An Efficient Update Algorithm for Mutable Order-Preserving Encryption

JIHYE YANG AND KEE SUNG KIM
Department of Computer Software, Daegu Catholic University, Daegu 38430, South Korea
Corresponding author: Kee Sung Kim (kee21@cu.ac.kr)
This work was supported by the Research Grants from Daegu Catholic University in 2022.

ABSTRACT
Order-preserving encryption (OPE) produces ciphertexts that preserve the numerical order of plaintexts. Many researchers have focused on designing ideally-secure OPE schemes. This security notion, known as IND-OCPA, has been shown to be achievable by the mutability of ciphertexts and interaction between a database server and a client. This implies that the ciphertexts stored on the server can be updated. Unfortunately, existing update algorithms of mutable OPE schemes are designed to generate ciphertexts uniformly regardless of the distribution of the plaintexts. This leads to inefficiency that requires frequent ciphertext updates for a certain input data pattern (e.g., sequential data). In this paper, we propose a more efficient ciphertext update algorithm that is suitable for mutable OPE schemes. This algorithm makes it possible to reduce the number of updates by considering the input pattern of encrypted data without loss of security. Our experimental results show that, when applied to existing mutable OPE schemes, our update algorithm delivers significantly improved performance on a variety of datasets.

INDEX TERMS
Order-preserving encryption, ciphertext update, cloud security, database encryption.

I. INTRODUCTION
When utilizing a database server operated by an untrusted third party such as a cloud server, protecting the confidentiality of the sensitive data of the client is of the utmost importance. The simplest way to protect personal information is to encrypt the data before outsourcing them to the server. However, the application of general encryption schemes, such as AES, to a database, renders the database unable to even support simple range queries. Order-preserving encryption (OPE) enables various operations such as min, max, count, and sort on an encrypted database, because the ciphertexts of OPE preserve the numerical ordering of the corresponding plaintexts.

In a situation where the numerical ordering information of a plaintext is exposed, the previous works have been focused on designing schemes that achieve the best possible security. According to this security notion, known as IND-OCPA, ciphertexts reveal no additional information except for the order of the underlying plaintexts. This has also been shown to be achievable by assuming that the OPE scheme is interactive and mutable, that is, the stored ciphertexts are updated under a certain condition. The efficiency and practicality of the mutable OPE schemes are thus obviously closely related to the frequency of updates. Because these updates are time and resource intensive with respect to re-encrypting and sending the data to the server, the shorter the update cycle, the greater the burden on the database system. However, despite its importance, research on the efficient construction of update algorithms has received relatively less attention compared with OPE encryption.

Existing update algorithms uniformly re-encrypt stored ciphertexts regardless of the distribution of input data. However, with the exception of uniform distributions, this approach is inefficient when the input data follow a specific (unique) distribution. For example, when encrypting sequentially increasing data, instead of uniformly re-encrypting ciphertexts, densely re-encrypting them with a small values could reduce the number of updates. Therefore, a new ciphertext update algorithm that updates ciphertexts by considering the input pattern of the encrypted data is necessary. Nonetheless, it should be noted that, to ensure that the security of
OPE remains unaffected, only the ordering information of plaintexts should be used, not the actual distribution.

A. RELATED WORK

In 2004, Agrawal et al. [1] introduced the first concept of OPE. Subsequently, Boldyreva et al. [2] provided the ideal security notion of OPE, which they referred to as indistinguishability under ordered chosen-plaintext attack (IND-OCPPA). They also demonstrated that it is impossible to achieve IND-OCPPA unless the ciphertext space is exponentially larger than the plaintext space. In a subsequent paper [3] pointed out that the ciphertexts of [2] leak approximately the first half of the bits of the underlying plaintexts. Despite many studies on OPE schemes [4], [5], [6], [7], [8], none of the previous OPE schemes have been able to achieve the ideal security.

1) (MUTABLE OPE)

In 2013, Popa et al. [9] presented the first IND-OCPPA secure OPE scheme of which the construction is both mutable and interactive. Their scheme requires multiple rounds to encrypt data; thus, the communication cost is high. Similar to [9] and Kerschbaum and Schröpfer [10] presented a stateful OPE with IND-OCPPA security, in which an ordered list consisting of plaintext-ciphertext pairs is stored on the client side to reduce the communication cost. All of the OPE schemes mentioned thus far are deterministic and thus expose the frequency information of the plaintexts. To overcome these limitations, Kerschbaum [11] proposed frequency-hiding order-preserving encryption (FH-OPE) and introduced a stronger security notion termed indistinguishability under frequency-analyzing ordered chosen plaintext attack (IND-FA-OCPPA). Subsequently, Maffei et al. [12] proved that the security model of [11] is inaccurate and proposed a revised model. They also presented a new FH-OPE scheme that is suitable for their improved security model; however, the client must maintain the order information of all encrypted plaintexts, thus causing a relatively high overhead. In 2021, Yang and Kim [13] proposed FH-OPE with improved IND-FA-OCPPA security by replacing the inefficient element of [12] with a combination of a random value and hash function. They also showed that the existing ciphertext update algorithm of the previous mutable OPE is designed by assuming that distinct plaintexts are encrypted, thereby resulting in frequent ciphertext updates when duplicate plaintexts are encrypted. To alleviate this problem, they suggested an improved update algorithm that is suitable for duplicate plaintexts.

2) (USE CASE)

Some of OPE schemes have been used in real-world applications such as CryptDB [14] and Monomi [15]. Both execute SQL queries over encrypted data using a collection of efficient SQL-aware encryption schemes (e.g., OPE for range queries and Paillier cryptosystem for arithmetic operations). OPE has also been used in private machine learning

3) (ORE)

The generalized version of OPE, order-revealing encryption (ORE), provides a public function that takes two encrypted plaintexts as inputs and outputs their numerical ordering. In 2015, Boneh et al. [17] provided the first ideally-secure ORE using multilinear maps. Thereafter, Chenette et al. [18] designed a practical ORE with the leakage of the most significant difference bits of two plaintexts. They also presented a simulation-based security definition to more accurately quantify the information leakage in ORE schemes. Lewi and Wu [19] provided a new ORE scheme that leaked less than [18], but the scheme still did not provide ideal security. In recent years, practical ORE schemes [20], [21], [22] were proposed to reduce information leakage and generate shorter ciphertexts compared to existing schemes.

B. OUR CONTRIBUTIONS

Existing update methods for mutable OPE schemes are designed to produce ciphertexts uniformly regardless of the distribution of the plaintexts. Because this design is suitable for a uniform distribution, it incurs frequent ciphertext updates for specific plaintext distributions (e.g., a normal distribution). This encouraged us to propose a more efficient ciphertext update algorithm for mutable OPE schemes.

Our algorithm does not require any additional storage or information and uses only information that inevitably leaks from the ciphertexts encrypted with any OPE scheme. Our experimental results demonstrate that our algorithm is more efficient and practical because it reduces the overall number of updates for various datasets.

C. OUTLINE

The rest of this paper is organized as follows. In Section II, we review the formal notion of (stateful) OPE and its security model. In Section III, we review the previous mutable OPE schemes and ciphertext update algorithms described in these schemes. Section IV presents our new ciphertext update algorithm and Section V evaluates the performance of our algorithm on various datasets. Finally, we conclude this paper in Section VI.
II. PRELIMINARIES

In this section, we briefly review order-preserving encryption and its security model.

A. FORMAL NOTION OF OPE

**Definition 1 (Order-Preserving Encryption):** A (stateful) OPE scheme consists of the following three algorithms (Setup, Encryption, Decryption):

- \( S \leftarrow \text{Setup}(1^n) \): The setup algorithm takes as input a security parameter \( \lambda \) and initializes a state \( S \).
- \( (y, S') \leftarrow \text{Encryption}(x, S) \): The encryption algorithm takes as input a plaintext \( x \) and a state \( S \). It outputs a ciphertext \( y \) and updates the state \( S \) to \( S' \).
- \( x \leftarrow \text{Decryption}(y, S) \): The decryption algorithm takes as input a ciphertext \( y \) and a state \( S \). It outputs a plaintext \( x \).

**Definition 2 (Order-Preserving):** An OPE scheme is order-preserving if for any two ciphertexts \( y_1 \) and \( y_2 \) with corresponding plaintexts \( x_1 \) and \( x_2 \), we have \( y_1 \geq y_2 \Rightarrow x_1 \geq x_2 \).

B. SECURITY DEFINITIONS

IND-OCPA security means that an efficient (polynomially bounded) adversary can distinguish between the ciphertexts of two equally ordered plaintext sequences does not exist. Specifically, IND-OCPA is defined as the following game. The security game Game_{\text{IND-OCPA}}(\lambda) between adversary \( A \) and challenger \( C \) for security parameter \( \lambda \) proceeds as follows:

1) The adversary \( A \) prepares two plaintext sequences \((x_0^1, x_1^1), \ldots, (x_0^n, x_1^n)\) where \( x_0^i < x_1^i \iff x_1^i < x_1^j\), \( 1 \leq i, j \leq n \) and sends them to the challenger \( C \).
2) The challenger \( C \) randomly chooses \( b \leftarrow \{0, 1\} \), executes Setup(\( 1^\lambda \)), and runs \((y_i, S_i) \leftarrow \text{Encryption}(x_i, b_i, S_{i-1})\) for all \( 1 \leq i \leq n \). Then, the challenger \( C \) sends \( y_{1 \leq i \leq n, b} \) to the adversary \( A \).
3) The adversary \( A \) guesses which sequence is encrypted and outputs \( b' \) as a guess of \( b \).

**Definition 3 (IND-OCPA):** An OPE scheme has IND-OCPA security if, for any PPT adversary \( A \), the probability of \( b' = b \) is \( 1/2 + \text{negl}(\lambda) \), where \( \text{negl}(\lambda) \) is a negligible function in the security parameter \( \lambda \).

**Definition 4 (Randomized Order):** A randomized order \( \Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\} \) for not necessarily distinct plaintexts in sequence \( X = \{x_1, x_2, \ldots, x_n\} \) indicates a possible permutation of \( \{1, 2, \ldots, n\} \) that is ordered according to a sort of \( X \). A randomized order \( \Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\} \), where \( 1 \leq \gamma_i \leq n \) and \( i \neq j \Rightarrow \gamma_i \neq \gamma_j \) for all \( i, j \), of sequence \( X \) holds that:

\[ \forall i, j, (x_i > x_j \Rightarrow \gamma_i > \gamma_j) \land (\gamma_1 > \gamma_2 \Rightarrow x_1 > x_2) \]

For example, a plaintext sequence \( X = \{1, 1, 2, 2\} \) can be represented by \( \Gamma_1 = \{1, 2, 3, 4\} \), \( \Gamma_2 = \{1, 2, 3, 4\} \), \( \Gamma_3 = \{1, 2, 3, 4\} \), or \( \Gamma_4 = \{1, 2, 4, 3\} \). The common randomized order \( \Gamma \) of \( X_0 \) and \( X_1 \) represents the intersection element of two randomized order sets of \( X_0 \) and \( X_1 \). For \( X_0 = \{1, 2, 3\} \) and \( X_1 = \{2, 2, 3, 3\} \), the common randomized order \( \Gamma \) can be \( \{1, 2, 3\} \) or \( \{1, 2, 4, 3\} \).

IND-FA-OCPA security is strictly stronger than that of IND-OCPA and it is defined as the following game. The security game Game_{\text{IND-FA-OCPA}}(\lambda) between adversary \( A \) and challenger \( C \) for security parameter \( \lambda \) proceeds as follows:

1) The adversary \( A \) prepares two plaintext sequences \((x_0^1, x_1^1), \ldots, (x_0^n, x_1^n)\). These sequences have at least one common randomized order \( \Gamma \). He sends them to the challenger \( C \).
2) The challenger \( C \) randomly chooses \( b \leftarrow \{0, 1\} \), executes Setup(\( 1^\lambda \)), and runs \((y_i, S_i) \leftarrow \text{Encryption}(x_i, b_i, S_{i-1})\) for all \( 1 \leq i \leq n \) based on \( \Gamma \). Then, the challenger \( C \) sends \( y_{1 \leq i \leq n, b} \) to the adversary \( A \).
3) The adversary \( A \) guesses which sequence is encrypted and outputs \( b' \) as a guess of \( b \).

**Definition 5 (IND-FA-OCPA):** A FH-OPE scheme has IND-FA-OCPA security if, for any PPT adversary \( A \), the probability of \( b' = b \) is \( 1/2 + \text{negl}(\lambda) \), where \( \text{negl}(\lambda) \) is a negligible function in the security parameter \( \lambda \).

III. REVIEW OF PREVIOUS OPE SCHEMES AND THEIR UPDATE ALGORITHMS

In this section, we review ideally-secure mutable OPE schemes [10], [11], [12], [13] and their ciphertext update algorithms. Let \( n \) be the number of plaintexts to be encrypted. We define the plaintext space \( \{1, N\} \) and ciphertext space \( \{-1, M\} \). Let \( x_1, \ldots, x_i, \ldots, x_n \) be the plaintext sequence, such that \( 0 \leq x_i < N \). Let \( y_1, \ldots, y_i, \ldots, y_n \) be the corresponding ciphertext sequence, such that \( 0 \leq y_i < M \). The client stores an ordered list of plaintext-ciphertext pairs \((x_i, y_i)\) as a state and sends the ciphertext \( y_i \) to the database server. State list \( S \) is initialized to \( \{(−1, −1), (N, M)\} \). At a high level, the encryption schemes in [10], [11], [12], and [13] can be explained as a work of inserting plaintext into a binary search tree. In other words, the client may store state \( S \) using a data structure such as a binary search tree. We denote the binary search tree as the set \( T \) of nodes \( \{r\} \). For node \( r \in T \), we use \( t.left \) and \( t.right \) to represent the child nodes of \( r \) on the left and right, respectively. Kerschbaum [11] presented an efficient compression techniques to reduce the amount of information stored on the client, which in certain cases can lead to compression ratios of 15.

A. OPE SCHEME OF KERSCHBAUM et al.

The main idea of [10] to achieve IND-OCPA security was to generate the ciphertext as a median value of the possible ciphertext space corresponding to the input plaintext. For example, assuming that five plaintexts \( 3, 7, 2, 5, \) and \( 10 \) are sequentially encrypted in the ciphertext space \( M = 128 \), the first plaintext \( 3 \) is encrypted as \( 64 \), which is the median value of the possible ciphertext space \( \{-1, 128\} \). As shown in Fig. 1, if encryption proceeds in this way, ciphertexts of \( 96, 32, 80, \) and \( 112 \) are sequentially generated. If the ciphertext space is no longer halved, re-encryption is executed according to the
update algorithm. The update algorithm used in [10] updates all ciphertexts produced so far. It operates by re-encrypting all plaintexts uniformly by calling their own encryption function, as shown in Algorithm 1. Specifically, it sorts the plaintext sequence $X = \{x_1, \ldots, x_n\}$ in ascending order, re-encrypts the middle value of $X$, and recursively does the same for left and right halves of $X$. The client obtains updated plaintext-ciphertext pairs by executing the ciphertext update algorithm. The update process is completed by sending and reflecting updated information to the server.

**Algorithm 1: Original Update [10]**

**Input:** sorted $X = \{x_1, \ldots, x_n\}$ in ascending order  
**Output:** $y$  
**State:** $S$

**Initialization:** $S = \{((-1, -1), (N, M))\}$  
$n = \text{length of } X$

if $n = 1$ then

```
Output $y_{\lceil \frac{n}{2} \rceil + 1} \leftarrow \text{Encryption}(x_{\lceil \frac{n}{2} \rceil + 1}, S)$
```

if $n = 2$ then

```
Output $y_{\lceil \frac{n}{2} \rceil + 1} \leftarrow \text{Encryption}(x_{\lceil \frac{n}{2} \rceil + 1}, S)$
Output $y_{\lceil \frac{n}{2} \rceil} \leftarrow \text{Encryption}(x_{\lceil \frac{n}{2} \rceil}, S)$
```

if $n \geq 3$ then

```
Output $y_{\lceil \frac{n}{2} \rceil + 1} \leftarrow \text{Encryption}(x_{\lceil \frac{n}{2} \rceil + 1}, S)$
Original Update($x_1, \ldots, x_{\lceil \frac{n}{2} \rceil}$)
```

Original Update($x_{\lceil \frac{n}{2} \rceil + 2}, \ldots, x_n$)


\[64 (= 3)\]
\[32 (= 2)\]
\[96 (= 7)\]
\[80 (= 5)\]
\[112 (= 10)\]

**FIGURE 1.** Search tree after encrypting the plaintext $X = \{3, 7, 2, 5, 10\}$ based on [10].

**C. FH-OPE SCHEME OF MAFFEI et al.**

The authors of [12] presented a new FH-OPE scheme under their revised security model. The main idea of this scheme is that the encryption algorithm runs deterministically with the relative order information for the input plaintexts as an additional input $\Gamma$. Fig. 2 shows the possible tree shapes when the duplicate plaintexts are encrypted. In [11], one of the four tree shapes was determined probabilistically based on the results of the $RC$ function. However, in [12], the shape of the tree was determined with a probability of 1 according to the randomized order $\Gamma$. For example, Fig. 2 shows the resultant search tree of $\{1, 2, 2, 1\}$ where $\Gamma = \{2, 4, 3, 1\}$ is given as the input. Here, Algorithm 1 is also used as the ciphertext update algorithm and the randomized order $\Gamma$ is used as an input to the encryption algorithm, this is, $(y, S') \leftarrow \text{Encryption}(x, S, \Gamma)$.

**D. FH-OPE SCHEME OF YANG et al.**

In [13], the authors presented a more efficient FH-OPE with an improved update algorithm under the security model of [12]. When the inserted data already exists in node $t$, the encryption algorithm computes a 1-bit hash of an additional input value $r$. Similar to [11], if the hash result is 0 (1), the input data are placed in the left (right) child node of node $t$. However, the crucial difference compared to [11] is that the orders of duplicate plaintexts are not determined internally and randomly but are intended externally as an input $r$; thus, their scheme possibly achieves the same security level as [12]. Both [11] and [12] modified the encryption algorithm of [10] for frequency-hiding, but Algorithm 1 was used to update ciphertexts without any modification. Because the node positions of the inserted duplicate plaintexts are randomly selected by the outputs of $RC$ and $\Gamma$, Algorithm 1 does...
not guarantee the generation of a balanced tree for duplicate plaintexts. To overcome this limitation, Yang and Kim [13] presented an improved update algorithm, described in Algorithm 2. Here, this algorithm simply generates a new distinct sequence \( X' = \{1, 2, \ldots, n\} \) from the original sequence \( X = \{x_1, \ldots, x_n\} \) as input to Algorithm 1. When the operation of Algorithm 1 is completed, a list of all re-encrypted ciphertexts and the updated state \( S \) are generated. Finally, \( x_i \) of \( S \) is replaced with the original plaintext \( x_i \) of \( X \). Because Algorithm 2 uses a distinct plaintext sequence as an input, it can be guaranteed to generate a balanced tree.

Algorithm 2: Improved Update [13]

\[
\begin{align*}
\text{Input:} & \text{ sorted } X = \{x_1, \ldots, x_n\} \text{ in ascending order} \\
\text{Output:} & \text{ updated } Y = \{y_1, \ldots, y_n\} \\
\text{State:} & S \\
\text{Initialization:} & S = \{(-1, -1), (N, M)\} \\
& n = \text{length of } X \\
& \text{Let } X' = \{1, 2, \ldots, n\} \\
& \text{Let } Y = \{y_1, \ldots, y_n\} \\
& \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
& \quad \text{let } x_i \text{ of } S \leftarrow x_i \text{ of } X \\
& \quad \text{return } Y
\end{align*}
\]

Although the recently proposed update algorithm 2 results in a significant reduction in the number of ciphertext updates for duplicate plaintexts, neither of the two previous update algorithms considers the input pattern of the encrypted data. As mentioned in the previous section, for a certain input pattern, for example, sequential data, uniform re-encryption into the ciphertext space of the ciphertexts to be updated may not be the best way to solve the problem.

IV. CONSTRUCTION OF OUR ALGORITHM

A. SYSTEM MODEL

The system model includes two interactive entities: an OPE client and an OPE server. The client encrypts the data using the OPE and then outsources it to the OPE server. The OPE server stores outsourced ciphertexts and records all operations (e.g., INSERT, DELETE, and UPDATE) as log files. Fig. 3 shows a system model including the simplified log records of the type and time of the most recent query for each stored ciphertext. The server can obtain an approximate input pattern (not a specific distribution) of the underlying plaintexts from the information collected from the log records and order-preserving ciphertexts.

B. BUILDING A WEIGHT SEQUENCE OF CIPHERTEXTS

The previous mutable OPE schemes [10], [11], [12], [13] generate a ciphertext by dividing the possible ciphertext space corresponding to the input plaintext in half. Thus, we can easily know that the ciphertexts are densely located in the frequently divided ciphertext space. Our main idea is to reduce the total number of updates by calculating a weight for each ciphertext based on the number of times each space is divided and updating ciphertexts with high (low) weights sparsely (densely).

Fig. 4 shows that the ciphertext space \((-1, M)\) is divided by \(y_1\) into two subspaces \((-1, y_1)\) and \((y_1, M)\). It also shows that space \((y_1, M)\) is divided again by \(y_2\) into two subspaces \((y_1, y_2)\) and \((y_2, M)\) where \(x_1 < x_2\). Here, we assign a weight \(w_i\) to each ciphertext \(y_i\) according to the number of times the ciphertext space on the left is divided; this is, \(w_1 = 1\) of \(y_1\), \(w_2 = 2\) of \(y_2\), and additionally define \(w_3\) as 2 of \(M\).

Definition 6 (Weight Sequence \(W\)): Let \(\overrightarrow{Y} = \{y_1, \ldots, y_n\}\) be a list of ciphertexts sorted according to the order in which they are generated. A weight sequence \(W = \{(y_1, w_1), \ldots, (y_n, w_n), (y_{n+1} = M, w_{n+1})\}\) is defined as follows. Let \(k\) be the index of the first ciphertext inserted after the last ciphertext update. For \(i \leq k - 1, w_i\) and \(w_{n+1}\) are initialized as 1. For \(k \leq i \leq n\), first finds index \(j \in \{1, \ldots, n\}\) such that \(y_i < y_j\) and \(|y_j - y_i|\) are the smallest in \(W\), then

![FIGURE 3. The system model.](image-url)

![FIGURE 4. Basic idea for building a weight sequence \(W\).](image-url)
updates \(w_i\) and \(w_j\) as follows:

\[
w_i \leftarrow w_j + 1, \quad w_j \leftarrow w_j + 1.
\]

Fig. 5 describes the process of calculating the weight of each ciphertext from the point of view of the server on \(\overrightarrow{Y} = \{64, 32, 48\}\) where \(M = 128\). First, \(W\) was initialized to \{\((\cdot, \cdot), (\cdot, \cdot), (\cdot, \cdot), (128, 1)\)\}, as shown in 5(a). For \(y_1 = 64\), \(w_1\) and \(w_4\) are updated as 2. Similarly, for \(y_2 = 32\), \(w_2\) and \(w_1\) are updated as 3. Lastly, for \(y_3 = 48\), \(w_3\) and \(w_1\) are updated as 4. Then the server can construct weight sequence \(W\) as \\{\((64, 4), (32, 3), (48, 4), (128, 2)\)\}. This weight sequence alludes to the distribution of the underlying plaintext, which suggests that it would be possible to construct more efficient update algorithms. Note that these algorithms can be constructed only from log records and ciphertexts on the server side; thus, the algorithm does not affect the security of the OPE scheme.

![Index y w](image1)

(a) Initialization of \(W\)

![Index y w](image2)

(b) \(y_1 = 64\)

![Index y w](image3)

(c) \(y_2 = 32\)

![Index y w](image4)

(d) \(y_3 = 48\)

**FIGURE 5.** How to generate \(W\) on \(\overrightarrow{Y} = \{y_1, y_2, y_3\}\).

When the server builds the weight sequence \(W\), it simply initializes the weights to 1 for the ciphertexts that have been updated previously (see Definition 6). The advantage of this setting is that the server does not have to recalculate the weight of the entire data; furthermore, the latest pattern of the input data is reflected in the update algorithm.

### C. PROPOSED UPDATE ALGORITHM

This subsection describes the more efficient ciphertext update algorithm we are proposing. The basic principle of the proposed algorithm is to set the size of \(|y_{i-1} - y_i|\) proportional to \(w_i\), because of the high probability that future ciphertexts will be placed around a ciphertext \(y_i\) with a large \(w_i\).

Consider the following update example: Let \(\overrightarrow{Y} = \{64, 96, 80, 72, 68, 76\}\) with \(M = 128\) and \(k = 4\). First \(W\) was initialized to \{\((64, 1), (96, 1), (80, 1), (128, 1)\)\} depending on the index \(k\). In this case, we compute \(w_{k \leq i \leq n}\) for \(y_k \leq y_i\) based on Definition 6. For \(y_4 = 72\), \(w_4\) and \(w_3\) are updated as 2 where \(y_4 < y_3\) and \(|y_3 - y_4|\) are the smallest in \(W\). Then we can construct weight sequence \(W = \{64, 1), (68, 3), (72, 3), (76, 3), (80, 3), (96, 1), (128, 1)\}\) in ascending order by the first element. In each iteration from loop, we calculate the ciphertext \(y_i\) by adding the result obtained by multiplying \(w_i\) and \(\frac{M}{\sum_{j=1}^{n} w_j}\) to \(y_{i-1}\), here \(\frac{M}{\sum_{j=1}^{n} w_j} = 8.533...\) Finally, we can get the updated \(Y = \{9, 34, 60, 85, 111, 119\}\).

We analyzed our construction by comparing it with the algorithm we previously reviewed in terms of efficiency.

- The outstanding feature of the proposed algorithm is that the server can proceed with ciphertext updates without any help from the client. Therefore, sufficient computing power of the server can be utilized, providing evidence that it does not adversely affect the security of the underlying OPE scheme. The client can update the state \(S\) by executing the update algorithm independently of the server or by receiving the updated results from the server.

- Because the previous algorithms internally call the encryption algorithm underlying the OPE, the cost of calling the function is added, and the update efficiency is dependent on the underlying OPE. However, our proposed algorithm does not present this problem at all.

- In the previous update process, the cost of communication between the client and server was \(O(n)\), considering that the client had to send all the updated ciphertexts to the server. However, in our proposed algorithm, because the server and client can perform the update operation independently, these communications costs can be saved.

- The encryption algorithm of the OPE generated computational cost of \(O(\log n)\) to traverse the binary search tree in \(S\). Therefore, for \(n\) elements, the computational cost of the previous update algorithms is \(O(n \log n)\). Algorithm 3 also runs in \(O(n \log n)\) time. Specifically, calculating the weight of each ciphertext requires \(O(\log n)\) search time to find index \(j\), then it requires \(O(n \log n)\) time for \(n\) ciphertexts. The loop in Algorithm 3 also iterates \(n\) times. Hence the overall
computational cost is bound by $O(n \log n) + O(n) = O(n \log n)$.

V. EXPERIMENTS

We measured the encryption time and the number of ciphertext updates by applying different update algorithms to the existing OPE schemes [10], [11], [12], [13]. Our experiments were run on a desktop with an AMD Ryzen 7 PRO 4750G (3.60 GHz, 16GB RAM) with Windows 10, and the code was written in Python 3.9.7.

(Datasets): Our experiments were performed on various datasets, including three synthetic and real-world datasets. The synthetic datasets are sequentially decreasing data (Dataset 1), distinct uniformly distributed data (Dataset 2), and non-distinct uniformly distributed data (Dataset 3) where $M$ is fixed as $2^{16}$. Dataset 1 consists of $\{N, N-1, \ldots, 1\}$ where $N \in \{2^{10}, 2^{11}, 2^{12}, 2^{13}\}$. Dataset 2 consists of $N$ distinct data that are randomly selected in $\{1, 2, \ldots, 2^{l}\}$ where $l \in \{10, 11, 12, 13\}$. Similarly, Dataset 3 consists of $2^{l+2}$ non-distinct data that are uniformly selected from $\{1, 2, \ldots, 2^{l}\}$ where $l \in \{8, 9, 10, 11\}$. The first real-world dataset contains the height and weight of the major league baseball players (MLB players). Here, we set $M$ as $2^\lambda$ where $\lambda \in \{17, 18, 19, 20\}$. The other dataset, which contains sensitive data about hospital patients, is a publicly available dataset that comprises Statewide Planning and Research Cooperative System (SPARCS) inpatient data from New York State [27]. In this dataset, we set $M$ equal to $2^\lambda$ where $\lambda \in \{24, 26, 28, 30, 32\}$.

1Dataset obtained from https://www.seanlahman.com/baseball-archive/statistics.
J. Yang, K. S. Kim: Efficient Update Algorithm for Mutable Order-Preserving Encryption

FIGURE 9. Experimental results under FH-OPE [13] on Dataset 1-3.

TABLE 1. Mean and standard deviation (SD) of weight and height of the MLB players.

|       | N   | Mean  | SD    |
|-------|-----|-------|-------|
| Weight (lbs.) | 28,293 | 107   | 199.04 | 22.65 |
| Height (in.)   | 28,293 | 18    | 73.50  | 2.28  |

FIGURE 10. Distributions of the MLB player datasets.

A. SYNTHETIC DATASETS

Figs. 6-9 compare the performance of the three different update algorithms based on the number of updates and encryption time. The results confirmed an improvement in the overall performance relative to that of the previous algorithms. In particular, from Figs. 6(a)-9(a), we can verify that our update algorithm has reduced the number of updates in Dataset 1 to a remarkable extent owing to the unique re-encryption method of our algorithm. As shown in Figs. 6(b)-9(b), the performance of our algorithm does not differ significantly from that of the previous algorithms because the weights of the ciphertexts are similar to each other in the distinct uniformly distributed data. For the non-distinct dataset Dataset 3, the deterministic OPE scheme of [10] undertakes a very small number of updates, unlike the FH-OPE schemes, because it produces the same ciphertext for duplicated plaintexts. However, because Algorithm 1 does not guarantee the generation of uniform ciphertexts, its performance is inferior to that of Algorithms 2-3, as shown in Figs. 7(c)-9(c).

B. REAL-WORLD DATASETS

In Dataset 3 of synthetic datasets, we have already confirmed that Algorithm 1 causes significantly lower performance than others. However, in real-world datasets, data duplication is actually a very common, thus using these datasets, we compared Algorithm 2 and Algorithm 3 more intensively under the FH-OPE schemes [11], [12], [13] by measuring the exact number of ciphertext updates.

1) MLB PLAYER DATASET

This dataset contains 28,293 records and we chose the Weight and Height fields of which the details are listed in Table 1. Fig. 10 shows that the data in both of the two chosen data fields follow Gaussian distributions. In the uniformly
• **Length of Stay.** The length of stay refers to the number of days each patient spent in the unit before dying or being discharged and ranges from 1 to 200.

• **CCS Diagnosis Code.** The CCS diagnosis code, developed by the Medical Research Quality Authority (AHQR), refers to ICD-9-CM clustered into a manageable number of clinically meaningful categories.

• **APR DRG Code.** The APR DRG code refers to all inpatients who are classified into disease groups with similar treatment contents.

• **APR MDC Code.** The APR MDC code is formed by dividing all possible principal diagnoses from ICD-9-CM into 25 diagnosis areas.

2) **HOSPITAL PATIENT DATASET**

The hospital patient dataset includes 1,048,575 records, each with 33 columns. We chose a subset of attributes such as the length of stay, CCS (Clinical Classification Software) Diagnosis Code, APR DRG (All Patient Refined Diagnosis Related Group) Code, and APR MDC (All Patient Refined Major Diagnosis Category) Code fields, the details of which are shown in Fig. 12. The distributions of the data for each attribute is shown in Fig. 13. Based on the results in Figs. 14(a)-14(c) and the previous sections, we can conclude that our proposed update algorithm, regardless of the underlying mutable OPE schemes, succeeds in improving the overall performance. In particular, when the input dataset has a unique distribution rather than a uniform distribution, the proposed algorithm improves the performance noticeably.
VI. CONCLUSION
Our analysis of the update algorithms of existing mutable (FH)OPE schemes indicated that these algorithms are inefficient. We subsequently proposed a new efficient update algorithm for any mutable OPE schemes with the main objective of reducing the total number of updates by considering the input pattern of the encrypted data. Our algorithm enables the server to proceed with ciphertext updates without any help from the client and without any security loss because the algorithm can operate using only system logs and stored ciphertexts. Our experimental results showed a significant performance improvement on a variety of datasets when the algorithm was applied to existing mutable OPE schemes.

REFERENCES
[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, “Order preserving encryption for numeric data,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2004, pp. 563–574.
[2] A. Boldyreva, N. Chenette, Y. Lee, and A. O’Neill, “Order-preserving symmetric encryption,” in Proc. Ann. Int. Conf. Theory Appl. Cryptograph. Techn., Cham, Switzerland: Springer, 2009, pp. 224–241.
[3] A. Boldyreva, N. Chenette, and A. O’Neill, “Order-preserving encryption revisited: Improved security analysis and alternative solutions,” in Proc. Ann. Cryptol. Conf. Cham, Switzerland: Springer, 2011, pp. 578–595.
[4] S. Lee, T.-J. Park, D. Lee, T. Nam, and S. Kim, “Chaotic order preserving encryption for efficient and secure queries on databases,” IEICE Trans. Inf. Syst., vol. E92-D, no. 11, pp. 2207–2217, 2009.
[5] H. Kadhem, T. Amagasa, and H. Kitagawa, “MV-OPE: Multivalued-order preserving encryption scheme: A novel scheme for encrypting integer value to many different values,” IEICE Trans. Inf. Syst., vol. E93-D, no. 9, pp. 2520–2533, 2010.
[6] L. Xiao, I.-L. Yen, and D. T. Huynh, “Extending order preserving encryption for multi-user systems,” Cryptol. ePrint Arch., Paper 2012/192, 2012. [Online]. Available: https://eprint.iacr.org/2012/192
[7] L. Xiao and I.-L. Yen, “A note for the ideal-order-preserving encryption object and generalized order-preserving encryption,” Cryptol. ePrint Arch., Paper 2012/350, 2012. [Online]. Available: https://eprint.iacr.org/2012/350
[8] D. Liu and S. Wang, “Nonlinear order preserving index for encrypted database query in service cloud environments,” Concurrency Comput. Pract. Exper., vol. 25, no. 13, pp. 1967–1984, Sep. 2013.
[9] R. A. Popa, F. H. Li, and N. Zeldovich, “An ideal-privacy protocol for order-preserving encoding,” in Proc. IEEE Symp. Secur. Privacy, May 2013, pp. 463–477.
[10] F. Kerschbaum and A. Schröpfer, “Optimal average-complexity ideal-privacy order-preserving encryption,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Nov. 2014, pp. 275–286.
[11] F. Kerschbaum, “Frequency-hiding order-preserving encryption,” in Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2015, pp. 656–667.
[12] M. Maffei, M. Reinert, and D. Schröder, “On the security of frequency-hiding order-preserving encryption,” in Proc. Int. Conf. Cryptol. Netw. Secur. Cham, Switzerland: Springer, 2017, pp. 51–70.
[13] J. Yang and K. S. Kim, “Practical frequency-hiding order-preserving encryption with improved update,” Secur. Commun. Netw., vol. 2021, pp. 1–8, Dec. 2021.
[14] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, “CryptDB: Protecting confidentiality with encrypted query processing,” in Proc. 23rd ACM Symp. Operating Syst. Princ. (SOSP), 2011, pp. 85–100.
[15] S. T. Samuel, M. F. Kaashoek, and M. N. Zeldovich, “Processing analytical queries over encrypted data,” Proc. VLDB Endow., vol. 6, no. 5, 2013.
[16] F. Taigel, A. K. Tueno, and R. Pibernik, “Privacy-preserving condition-based forecasting using machine learning,” J. Bus. Econ., vol. 88, no. 5, pp. 563–592, 2018.
[17] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, “Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation,” in Proc. Ann. Int. Conf. Theory Appl. Cryptograph. Techn., Cham, Switzerland: Springer, 2015, pp. 563–594.
[18] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu, “Practical order-revealing encryption with limited leakage,” in Proc. Int. Conf. Fast Softw. Encrypt. Cham, Switzerland: Springer, 2016, pp. 474–493.
[19] K. Lewi and D. J. Wu, “Order-revealing encryption: New constructions, applications, and lower bounds,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2016, pp. 1167–1178.
[20] D. Cash, F.-H. Liu, A. O’Neill, and C. Zhang, “Reducing the leakage in practical order-revealing encryption,” Cryptol. ePrint Arch., Paper 2016/661, 2016. [Online]. Available: https://eprint.iacr.org/2016/661
[21] D. Cash, F.-H. Liu, A. O’Neill, M. Zhandry, and C. Zhang, “Parameter-hiding order revealing encryption,” in Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur. Cham, Switzerland: Springer, 2018, pp. 181–210.
[22] K. S. Kim, “New order-revealing encryption with shorter ciphertexts,” Information, vol. 11, no. 10, p. 457, Sep. 2020.
[23] M. Naveed, S. Kamara, and C. V. Wright, “Inference attacks on property-preserving encrypted databases,” in Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2015, pp. 644–655.
[24] F. B. Durak, T. M. Dübuisson, and D. Cash, “What else is revealed by order-revealing encryption?” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2016, pp. 1155–1166.
[25] P. Grubbs, K. Sekniqi, V. Bindshadler, M. Naveed, and T. Ristenpart, “Leakage-abuse attacks against order-revealing encryption,” in Proc. IEEE Symp. Secur. Privacy (SP), May 2017, pp. 655–672.
[26] V. Bindshadler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov, “The tao of inference in privacy-protected databases,” Cryptol. ePrint Arch., vol. 11, no. 11, pp. 1715–1728, Jul. 2018.
[27] (2012). Hospital Inpatient Discharges (Sparcs De-Identified):2012. [Online]. Available: https://health.data.ny.gov/Health/Hospital-Inpatient-Discharges-SPARCS-De-Identified/3m9u-ws8e

JIHYE YANG received the B.S. degree in cyber security from Daegu Catholic University, Daegu, South Korea, in 2021, where she is currently pursuing the M.S. degree in computer science and engineering. Her research interests include cryptography and database security.

KEE SUNG KIM received the M.S. and Ph.D. degrees from the Graduate School of Information Security, Korea University, Seoul, South Korea, in 2011 and 2015, respectively. He was a Senior Researcher at the Affiliated Institute of ETRI, Daejeon, South Korea, from 2014 to 2018. He is currently an Assistant Professor at the School of Information Technology Engineering, Daegu Catholic University, South Korea. His research interests include cryptography, database security, privacy-enhancing technology, and secure protocols.

* * *

102018 VOLUME 10, 2022