as addition and subtraction. The method of improving this is the 2's complement method, which is represented by adding the integer 1 in the 1's complement method. Since, when 1 is represented by a binary number, it is filled with zeros except for lsb, the carry can be added to the next bit of 1 to perform addition. Therefore, when adding, it is possible to reduce the speed by adding the NOT gate to the half adder which does not need carry, without using the previous full adder: $b_{i+1} = a_i \wedge b_i$ and $s_i = a_i \oplus b_i$. This is expressed by a circuit as shown in Figure 2.

Set the number a and $b = [00\ldots01]$ to take the 2's complement operation and input from each lsb. The output s_i from the above circuit is the result of the corresponding bits; the carry is b_{i+1}, which is the next input.

4.3. Subtraction Operation. In a typical computer environment, you can implement subtraction using the 2's complement method and addition, so subtraction logic is not implemented separately. However, subtraction can be processed using the 2's complement method and addition as in plaintext, but it can be newly implemented with 2 XOR, 2 AND, 1 OR, and 2 NOT gates. The detailed circuit diagrams are demonstrated in Figure 3.

Subtraction enters the input from lsb of a and b. d_i passed through the circuit is the result of the subtraction of that bit, and c_{i+1} is the carry of the next bit. d_i and c_{i+1} are defined according to their value after defining D as shown in the following equation: $D = a_i - b_i - c_i$. In this equation, if $D = 1$, then $d_i = 1$ and $c_{i+1} = 0$. If $D = 0$, then $d_i = 0$ and $c_{i+1} = 0$. If $D = -1$, then $d_i = 1$ and $c_{i+1} = 1$. And if $D = -2$, then $d_i = 0$ and $c_{i+1} = 1$.

4.4. Comparison Operation

4.4.1. Equivalent Comparison. Equivalent comparison in plaintext compares each bit for two input values and outputs 1 if all are equal and 0 if there are other values. However, in encrypted data, it is possible to determine whether each bit is the same through an XOR gate, but since it comes out encrypted, it does not know what the value is. Therefore, to get the results we want, all the results of the XNOR gate of each bit are operated with the AND gate as shown in Figure 4. Then, $\mathbb{E}[0]$ is output when there are different bits in two inputs, and $\mathbb{E}[1]$ is output if each bit is the same value. Then, if the input values are different, $\mathbb{E}[0]$ is returned for the output and $\mathbb{E}[1]$ for the same input values.

4.4.2. Large and Small Comparison. We will explain this as a large comparison because the large comparison and the small comparison are logically similar. In a computer, large comparison is a system that outputs results when bits with different values are compared while comparing from upper bit to lower bit. However, since it is not known whether the value of the comparison of each bit is ciphertext of 1 or ciphertext of 0, it does not know which bit has a different value and which of the two numbers is larger. Thus, we have to use the new logic.

First, let us consider the sign bit of the result of subtracting the preceding number from the latter number of two inputs. If the preceding number is less than or equal to the latter number, $\mathbb{E}[0]$ is output and larger $\mathbb{E}[1]$ is output. Therefore, we will use this subtraction to make a large comparison. However, considering the speed of the circuit, we will use a method that uses a multiplexer function and XNOR gate. The detailed circuit diagrams are demonstrated in Figure 5. The result of the comparison is the result of repeating the circuit by the length of the data.

Larger than or equivalent comparison or smaller than or equivalent comparison can take a NOT gate as the result of a small comparison or a large comparison, respectively.

4.5. Shift Operation. Since the ciphertext is encrypted bitwise, it can be shifted in the same way as for the shift in plaintext. Shift the k bits to the left and fill the empty right k bits with $\mathbb{E}[0]$. Shifting k bits has the effect of multiplying 2^k as shown in Algorithm 1.

In this algorithm, "*HomCONSTANT*" is a function that produces one bit ciphertext corresponding to the input value

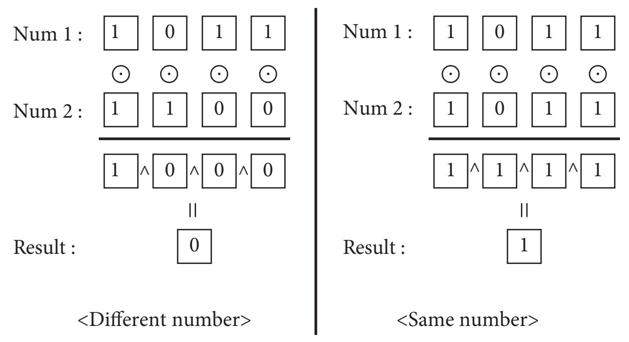


Figure 4: Example of equivalent comparison.

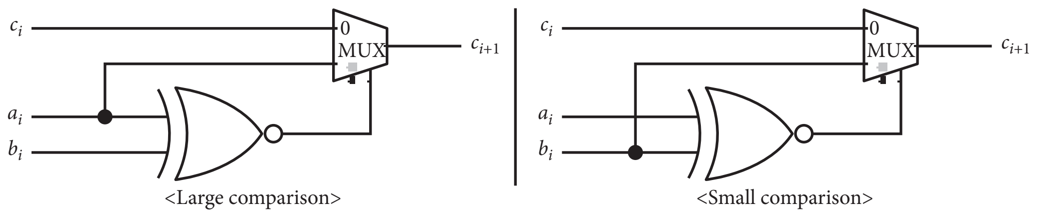


Figure 5: The circuit design for comparison.

```
    Input: $a = [a_{l-1}, a_{l-2}, \dots, a_0]$, $k$
    Output: LSHIFT($a$, $k$)
(1) for $i = 0 : (k-1)$ do
(2)     $a_i = HomCONSTANT(0)$
(3) end for
(4) for $i = k : (n-1)$ do
(5)     $a_i = HomCOPY(a_{i-k})$
(6) end for
(7) return $[a_{l-1-k}, \dots, a_0, \mathbb{E}[0], \dots, \mathbb{E}[0]]$
```

Algorithm 1: Pseudocode of left shift.

and "*HomCOPY*" is a function that produces the same one bit ciphertext as the result of the decryption, but different ciphertexts.

The right shift can be divided into a general shift, which is a method of shifting the upper k bits to $\mathbb{E}[0]$ after shifting like a left shift, and an arithmetic shift which shifts the upper k bits to the same value as the sign bit. An arithmetic shift is mainly used, and shifting k bits has the effect of dividing by 2^k.

4.6. Absolute Value Operation. In the plaintext, the absolute value algorithm outputs as it is if the most significant bit is 0 and takes the complement of 2 if the most significant bit is 1. Since the value of the most significant bit is not known in a ciphertext, a new algorithm must be designed. Let the original value be a and the value obtained by taking the complement of 2 to a be b; then, one is positive and the other is negative (except for 0). Now, let sign bit of a be a multiplexer factor, which returns a or b depending on the value:

$$|a| = \mathrm{HomMUX}(\mathrm{msb}(a), a, b). \quad (2)$$

4.7. Multiplication Operation. In general multiplication, multiplying m bits by n bits results in $(m+n)$ bits. When the two numbers to be multiplied are positive, the multiplicand is multiplied from the lsb of the multiplier to the upper bit as if it were calculated by hand. Then, the result of multiplication is the sum of all the left shifted values as the bit position of the multiplier increases. Thus, the smaller 1-bit of the multiplicand is, the more efficient it is. Therefore, we divide the multiplier by addition or subtraction to reduce the number of 1-bit as much as possible. However, as mentioned earlier, this is an algorithm that can be applied only to positive numbers, so a more advanced form of algorithm is needed to consider negative numbers. This is because, in the case of the unencrypted plaintext data, the sign of the data can be inspected by checking the msb, but in the case of the encrypted data, the value of the msb cannot be confirmed. That is, a new algorithm should be designed to output the correct result regardless of the sign of the given data. To solve this problem, we can calculate the product of positive numbers through an absolute value operation and then perform a 2's complement operation on the result according to the sign. That is, for multiplication of a and b, we follow the below way:

$$\begin{aligned} \mathrm{msb}(a) \oplus \mathrm{msb}(b) &= p, \\ M &= |a| \times |b|, \\ M' &= 2\text{'s complement of } M, \\ a \times b &= \mathrm{HomMUX}(p, M', M). \end{aligned} \quad (3)$$

Therefore, our algorithm adopts the latter method, and its circuit diagram is shown in Figure 6.

4.8. Division Operation. Binary division algorithms can be thought of as dividing input into positive cases and negative

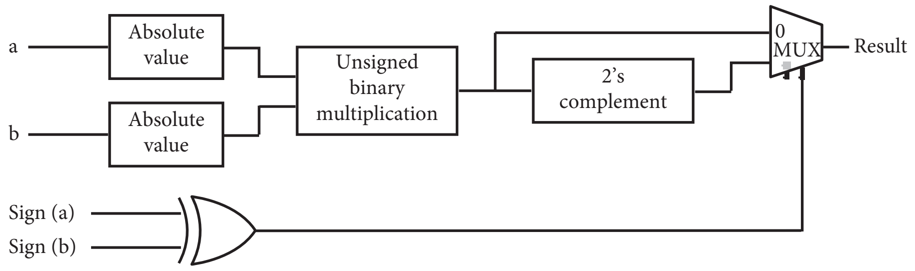


Figure 6: The circuit design for signed multiplication.

cases. First, let us consider the case where both the divisor and the dividend are positive. Let the array M, Q, and A have the same length of l, and initialize M to divisor, Q to dividend, and A to zero. The count value is the dividend length, l. And let $AQ = [A\|Q]$ with a length of $2l$ and start the main part of the algorithm.

If the divisor or dividend is negative, we need to use a slightly different algorithm. First, we can implement negative binary division algorithm by modifying Algorithm 2 slightly. However, since the sign of the input value cannot be known, when the negative binary division algorithm is implemented with a new algorithm, both algorithms must be performed and a single result should be output according to the sign of the input value. This is inefficient because it takes time to perform Algorithm 2 twice. Therefore, we will implement the signed binary division algorithm using a second method that uses absolute values and multiplexer function as in multiplication. That is, for signed binary division M and Q, we follow the following way:

$$\begin{aligned} \operatorname{msb}(M) \oplus \operatorname{msb}(Q) &= p, \\ D &= \text{positive binary division}(|M|, |Q|), \\ D' &= \text{2's complement of } D, \\ \frac{Q}{M} &= \operatorname{HomMUX}(p, D', D). \end{aligned} \tag{4}$$

5. Experiments

5.1. Basic Gate Experiment. We implemented the operations of Section 4 based on the basic gates and checked the speed of 1-bit basic gate operation in TFHE 1000 times.

As shown in Table 2, the basic gates except the NOT gate have the same speed, and the speed of the NOT gate is significantly lower than that of the other gates. Also, the multiplexer function is implemented differently from the basic gates so that there is a difference in speed. It can be seen that the speed of the multiplexer function is faster than the speed of computing basic gate about two times.

5.2. Number of Gates Used in Designed Homomorphic Atomic Operations. Since all gates except NOT gate and MUX gate have the same speed, we will denote execution time of these gates as T_G. Time of the MUX gate is represented by T_M, and the NOT gate is omitted because the speed converges to zero. Table 3 shows the number of gates used when performing designed homomorphic operations with l-bit input values for each operation.

Most of the operations listed in Table 3 are linear for data length. In shift operation, the position of bit is shifted without using a gate operation, and the number of gates in multiplication and division operations is proportional to the square of the data length.

5.3. Execution Time of the Homomorphic Atomic Operations. In Table 4, we measure the speed of the operations based on 16 bits. The speed of the shift operation is not measured because gate is not used; for nonlinear operations, we measured 8, 16, and 32 bits to see the change in speed.

Looking at the measured values, the doubling of the length of the data increases the speed of both algorithms by about four times. This is because the speed of addition, subtraction, and comparison operations constituting the multiplication and division is linearly increased with respect to the data length, and the number of iterations of the algorithm is also proportional to the length of the data.

6. Applications

6.1. Linear Regression. Given a d-dimensional input variable $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and its corresponding target variable $\mathbf{y}^{(i)} \in \mathbb{R}$ for $i = 1, 2, \ldots, n$, an inference on parameters of the linear function within hypotheses is defined as

$$h_\theta\left(\mathbf{x}^{(i)}\right) = \theta^{\mathrm{T}}\mathbf{x}^{(i)} = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_d x_d^{(i)}, \tag{5}$$

for $\mathbf{x}^{(i)} = [1, x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]^{\mathrm{T}}$, parameters $\theta = [\theta_0, \theta_1, \theta_2, \ldots, \theta_d]^{\mathrm{T}}$, and number of features, $d + 1$. This regression describes a hyperplane in the d-dimensional space of the independent variables $\mathbf{x}$.

In general, the linear regression can be easily estimated by using least square estimation as follows:

$$\widehat{\theta} = \left(\mathbf{X}\mathbf{X}^{\mathrm{T}}\right)^{-1}\mathbf{X}\mathbf{Y}^{\mathrm{T}}, \tag{6}$$

where $\mathbf{Y} = [\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(n)}]$, $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}]$. However, in FHE, it is rather difficult to design and implement the inversion matrix of equation (6). Therefore, instead of the exact solution, we choose an approximation estimation which is based on the gradient descent update in order to avoid the calculation of inverse matrix.

The approximation estimation uses error function to optimize the parameters of both simple and multiple linear regression as follows:

```
Input: divisor $M$, dividend $Q$
(1) Shift the $AQ$ to the left by one bit and let the upper $l$ bit of $AQ$ at $A$.
(2) Calculate $A - M$ and put it in $A$.
(3) If $A$ is negative, the last bit of $AQ$ becomes 0 and $A + M$ is calculated and put it in $A$ to return to the value before step 2.
(4) If $A$ is positive or zero, the last bit of $AQ$ is 0.
(5) The count value is decremented by 1.
(6) If the count is not 0, the algorithm goes to step 1 and the algorithm is progressed.
(7) If the count value is 0, the result of algorithm is output (the lower $l$ bit of $AQ$ becomes the quotient and the upper $l$ bit becomes the
    remainder).
```

Algorithm 2: Positive binary division operation.

Table 2: Execution time of basic gates (s).

AND	OR	NAND	NOR	XOR	XNOR	NOT	MUX
11.9	11.9	11.9	11.9	11.9	11.9	0.000162	22.4

Table 3: Time complexity of designed homomorphic atomic operation with l-bit input values.

Operation	Time complexity of designed operations
Addition	$(5l - 3)T_G$
2's complement	$(2l - 3)T_G$
Subtraction	$(5l - 3)T_G$
Equivalent comparison	$(2l - 1)T_G$
Large (small) comparison	$lT_G + lT_M$
Shift	≈ 0
Absolute value	$2lT_G + lT_M$
Multiplication	$(6l^2 + 4)T_G + 4lT_M$
Division	$(8l^2 - 4l + 4)T_G + (l^2 + 2l)T_M$

Table 4: Execution time of designed homomorphic atomic operation.

Operation	Estimation (s)	Execution (s)	Error (%)
Addition	0.916	0.917	0.10
2's complement	0.345	0.351	1.73
Subtraction	0.916	0.919	0.32
Equivalent comparison	0.369	0.375	1.62
Large (small) comparison	0.548	0.548	0
Absolute value	0.703	0.712	1.28
Multiplication_8	5.334	5.396	1.11
Multiplication_16	19.759	19.898	0.70
Multiplication_32	76.028	77.413	1.82
Division_8	7.551	7.772	2.92
Division_16	30.108	30.553	1.47
Division_32	120.38	121.781	1.16

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta\left(\mathbf{x}^{(i)}\right) - \mathbf{y}^{(i)}\right)^2. \tag{7}$$

The main goal of linear regression is to fit a straight line through the data, so we minimize the error function $J(\theta)$. Gradient descent is achieved by an algorithm that starts with an initial θ and repeatedly performs the update:

$$\theta_j := \theta_j - \frac{\alpha}{n}\sum_{i=1}^{n}\frac{\partial}{\partial\theta_j}J(\theta), \tag{8}$$

where α is denoted by a learning rate. The parameters θ_j are updated concurrently for every iterations till convergence. Our algorithm of linear regression is given in Algorithm 3.

The method of implementing the linear regression is very similar to operation in the plaintext. However, it is calculated in an encrypted state; therefore, in an encrypted domain, we can calculate all operations in gradient descent algorithm which includes multiplication, addition, and subtraction operations. We initialized parameters θ to 0 and updated our parameters using linear regression function with FHE operations.

```
    Input: data $\mathbf{X} \in \mathbb{R}^{D\times N}$, $\mathbf{Y} \in \mathbb{R}^{N}$, learning rate $\alpha$, number of iteration
    Output: Parameter $\theta \in \mathbb{R}^{D}$
(1) Initialize parameter $\theta$ to FX
(2) Gradient descent part 1: calculate partial derivative of cost function $J(\theta)$
(3) Gradient descent part 2: multiply $\alpha$ with the value of part 1
(4) Gradient descent part 3: update $\theta$ until iteration times
(5) return each of $\theta$'s
```

Algorithm 3: The algorithm of linear regression.

6.1.1. Performance Evaluation of FHE Linear Regression. We performed two experiments with varying d, the simple linear regression ($d = 1$) and the multiple linear regression ($d > 1$). We set the number of data (N), the number of dimensions (d), the length of data (l), and the number of iterations of the algorithm (p) as factors for the linear regression algorithm. Then, the number of gates (T) can be expressed as follows:

$$\begin{aligned} T(N,d,l,p) = 2Np\Big[&\left\{d\left(6l^2+4\right)T_{\mathrm{G}} + 4lT_{\mathrm{M}}\right\} \\ &+ (d+1)(5l-3)T_{\mathrm{G}}\Big] + 2(d+1)\Big\{\left(6l^2+4\right)T_{\mathrm{G}} \\ &+ 4lT_{\mathrm{M}}\Big\} + (d+1)(5l-3)T_{\mathrm{G}}. \end{aligned} \tag{9}$$

For the simple linear regression, we set the initial values to (N, d, l, p) = (10, 1, 16, 1) for the experiment. The dataset consists of a feature vector $x = [2,4,5,6,8,10,13,16,17,19]$ and a target variable $y = [5,9,12,14,15,18,24,26,30,32]$ with 10 data created artificially, and it takes 554 seconds with 0.01 running rate. The iteration proceeded 100 steps to converge $\theta = (3.404, 1.484)$ with threshold value, $\varepsilon = 0.1$.

For the multiple linear regression ($d > 1$), we set the initial values to (N, d, l, p) = (10, 2, 16, 1) for the experiment. The dataset consists of feature vectors $x_1 = [2,4,5,6,8,10,13,16,17,19]$, $x_2 = [3,5,6,7,8,11,14,15,18,20]$, and a target variable $y = [5,9,12,14,15,18,24,26,30,32]$ with 10 data created artificially, and it takes 1047 seconds with 0.01 running rate. The iteration proceeded 50 steps to converge $\theta = (-0.952, 1.094, 3.331)$ with threshold value, $\varepsilon = 0.1$.

6.2. Logistic Regression. Implementation of various algorithms such as linear regression can be easily facilitated by our FHE arithmetic operations. However, logistic regression is an algorithm that holds a nonlinear function which requires variation in the equation to be calculated. Therefore, the key point of deriving FHE logistic regression lies in designing a nonlinear sigmoid function. We initially elaborate a brief derivation and structure of FHE logistic regression followed by explaining two ways of constructing logistic function.

Given an input variable $\mathbf{x}^{(i)} \in \mathbb{R}^{d}$ and its corresponding target variable $\mathbf{y}^{(i)} \in \mathbb{Z}_2$ for $i = 1, 2, \ldots, n$, an inference on parameters of the logistic function $g(z)$ within hypotheses is defined as

$$g(z) = \frac{1}{1+e^{-z}}, \quad z = \theta^{\mathrm{T}} x^{(i)}, \tag{10}$$

where $\theta^{\mathrm{T}}\mathbf{x}^{(i)} = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_d x_d^{(i)}$ for $\mathbf{x}^{(i)} = [1, x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]^{\mathrm{T}}$, $\theta = [\theta_0, \theta_1, \theta_2, \ldots, \theta_d]^{\mathrm{T}}$, and number of features, $d+1$. We also denote $x_j^{(i)}$ as an element of a matrix in the i-th row and j-th column position.

The logistic regression uses likelihood function to make an estimate on weight θ. If we let $p(y^{(i)} = 1 \mid \mathbf{x}^{(i)}; \theta) = h_\theta(\mathbf{x}^{(i)})$ and $p(y^{(i)} = 0 \mid \mathbf{x}^{(i)}; \theta) = 1 - h_\theta(\mathbf{x}^{(i)})$, the likelihood for a single data $\mathbf{x}^{(i)}$ is. $p(y^{(i)} \mid \mathbf{x}^{(i)}; \theta) = (h_\theta(\mathbf{x}^{(i)}))^{y^{(i)}}(1 - h_\theta(\mathbf{x}^{(i)}))^{1-y^{(i)}}$.

Finally, the likelihood function for the whole data, $\{x^{(i)}\}_{i=1}^{n}$, is to multiply likelihood of each data. Next, log operation is performed to enumerate log likelihoods in a linear combination as the follows:

$$L(\theta) = \sum_{i=1}^{n} y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right). \tag{11}$$

In order to maximize the likelihood, $L(\theta)$, we chose to perform gradient descent algorithm that iteratively updates cost function, $J(\theta)$, where $J(\theta) = -L(\theta)$. Therefore, θ is updated with the following equation:

$$\theta_j := \theta_j - \frac{\alpha}{n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}. \tag{12}$$

Existing literature [18] designed a nonlinear logistic function by two approximation techniques, namely, the Taylor series method and least square approximation. In this paper, we show feasibility of constructing two different approximation techniques based on our proposed bitwise FHE operations to perform the logistic regression.

6.2.1. Taylor Series Method. It is well-known that Taylor expansion enables a differentiable real-valued function $f(x)$ to be expanded in a series at $x = a$ such that $f(x) = \sum_{r=1}^{\infty} f^{(r)}(a)/r!(x-a)^r = f(a) + (f'(a)/1!)(x-a) + (f''(a)/2!)(x-a)^2 + \cdots$ where $f^{(r)}$ is denoted by r-th derivative of f.

Bos et al. applied Taylor series expansion to logistic function which facilitates calculation of the nonlinear function since the altered equation incorporates only the four fundamental operations [18]. Therefore, Taylor series polynomial of degree 9 for sigmoid function can be derived as

$$g(x) = \frac{1}{1+e^{-x}} \approx \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 - \frac{17}{80640}x^7 + \frac{31}{1451520}x^9. \quad (13)$$

Using our basic bitwise FHE operations that are presented in the previous section, we can construct approximate logistic function by Algorithm 4. In addition, we refer c_i to coefficients of $g(x)$ where $c_0 = 1/2$, $c_1 = 1/4 \cdots$, $c_5 = 31/1451520$.

Figure 7 illustrates approximated logistic function with respect to Taylor series expansion. Our approach guarantees a boundary of $(-1, 1)$ while $(-2, 2)$ for the existing literature [18]. This is due to the 4th and 5th coefficients that are 0 for the length of the input designated by 32 bit. This can be solved by assigning larger length to represent the coefficient numbers.

6.2.2. Least Square Approximation. Kim and Cheon et al. proposed a least square polynomial that broadens bounded domain of Taylor series expansion to $(-8, 8)$ [19, 20]. The underlying principle is to derive a function $g(x)$ that minimizes mean squared error (MSE) such that $1/|I| \int_I (g(x) - f(x))^2 dx$ where $|I|$ is denoted by the length of an interval.

We omit an algorithm for implementing the least square approximation with respect to our scheme since the algorithm follows a similar procedure as in Algorithm 4. The visualized comparison of the real sigmoid function with our approach and that of the existing literature [18] can be seen in Figure 8 to verify that our FHE scheme can approximate the desired function equal to the current literature.

6.2.3. FHE Gradient Descent Algorithm. When the logistic function is designed either by the Taylor series or the least square approximation technique, we are able to perform the gradient descent algorithm for parameter estimate. The process of logistic regression is indicated in Algorithm 5.

6.2.4. Performance Evaluation of the FHE Logistic Regression. We implemented logistic regression with two of the strategies mentioned previously. From Algorithm 4, we claim that number of data (N), length of data (l), dimension (d), and iteration (p) are the principal factors of time complexity (T) for both methods. We deliver their time performances in a precise manner, where T_{Taylor} and T_{ls} are time complexity of the Taylor series and least square approximation, respectively:

$$\begin{aligned} T_{\text{taylor}}(N, l, d, p) &= Ndp\left[13\left\{\left(6l^2 + 4\right)T_{\text{G}} + 4lT_{\text{M}}\right\} + 6(5l-3)T_{\text{G}}\right] + 6(5l-3)T_{\text{G}}dp, \\ T_{\text{ls}}(N, l, d, p) &= Ndp\left[10\left\{\left(6l^2 + 4\right)T_{\text{G}} + 4lT_{\text{M}}\right\} + 5(5l-3)T_{\text{G}}\right] + 6(5l-3)T_{\text{G}}dp. \end{aligned} \quad (14)$$

Since the time for experiment requires fairly significant amount of time, we set number of data, dimension, and iteration to be 10, 2, and 1, respectively. The summary of time performance with respect to 16 bit is elaborated in Table 5.

6.3. kNN Classifier. The bitwise FHE method of implementing the kNN algorithm in Algorithm 6 is almost similar to that of the plaintext, except the sorting operation which is described in the next section. The conventional kNN algorithm uses Euclidean distance between data, but our algorithm replaced the distance as the sum of the absolute value for speed efficiency. Also, when sorting the calculated distances, we searched for only the k smallest values to reduce the computation time. As shown in Algorithm 6, we need to design two additional homomorphic operations for the homomorphic kNN classifier: sort of Algorithm 7 and conditional swap of Algorithm 8.

When sorting is completed, we check the labels of the nearest k data and output the major labels. Since the label is also encrypted, it is not possible to know which label is the most major.In order to attain the most frequently used label, we first counted number of data with the same label. Since the counting numbers are encrypted, we perform equivalent compare operation of a label to the other labels. Lastly, we add all the output numbers and sort out in descending order to pick the largest number, which is our desired label. Algorithm 9 represents the pseudocode that finds the most major label among the labels of k-nearest data in our kNN algorithm.

6.3.1. Sorting for kNN Algorithm. The kNN algorithm on encrypted domain requires sorting algorithm to find the nearest neighbors, so we design a new sort algorithm for ciphertext. Algorithm 7 represents the pseudocode to sort the numbers in arr[n] by the selection sort algorithm.

A swap operation that simply exchanges a location in a ciphertext should only change its position as in plaintext, but to apply the selection sort algorithm to ciphertext, we must decide whether to relocate it through a large or small comparison. So, we have to input the factor to determine whether to swap or not, and we call this swap operation conditional swap.

6.3.2. Conditional Swap. Conditional swap operation runs swapping if a determining factor is $E[1]$. Otherwise, data are not swapped. In the selection sort algorithm, if arr[i] is bigger than arr[j], it has to be swapped. Therefore, it outputs $E[1]$ through a large comparison operation and puts it into the factor to decide whether to swap or not. If arr[i] is less than or equal to arr[j], swap will not occur because it outputs $E[0]$ through the large comparison operation. Algorithm 8 represents the conditional swap pseudocode that takes this situation into consideration.

When S is $E[1]$, arr[i] = NS arr[i] + S arr[j] is arr[j] and arr[j] = S arr[i] + NS arr[j] is arr[i]. Thus, swap operation has occurred. The other way, in case $S = E[0]$, arr[i] = NS arr[i] + S arr[j] is arr[i] and arr[j] = S arr[i] + NS arr[j] is arr[j]. Thus, swap operation has not occurred.

```
    Input: a training data $\mathbf{x}^{(i)}$
    Output: logistic value of $\mathbf{x}^{(i)}$ w.r.t the Taylor expansion method
(1) Convert coefficient $c_i$ into arrays
(2) Construct power series of $x$ to 9th power
(3) Multiply $c_i$ with corresponding power of $x$
(4) Add all the derived terms in step 3
```

Algorithm 4: FHE sigmoid function by Taylor expansion.

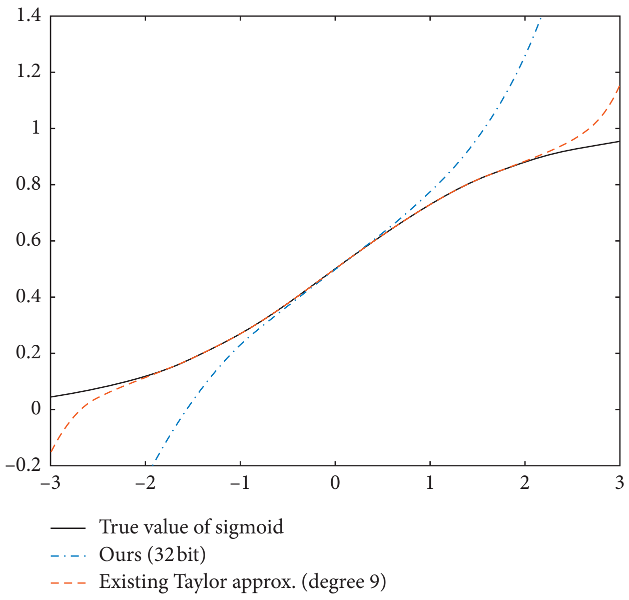


Figure 7: Comparison of real sigmoid with our approach and Taylor series approximation from existing literature [18].

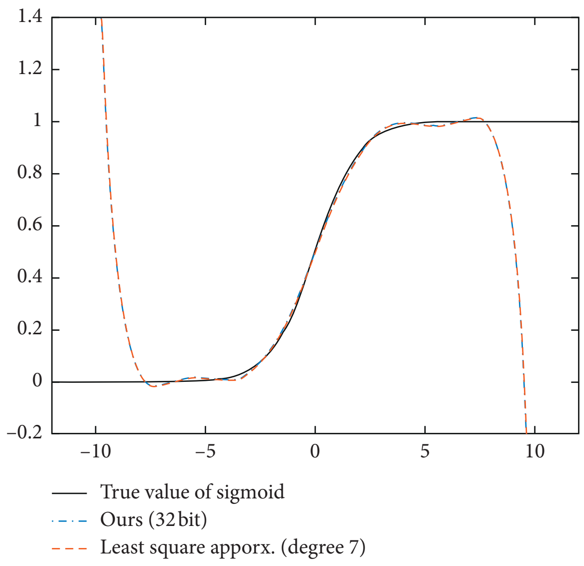


Figure 8: Comparison of real sigmoid with our approach and least square approximation.

6.3.3. Performance Evaluation of the FHE kNN Classifier. We set the number of data (N), the dimension of data (d), the length of data (l), the number of near neighbors (k), and the length of label (L) as factors of the kNN algorithm. Then, the time complexity of kNN algorithm (T) can be expressed as follows:

$$\begin{aligned} T(N,d,l,k,L) = N[&\{d(12l-6)-5l+3\}T_G \\ &+ dT_M] + \frac{k^2+k(2N-3)}{2}\{lT_G \\ &+ (3l+2L)T_M\} \\ &+ \frac{(k-1)(k-2)}{2}(7L-4)T_G \\ &+ (k-1)(LT_G+5LT_M). \end{aligned} \tag{15}$$

We set the initial values to $(N, d, l, k, L) = (64, 1, 10, 3, 1)$ for the experiment. When conducting experiment with the initial value, it took 226 seconds. Then, we performed the experiment by changing the value of each factor one by one. As a result, because the algorithm consists solely of linear operations except k, we confirmed that the speed of the algorithm is almost proportional to the value of each factors.

6.4. k-Means Algorithm for Image Segmentation. We also performed gray color image segmentation using the k-means algorithm. The target image for the homomorphic segmentation has the 8-bit gray color of each pixel in the image, and the k-means algorithm is used to input the encrypted color value of all the pixels. In order to do this, the cloud server first obtains encrypted values of the pixels at N random locations rather than all encrypted pixels for efficient computation. Afterwards, the k-means algorithm is applied to partition N encrypted pixels into k clusters. As a result, the cloud server calculates the representative values of k clusters in a homomorphic way. After deciphering the representative values in the client's side, the colors of all the pixels in the image are compared with the representative values, and image segmentation is performed by replacing the color with the representative value of the near cluster. Our algorithm of k-means is given in Algorithm 10; we performed the algorithm by expanding the total data size to 10 bits considering 8-bit original data, the sign bit, and addition operation.

In general, use the Euclidean distance when calculating the distance between two points. In this experiment, however, another method can be used because the dimension of

```
     Input: training data $\mathbf{X}, \mathbf{Y}$
     Output: parameter $\theta$
(1)  Set parameter $\theta$ to $\mathbf{0}$
(2)  Assign learning rate $\alpha$ and iteration number $p$, respectively
(3)  Calculate partial derivative of cost function $J(\theta)$ for the training data $\mathbf{X}, \mathbf{Y}$
(4)  Multiply $\alpha/n$ by the previous outcome
(5)  Update $\theta$ by the result of step 4
(6)  Repeat steps 3 to 5 for $p$ times to obtain $\theta$
```

Algorithm 5: The algorithm of logistic regression.

Table 5: Execution time of logistic regression w.r.t two different strategies (16-bit inputs).

Strategy	Taylor expansion series	Least square approximation
Time (s)	12961	11315

```
     Input: training data ($\mathbf{X}$, $\mathbf{Y}$, $\mathbf{l}$), test data ($\mathbf{x}$, $\mathbf{y}$), and the number of neighbors, $k$
     Output: test label $l_t$
(1)  Calculate distance with training data ($\mathbf{X}$, $\mathbf{Y}$) and test data ($\mathbf{x}$, $\mathbf{y}$) with absolute value operation.
(2)  Sort the smallest $k$ distance using conditional swap operation on selection sort algorithm.
(3)  Output most major label among the labels of nearest $k$ data.
```

Algorithm 6: The algorithm of kNN classification.

```
     Input: $\text{arr}[n] = [a_1, a_2, \ldots, a_n]$
     Output: SORT($\text{arr}[n]$)
(1)  for $i = 1 : (n-1)$ do
(2)      for $j = (i+1) : (n-1)$ do
(3)          COND_SWAP($\text{arr}[i], \text{arr}[j], S$)
(4)      end for
(5)  end for
(6)  return $\text{arr}[n]$
```

Algorithm 7: Pseudocode of sorting.

```
     Input: $\text{arr}[i]$, $\text{arr}[j]$, $S$
     Output: COND_SWAP($\text{arr}[i], \text{arr}[j], S$)
(1)  $S = $ L_COMP($\text{arr}[i], \text{arr}[j]$): large comparison
(2)  $NS = S$
(3)  $\text{arr}[i] = NS \wedge \text{arr}[i] + S \wedge \text{arr}[j]$
(4)  $\text{arr}[j] = S \wedge \text{arr}[i] + NS \wedge \text{arr}[j]$
(5)  return $\text{arr}[i], \text{arr}[j]$
```

Algorithm 8: Pseudocode of conditional swap.

the data is one-dimensional. After calculating the center point of each representative value of each cluster, labeling is performed without calculating the distance through comparison of the data with the values.

Since the values of the data are not known in the encrypted state, when the representative values of the cluster are given, it is not known which value is closest to the representative value. However, we can set the label to distinguish the nearest value from the representative value of each cluster. Let $E[1]$ and $E[0]$ denote each label. Through an AND operation of each data and its label, we can divide all data into **0** of which all bits are set to $E[0]$ and non-**0** values (if a data's label is $E[0]$, the result of AND operation is **0** value; otherwise, the result is non-**0** value). Now, we can

```
     **Input:** $l_1, l_2, \cdots, l_k$
     **Output:** $l_p$
(1)  for $i = 1 : (k-1)$ do
(2)    $s_i = 1$
(3)    for $j = (i+1) : k$ do
(4)      if $l_i = l_j$ then
(5)        $c = 1$
(6)      else
(7)        $c = 0$
(8)      end if
(9)      $s_i = s_i + c$
(10)   end for
(11) end for
(12) $p = \arg_i \max(s_i)$ for $i = 1 : k$
(13) return $l_p$
```

Algorithm 9: The pseudocode to find a label with majority.

```
    Input: data $\mathbf{X}$, the number of neighbors $k$, and the initial value of clusters $u_k$
    Output: labeled data $(\mathbf{X}, \mathbf{l})$
(1) Obtain the distance between each cluster $u_k$ and the data $\mathbf{X}$.
(2) For each data, label closest clusters.
(3) Initialize the cluster $u_k$ by averaging the data with the same cluster value.
(4) Repeat steps 1 to 3 to obtain converged clusters and return the labeled data.
```

Algorithm 10: The algorithm of k-means.

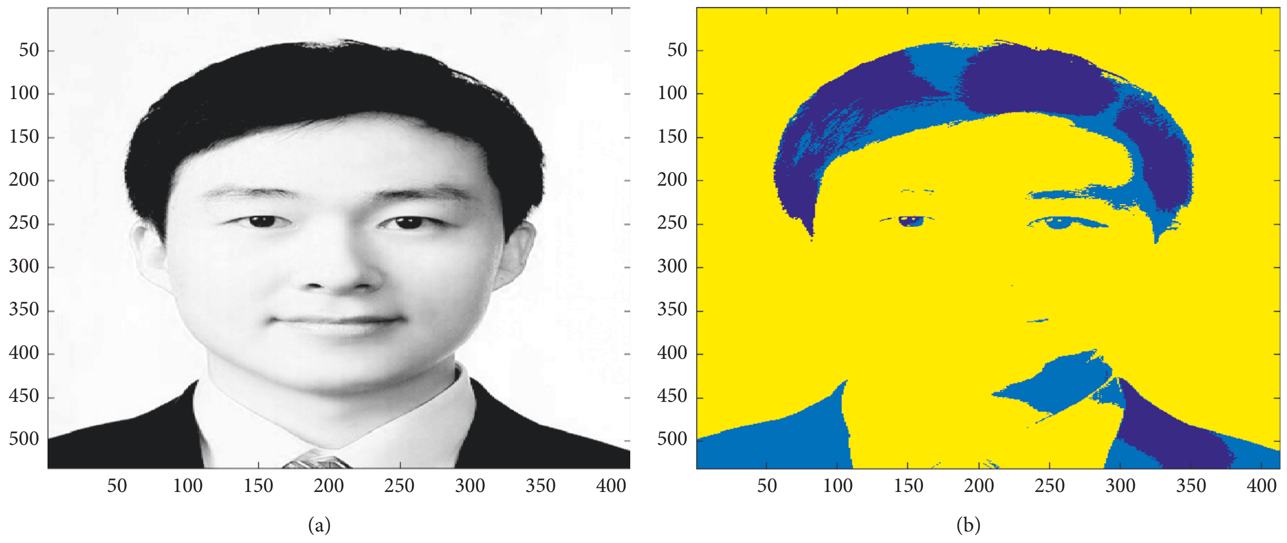


Figure 9: (a) An original image and (b) its segmented image on the encrypted domain.

obtain the average of the clusters by dividing sum of data with sum of labels.

6.4.1. Performance Evaluation of the FHE k-Means Clustering Algorithm. We set the number of data (N), the length of data (l), the number of clusters (k), and the number of iterations of the algorithm (p) as factors for the k-means algorithm. Then, the number of gates (T) can be expressed as follows:

$$\begin{aligned} T(N,l,k,p) = p[&(k-1)(5l-3)T_G \\ &+ N(k-1)(T_G + T_M) \\ &+ k\{NlT_G + (N-1)(5l + 5\log N - 6)T_G\} \\ &+ k\{(8l^2 - 4l + 4)T_G + (l^2 + 2l)T_M\} \\ &+ NlkT_G]. \end{aligned} \tag{16}$$

The algorithm took 148 seconds given the initial value, $(N, l, k, p) = (64, 10, 3, 1)$. The experiment was set up with an input of an image in Figure 9(a), where the parameters are given as 64, 10, 3, and 10. The representative value for each cluster is recorded as 23, 56, and 170, respectively. The experiment took approximately 1,500 seconds, and the result of segmentation can be checked in Figure 9(b).

7. Discussion

In this section, we describe the limitation of our proposed approach in usage. Our proposed approach has a concern: it has extremely slow computation with large memory space.

Currently, it is true that bit-based schemes are inefficient in terms of speed and memory compared to integer-based schemes. However, integer-based schemes have a fatal disadvantage that their possible operations are limited and can only be used for specific algorithms. This is a fundamental problem and hard to improve. On the other hand, the speed of computation, which is a disadvantage of bit-based schemes, can be improved more flexibly.

Our current approach is not optimized yet, so each operation on encrypted domain is extremely time consuming. However, this problem may be addressed by accelerating the computation with a lot of state-of-the-art techniques. For instance, the atomic operations can be implemented in a hardware level rather than in a software level. FPGA and ASIC would be the good candidates for the implementation. Additionally, we can reduce the computation time by optimizing the logic and programming codes in a software level. We can also save the computation time using a graphical processing unit (GPU) and parallel computing scheme.

8. Conclusion

In this paper, we have proposed basic homomorphic arithmetic operations using bitwise homomorphic gates. We applied these bitwise homomorphic operations to several well-known data mining techniques: the linear regression, logistic regression, k-NN classifier, and k-means clustering. To implement the algorithms, we introduced advanced bitwise operations such as sorting and conditional swap, which are specific to bitwise homomorphic operations. With our proposed bitwise homomorphic atomic and additional operations, even data scientists without any knowledge of FHE can easily analyze and process data on encrypted domain.

Data Availability

The training data and image data used to support the findings of this study have not been made available because it is artificially created and also small enough so that the reader can easily create it.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This study was supported by the Institute for Information and Communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (no. 2017-0-00545).

References

[1] M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, “Fully homomorphic encryption over the integers,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 24–43, Springer, Berlin, Germany, 2010.

[2] Z. Brakerski and V. Vaikuntanathan, “Efficient fully homomorphic encryption from (standard) LWE,” *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[3] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proceedings of the STOC*, vol. 9, pp. 169–178, Washington, DC, USA, May 2009.

[4] D. Li, C. Liu, and W. Gan, “A new cognitive model: cloud model,” *International Journal of Intelligent Systems*, vol. 24, no. 3, pp. 357–375, 2009.

[5] T. Veugen, “Encrypted integer division,” in *Proceedings of the IEEE International Workshop on Information Forensics and Security, WIFS 2010*, pp. 1–6, IEEE, Seattle, WA, USA, December 2010.

[6] T. Graepel, K. Lauter, and M. Naehrig, “Ml confidential: machine learning on encrypted data,” in *Proceedings of the International Conference on Information Security And Cryptology*, pp. 1–21, Springer, Seoul, Korea, November 2012.

[7] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, “Privacy-preserving ridge regression on hundreds of millions of records,” in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pp. 334–348, IEEE, San Francisco, CA, USA, 2013.

[8] T. Veugen, “Encrypted integer division and secure comparison,” *International Journal of Applied Cryptography*, vol. 3, no. 2, pp. 166–180, 2014.

[9] L. J. Aslett, P. M. Esperança, and C. C. Holmes, “Encrypted statistical machine learning: new privacy preserving methods,” 2015, http://arxiv.org/abs/1508.06845.

[10] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, “Machine learning classification over encrypted data,” in *Proceedings of the NDSS*, San Diego, CA, USA, 2015.

[11] R. Gilad-Bachrach, N. Dowlin, K. Laine et al., “Applying neural networks to encrypted data with high throughput and accuracy,” in *Proceedings of the International Conference on Machine Learning*, pp. 201–210, New York City, NY, USA, June 2016.

[12] R. L. Rivest, L. Adleman, M. L. Dertouzos et al., “On data banks and privacy homomorphisms,” *Foundations of Secure Computation*, vol. 4, no. 11, pp. 169–180, 1978.

[13] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *Proceedings of the Advances in Cryptology—EUROCRYPT’ 99*, vol. 99, pp. 223–238, Springer, Espoo, Finland, May 1999.

[14] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “TFHE: Fast Fully homomorphic encryption library over the torus,” *Journal of Cryptology*, pp. 1–58, 2017.

[15] L. Ducas and D. Micciancio, “FHEW: bootstrapping homomorphic encryption in less than a second,” *Eurocrypt (1)*, vol. 9056, pp. 617–640, 2015.

[16] W. Lu, S. Kawasaki, and J. Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data," in *Proceedings of the 2017 Network and Distributed System Security Symposium*, p. 1163, San Diego, CA, USA, 2017.

[17] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Faster packed homomorphic operations and efficient circuit bootstrapping for tfhe," in *Proceedings of the International Conference on the Theory And Application of Cryptology And Information Security*, pp. 377–408, Springer, Hanoi, Vietnam, December 2017.

[18] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of Biomedical Informatics*, vol. 50, pp. 234–243, 2014.

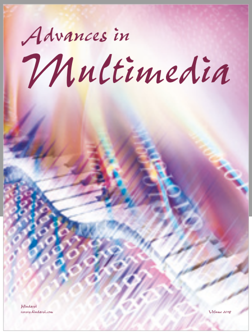
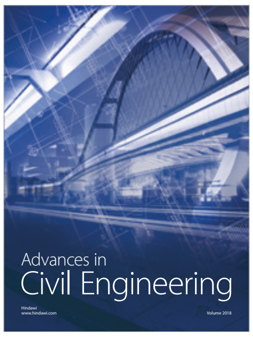
[19] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, "Secure logistic regression based on homomorphic encryption: design and evaluation," *JMIR Medical Informatics*, vol. 6, no. 2, 2018.

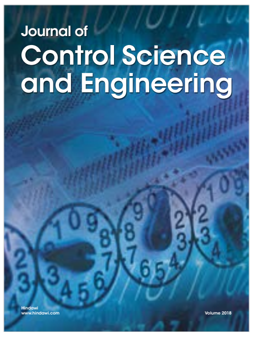
[20] J. H. Cheon, D. Kim, Y. Kim, and Y. Song, "Ensemble method for privacy-preserving logistic regression based on homomorphic encryption," *IEEE Access*, vol. 6, pp. 46938–46948, 2018.

Journal of
Engineering

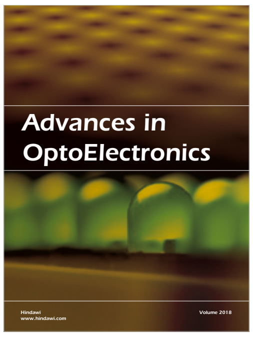
The Scientific World Journal

International Journal of
Rotating Machinery

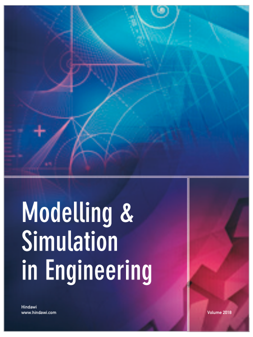
Journal of
Sensors

Advances in
Multimedia

Advances in
Civil Engineering

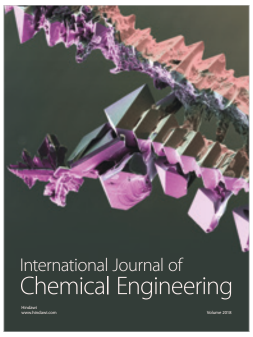
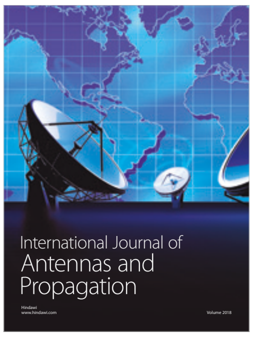

Journal of
Control Science and Engineering

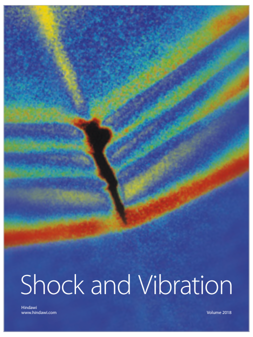
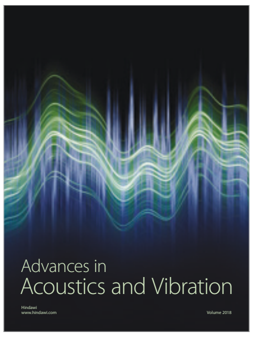
Journal of
Robotics

Journal of
Electrical and Computer Engineering

Advances in
OptoElectronics

VLSI Design

International Journal of
Navigation and Observation

Modelling & Simulation in Engineering

International Journal of
Aerospace Engineering

International Journal of
Chemical Engineering

International Journal of
Antennas and Propagation

Active and Passive Electronic Components

Shock and Vibration

Advances in
Acoustics and Vibration