HOPE-L: A Lossless Database Watermarking Method in Homomorphic Encryption Domain

1st Xueqi Zhang
University of Science and Technology of China
Anhui, China
xqzhang7@mail.ustc.edu.cn

2nd Haiyong Xie
University of Science and Technology of China
Anhui, China
hxie@ustc.edu.cn

3rd Hui Lin
National Engineering Laboratory for Public Safety Risk Perception and Control by Big Data
Beijing, China
linhui@whu.edu.cn

***Abstract*—Cloud computing has been widely adopted in the Internet economy; however, this poses numerous risks including illegal data copying and digital copyright infringement, leading to the challenge on protecting the security and copyright of key data stored in databases. In this paper, We propose a novel method, Lossless Database Watermarking in Homomorphic Encryption domain (HOPE-L), to address this challenge. Our method combines the database encryption (*i.e.*, homomorphic encryption, order-preserving encryption) and information hiding (*i.e.*, lossless databased watermarking) technologies. More specifically, the information hiding algorithm leverages the homomorphic properties of the homomorphic encryption algorithm to embed secret data. No distortion is introduced throughout the data embedding process. Therefore, we can embed the watermark into the encrypted data losslessly. The watermark can be used to authenticate the copyright, and the recipients can directly recover the original database without loss. Analysis and theorem prove that HOPE-L can achieve more embedding space and no distortion. Extensive experiments show that the whole operation process is time-efficient, the embedded watermark is robust, and HOPE-L performs more robust than the existing algorithms and can resist common database attacks.**

***Index Terms*—Encrypted database, Lossless database watermark, Homomorphic encryption, Order-preserving encryption**

I. Introduction

The emergence of Internet has profoundly influenced and changed people's daily lives. With the rapid development of cloud computing in the past decade, Internet-based systems have collected enormous data and stored them in various databases in the cloud [1]. Such data typically stores sensitive information such as human behavior, video, location, and so on. Numerous commercial systems have been built upon collection as well as utilization of such data. In these systems, digital transactions have become the norm. As a result, it has become an urgent and significant challenge on how to protect the security and copyright of these data stored in databases.

Researchers have proposed numerous solutoins in order to protect and track the sensitive data (*e.g.*, anti-counterfeiting, copyright, and trace to the source). The existing solutions involve two key technologies. The first key technology is database encryption, which guarantees the security of database content [2]. Since 1978, researchers have made many progresses on homomorphic encryption [2]–[6]. However, the encrypted data must be decrypted when used them. Agrawal *et al.* proposed methods to allow comparison on encrypted data [7]–[10]; thus, when a database is encrypted and becomes a ciphertext database, we can still perform relational operations.

The second key technology is database watermarking. Database watermarking algorithms are challenging, because databases often face frequent update operations and it is difficult to find a large amount of redundant space to embed watermarks. Due to these challenges, the research progress is relatively slow. Furthermore, there is a more important issue that the process of embedding watermarks may lead to changes of original data. Additionally, in most cases, data should be stored and retrieved wiht high accuracy, even a little distortion may lead to incorrect results. In order to restore original data without distortion, researchers proposed a reversible database watermark embedding method [11]–[15].

So far, the combination of lossless watermarking algorithm and database encryption is still a relatively frontier research field in the world. The existing solution is to protect the content security through database encryption, and verify the copyright through database watermark. However, when storing data in cloud-based databases, existing solutions face multiple challenges on the security of the database content: joint implementation of data security and copyright verification, lossless watermark suitable for database, space- and time-efficiency of embedding, robustness of database watermark to resist common database attacks.

In this paper, we propose a two-step, novel method to address the above challenges. In the first step, we propose a novel database encryption system, homomorphic order-preserving encryption (HOPE). In the second step, based on HOPE, we propose a lossless database watermarking method in homomorphic encryption domain, by combining homomorphic encryption, order-preserving encryption and information hiding altogether. We refer to the final approach as HOPE-L. Our method can achieve the protection of the copyright and database content simultaneously. The recipient can directly recover the original database without loss. The experimental results and analysis show that HOPE-L achieves the advantages of both space- and time-efficiency of embedding, strong robustness, and no distortion.

We summarize our contributions as follows:

- We propose a novel method (HOPE), combining homomorphic encryption and order-preserving encryption, to implement an encryption system on databases. HOPE can protect the privacy of database content while maintaining database availability.
- We propose a novel lossly database watermarking in

homomorphic encryption domain (HOPE-L). To the best of our knowledge, HOPE-L is the first approach to address the challenges of database security and copyright protection simultanenously.
- We conduct theoretical analysis and extensive experiments to prove that HOPE-L can achieve more embedding space and no distortion. The embedded watermark has strong robustness, and the whole operation process is time-efficient. Additionally, HOPE-L performs more robust than the existing algorithms and can resist common database attacks.

II. Related Work

Database Encryption. There are many databased encryption algorithms, most of which can be further divided into the following two sub-categories.

The first sub-category is homomorphic encryption. Rivest *et al.* proposed a semi-homomorphic encryption method based on the problem of large integers [3]. Elgamal proposed a multiplicative semi-homomorphism based on the discrete logarithm problem [4]. Paillier proposed an additive semi-homomorphic encryption scheme, which improves the security of the algorithm, and the algorithm is simple in construction [5]. The above encryption schemes belong to semi-homomorphic encryption. Based on the lattice cryptography theory [6], Gentry proposed a fully homomorphism scheme that allows any operation on ciphertext. At present, fully homomorphism is still not perfect, as this method is computationally expensive, time consuming and in-efficient.

The second sub-category is order-preserved encryption (OPE). To address the disadvantage of homomorphic encryption (*i.e.*, the encrypted data must be decrypted before performing any operations on data, which is computationally expensive), Agrawal *et al.* first proposed the concept of OPE [7] in 2004. Subsequently, Popa *et al.* designed and developed the CryptDB ciphertext database system [9] in 2011, which uses the order-preserving BCLO algorithm [8] to achieve comparison and query operations in the database. In 2013, Popa *et al.* proposed an order-preserved encryption algorithm, mOPE, based on a binary sort tree [10].

Database Watermarking Technology. In 2002, Agrawal *et al.* proposed the first database watermarking algorithm on relational database [16]. Sion *et al.* proposed a new watermarking algorithm using the statistical characteristics of data [17]. Subsequently, researchers proposed numerous watermarking algorithms [17]–[20], most of which lead to data distortion.

In order to restore the original data without distortion, researchers proposed a reversible database watermark embedding method [11]–[15]. In 2006, Zhang *et al.* proposed the first reversible watermarking scheme [21], but this method is not robust. Gupta *et al.* proposed a reversible watermarking technique in 2008 based on differential extension (GADEW) [22]. In 2017, Imamoglu *et al.* designed a new type of reversible database watermarking method with the firefly algorithm (FFADEW) [23], which reduced the distortion. The existing research has been exploring lossless watermarking, but there is still no good method to achieve lossless and robust. The lossless watermark embedding provides an excellent solution for the database usability after embedding watermark.

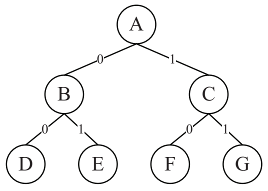


Fig. 1: the sort tree.

TABLE I: OPCs for nodes in Fig. 1

Node	Path (P)	OPC (c)	Height (h)
A	[]	[]100	3
B	[0]	[0]10	2
C	[1]	[1]10	2
D	[00]	[00]1	1
E	[01]	[01]1	1
F	[10]	[10]1	1
G	[11]	[11]1	1

Combination of database encryption and watermarking. In recent years, there are only a few research on the combination of these two technologies. For example, Xiang *et al.* proposed a method of combining OPE with database watermarking technology [24]. However, this method does not allow database updates and can incur distortion on data.

III. HOPE: Homomorphic Order-Preserving Encryption

In this section, we present a novel database encryption algorithm that combines homomorphic encryption and order-preserving encryption. The new algorithm is homomorphic order-preserving encryption (HOPE). HOPE first encodes all of the data using order-preserving coding, and then encrypts the original data with homomorphic encryption.

In order to best illustrate the design of HOPE algorithm, we next first review the two basic concepts of order-preserving encryption and homomorphic encryption in Section III-A; we then present the HOPE algorithm in details in Section III-B.

A. Basic Concepts

1) Order-Preserving Code (OPC): Order-preserving encryption we used is actually a process of encoding. Given a set of data as input, we first organize the data into a binary balanced sorting tree. In such a tree, each node has a corresponding path P, a height h, and an order-preserving code c consisting of multiple binary bits.

We denote by γ the root node. The root node's path is empty (*i.e.*, $P(\gamma) = []$), and its height $h(\gamma)$ is the height of the tree. The leaf nodes' height is 1. For brevity, we refer to a node with its data value being A as node A. We denote by l the total length of OPC. Note that l is equal to the total height of tree, namely, $l = h(\gamma)$.

Suppose that we need to derive the order-preserving code for a node A. The process can be divided into two steps.

In the first step, we start from the root node and repeatedly query the value A against the current node's value, so that we can obtain the binary encoding path for A as follows: if A is less than the current node's value, the current coded bit is '0' and otherwise '1'. These coded bits are appended to P while we repeatedly query the tree.

In the second step, we append a bit '1' to the current binary encoding path P, and then determine if the length of the current binary encoding path value is less than l. If it is, we append multiple bits of '0' to P until its length reaches l:

$$c = [P]\,10...0 \tag{1}$$

Note that in order to minimize the length of OPC and the query path, thus minimizing the space for storing the order-preserving codes, we need to construct a binary balanced sorting tree to minimize the height of the tree.

We use an example to illustrate the process of deriving the OPC. Suppose that we have a set of 7 data values denoted by alphabetical letters from A to G. Assume that $D < B < E < A < F < C < G$. A binary balanced sorting tree can be constructed as shown in Fig. 1.

Apparently, the total length of OPC for the nodes is 3, namely, $l = h(A) = 3$. Table I enumerates all OPCs for each node in Fig. 1. More specifically, the root node A's path value is $P(A) = []$. According to Eq. (1), its OPC is $c(A) = []100$. The leaf node D's path is $P(D) = [00]$ (*i.e.*, a concatenation of the bits on the links along the path from the root A to D). Therefore, according to Eq. (1), its OPC is $c(D) = [00]1$.

2) Homomorphic Encryption: Homomorphic encryption refers to an encryption function that satisfies the following condition: when performing an operation on encrypted data, the result (after decryption) is the same as the result obtained when applying the same operation on the original plaintext data. More specifically, given two plaintext data x and y, homomorphic encryption guarantees

$$D\left(E\left(x\right) \odot E\left(y\right)\right) = x \oplus y, \tag{2}$$

where E represents an encryption function, D represents a decryption function, $\odot$ and $\oplus$ represent operational symbols, respectively. When $\oplus$ represents the addition operator, the algorithm is called additive homomorphic encryption; when $\oplus$ represents multiplication operator, the algorithm is called multiplicative homomorphic encryption. If $\oplus$ can represent any operation, the algorithm is fully homomorphic.

Homomorphic encryption allows third-party users to operate on encrypted data without revealing any data information. However, homomorphic encryption is computationally expensive. In order to balance the effectiveness and efficiency, we adopt Paillier semi-homomorphic algorithms instead, and other semi-homomorphic algorithms are also applicable.

Note that in the encryption process of the Paillier algorithm, each encryption process generates a large random number for calculation. Therefore, even with the same plaintext, two different encryption processes will generate different encryption results due to the fact that they use different random numbers. This can be used to resist statistical attacks.

B. HOPE Algorithm

Inpsired by the order-preserving code and the homomorphic encryption, we design the HOPE algorithm as follows. Firstly, we sort the data stored in a given database and construct a binary sorting tree. Secondly, we adjust the tree structure to form a balanced tree. Thirdly, we derive the order-preserving code for each node in the balanced tree. Lastly, we encrypt the original data using Paillier homomorphism algorithm. The entire HOPE algorithm is shown in Algorithm 1.

It is most desirable that Algorithm 1 can inherit the benefits of both order-preserving code and homomorphic encryption after we combine them. We prove that combination of the two methods still maintain the key properties of each method.

Theorem 1: Algorithm 1 is order-preserving.

Proof 1: For any given data a and data b where $a \leq b$ and $a, b \in \mathcal{D}$, we only need to prove that after application of Algorithm 1, the condition $e_a \leq e_b$ still holds.

Algorithm 1 applies the order-preserving encoding to each data in the set $\mathcal{D}$ and generates $c_a = \texttt{OPC}(a)$ and $c_b = \texttt{OPC}(b)$. Apparently, $\texttt{OPC}(a) \leq \texttt{OPC}(b)$, therefore, $c_a \leq c_b$.

Algorithm 1 HOPE

```
Input: a set $\mathcal{D}$ of data ; encryption key $k$
Output: a set $E(\mathcal{D})$ of encrypted data
1: $\mathcal{D}' = \texttt{sort}(\mathcal{D})$ //sort all data in $\mathcal{D}$ by their sizes.
2: $T = \texttt{form\_binary\_sort\_tree}(\mathcal{D}')$ //Construct a binary sort tree.
3: $T' = \texttt{form\_balanced\_tree}(T)$ //Adjusting binary tree to the balanced state.
4: for $d \in \mathcal{D}$ do
5:     $c_d = \texttt{OPC}(d)$ //Derive OPC for each data.
6: end for
7: for $d \in \mathcal{D}$ do
8:     $e_d = E_k(d)$ //Apply Paillier algorithm.
9: end for
10: return $E(\mathcal{D}) = \{e_d, c_d | d \in \mathcal{D}\}$
```

No matter how data changes later, OPC always records the size of the original data. Therefore, encrypted data can still compare the numerical order.

Theorem 2: Algorithm 1 achieves addition homomorphism.

Proof 2: We first review the relevant key details of the Paillier algorithm, as Algorithm 1 is built upon the Paillier algorithm. The Paillier algorithm consists of three key steps: key generation, encryption, and decryption.

In the key generation step, two large independent prime numbers p and q are randomly generated. Generate a random integer g, $g \in Z^*_{n^2}$. Calculate $n = pq$ and $\lambda = lcm\left(p-1, q-1\right)$ (the least common multiple of $p-1$ and $q-1$). Calculate μ by $\mu = \left(L\left(g^\lambda \bmod n^2\right)\right)^{-1} mod\, n$, where the function $L\left(x\right) = \frac{(x-1)}{n}$. At the end of this step, we obtain the public key (n, g), and the private key (λ, μ).

In the encryption step, select a random number r, where $0 < r < n$, $r \in Z^*_{n2}$. For any information m to be encrypted, $0 \leq m < n$, we calculate its ciphertext $c = E\left(m, r\right) = g^m r^n \bmod n^2$.

In the decryption step, for any ciphertext c to be decrypted, we can derive its plaintext by calculating $m = D(c) = L\left(c^\lambda \bmod n^2\right) \times \mu \bmod n$.

We next start to prove that Algorithm 1 maintains the addition homomorphism, which can be expressed in two forms (Eq. (3),Eq. (4)). For the first form, we prove that for any given plaintext data m_1, m_2, The product of their corresponding ciphertexts is equal to the encrypted result of the sum of their plaintexts. In other words:

$$D(E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2) = m_1 + m_2 \bmod n \tag{3}$$

Assume that for two given plaintexts m_1, m_2, the encryption process selects random numbers r_1, r_2. Then we have:

$$\begin{aligned} left &= D(g^{m_1} r_1^n \cdot g^{m_2} r_2^n \bmod n^2) \bmod n \\ &= D(g^{m_1+m_2} r_1 r_2^n \bmod n^2) \bmod n \\ &= L(g^{(m_1+m_2)\lambda} r^{n\lambda} \bmod n^2)\mu \bmod n \\ &= L(g^{(m_1+m_2)\lambda} \bmod n^2) L(g^\lambda \bmod n^2)^{-1} \bmod n \\ &= L((1+n)^{(m_1+m_2)\lambda}) L((1+n)^\lambda)^{-1} \bmod n \\ &= (m_1+m_2)\lambda \bmod n^2 (\lambda \bmod n^2)^{-1} \bmod n \\ &= m_1 + m_2 \bmod n \end{aligned}$$

TABLE II: Notations

Symbol	Meaning
t	a tuple in a given database
$t.P$	the primary key of tuple t
$t.A$	the attribute value of tuple t
N	the total number of tuples
n	group number
w	the watermark
L	the length of the watermark
K	a hash value used as a key
H	a hash algorithm
h_1, h_2	variables
f	the f-th digit of watermark w
p	the p-th digit of attribute values of tuples $t.A$
x, y	plaintexts
$Dig(T, s)$	Dig($\cdot$) is a function, T is a BigInteger and s is an integer, Dig(T,s) returns the t-th digit of T
$E(\cdot), D(\cdot)$	the encryption, decryption function

For the second form, we prove that the product of a ciphertext with a plaintext raising g will decrypt to the sum of the corresponding plaintexts. In other words:

$$D(E(m_1, r_1) \cdot g^{m_2} \bmod n^2) = m_1 + m_2 \bmod n \quad (4)$$

The proof process is similar to that of Eq. (3). Due to the space limitation,we skip the proof.

IV. HOPE-L: Lossless Database Watermarking in Homomorphic Encryption Domain

We now present the design of the lossless database watermarking algorithm in homomorphic encryption domain. First, we apply order-preserving encryption to allow comparison on encrypted data directly. Second, we apply homomorphic encryption to protect data security. Last, in order to keep the data unchanged after embedding watermark, we design a novel watermark embedding method, by leveraging the properties of homomorphism, to achieve lossless database watermark method in homomorphic encryption domain.

A. Notations

We use a set of notations, as show in Table II, to ease the description and the formulation of HOPE-L.

B. Watermark Generation

Watermark generation includes two steps. The first step is to concatenate the database name, table name and user identity. We refer to the concatenated string as the user fingerprint information (ID). The second step is to translate the ID string into a fixed-length hash value using a hash algorithm. We use the hash value as the watermark w. Note that all state-of-the-art hash functions can be used to generate the watermark.

C. Watermark Embedding

Once the watermark for a given database is generated, we start to embed the watermark into the database. We embed the watermark into the encrypted database (namely, apply HOPE to encrypt the original plaintext database). Note that the encrypted database contains encrypted data and OPC.

we use the HOPE-L algorithm, as shown in Algorithm 2, to embed watermark w into encrypted database. In the algorithm, we adopt the double hash positioning method during embedding. The whole embedding process can be divided into three steps. The first step is grouping, in which we calculate the group number n of the database table according to the length L of watermark, where $n = \frac{N}{L-1}$. Embed as many watermarks as possible in all tuples to resist database level attacks.

Algorithm 2 Embedding process of HOPE-L

```
Input: Encrypted database ; Watermarking $w$ ; Key $K$
Output: Encrypted database embedded watermark $w$
1: $n = \frac{N}{L-1}$ //Number of watermark in database.
2: for $i<N$ do //Traversing all tuples.
3:     $h_1 = H(t_i.P \parallel K)$
4:     $f = h_1 \bmod L$ //$Dig(w, f)$: The $f$-th digit of $w$.
5:     $h_2 = H(K \parallel h_1)$
6:     $p = h_1 \bmod 10$ //$Dig(t.A, p)$: The $p$-th digit of the attribute value $A$ of tuple $t$.
7:     while $Dig(t.A, p) \neq Dig(w, f)$ do
8:         $t.A = t.A \cdot E(0)$ //Embed $Dig(w, f)$ to $Dig(t.A, p)$.
9:     end while
10: end for
11: return Encrypted database embedded watermark $w$
```

The second step is to determine the watermark and embedding position by using the double hash positioning method.

Assuming that P is not attacked and unchangeable, we can determine which tuples to embed watermark. We compute the first hash value on the concatenation of P and K, namely, $h_1 = H(t_i.P \parallel K)$, where $\parallel$ means concatenation of two strings. We then compute $f = h_1 \bmod L$; we take f-th digit of w as a watermark to embed in this round.

We then compute the second hash value $h_2 = H(K \parallel h_1)$, and obtain the lowest digit of h_2 by computing $p = h_1 \bmod 10$. We use p to determine the location of watermark embedding. In other words, we embed watermark $Dig(w, f)$ into the p-th digit of attribute values of tuples $t.A$. The double hash method helps improve the resistance against watermark attacks.

The third step is to leverage the homomorphism properties in Eq. (3) to finalize lossless watermark embedding. Note that data encrypted by HOPE satisfies:

$$E(x) \cdot E(y) = E(x + y)$$

If we let $y = 0$, the product of $E(x)$ and $E(0)$ is equal to $E(x)$:

$$E(x) \cdot E(0) = E(x + 0) = E(x) \quad (5)$$

We leverage this property to achieve losslessly watermark embedding. More specifically, when we embed watermark into attribute values $t.A$ (note that $t.A$ is an encrypted data by HOPE and owns the homomorphic properties), we can repeat multiplying $t.A$ and $E(0)$ until the new $Dig(t.A, p)$ is equal to $Dig(w, f)$. Since the homomorphic encryption process introduces a nonce, the result of each encryption of the same number is different. By using the homomorphic characteristic, we modify a certain digit in the encrypted value by continuously multiplying $E(0)$, and finally achieve the purpose of embedding the watermark.

Theorem 3: Algorithm 2 achieves lossless watermark embedding.

Proof 3: HOPE-L completes the watermark embedding process by multiplying $E(0)$ continuously, which does not

Algorithm 3 Extracting process of HOPE-L

```
Input: Encrypted database embedded watermark $w$; Key $K$
Output: Watermark $w$
1: $n = \frac{N}{L-1}$ //Number of watermark in database.
2: for $i<N$ do //Traversing all tuples.
3:     $h_1 = H(t_i.P \parallel KEY)$
4:     $f = h_1 \bmod L$ //Extract the $f$-th digit of $w$.
5:     $h_2 = H(KEY \parallel h_1)$
6:     $p = h_1 \bmod 10$ //Extract the $p$-th digit of the attribute value $A$ of tuple $t$.
7:     for $A \in$ Attributes do
8:         $W[f][j] = Dig(t.A, p)$ //Store all extractions in a two-dimensional array.
9:     end for
10: end for
11: $w =$ voting mechanism($W[][]$)
12: return watermark $w$
```

change the original value. Therefore, no matter how many times this step has gone through, the newly obtained value remains unchanged after decryption.

It is worth of noting that relational databases are composed of tuples; thus it is typically difficult to find a large amount of redundant space to embed watermarks. However, HOPE-L can effectively leverage the existing limited redundant space in relational databases. In HOPE-L, each numerical attribute can embed watermarks. Vertically, according to the length L of watermark, we divide all tuples in the database into L groups, and each bit of watermark is embedded into $\frac{N}{L}$ tuples. This is first a considerable embedding space. At the horizontal level, each tuple has multiple attributes (assuming v), the same watermark bit will be embedded into all attributes. In total, 1 bit watermark will have $\frac{Nv}{L}$ embedding space.

D. Watermark Extraction

The process of watermark extraction, as shown in Algorithm 3, is the revserse of the watermark embedding process. It will extract a watermark with practical significance losslessly, which connects the database with the owner, so as to realize the copyright protection. The watermark extraction requires the same key as the key used in embedding. During extraction, every digit extracted from the watermark is determined according to all the embedded attribute values.

In reality, databases will undergo frequent update operations, which might cause loss or damage to the watermark. In order to address this problem, we adopt a voting mechanism to recover the watermark, namely, the most frequent one is the final extracted watermark. Recall that during watermark embedding, we embed the same digit into multiple groups. Horizontally, watermark is embedded in multiple attributes of the same tuple. Vertically, all tuples are divided into several groups, and each group records a watermark. Among all the candidate extracted results, the value with the most occurrences are the extracted watermark.

V. Implementation

Implementation Environment. We use the Windows 10 operating system and JVM version 1.8.0 as the development environment. The system has a 2.50GHz Intel i5 processor and 8GB main memory. We implement HOPE and HOPE-L using the Java programming language. The backend database is the MySQL 5.7 database, where we use the JDBC APIs to connect to the database.

Implementation Sketch. In our implementation, HOPE is used for the entire database encryption and decryption process. After HOPE, all data is encrypted into ciphertext and datebase is added mark attribute columns to store OPC. Many SQL languages are no longer executable because of the ciphertext. Therefore, we also implement the conversion of the SQL statement so that it can be executed on the encrypted database and return the encrypted query result. On the database encrypted by HOPE, `SUM`, `COUNT` and some comparison operations can be performed.

SQL Operations on Encrypted Data. Next we explain the implementation details of SQL operations on encrypted database. SQL operations can be divided into two categories: comparison operations and non-comparison operations.

For SQL statements with non-comparison operations, due to that fact that HOPE and HOPE-L maintain the homomorphic property, typical operations such as `SUM` can be directly performed. The result is the same as the result obtained when each encrypted data is retrieved, decrypted, and then applied in the corresponding operation. Other operations such as `COUNT` can also be directly performed on encrypted data as well.

For SQL statements with comparison operations, we have to locate the corresponding object in the database and set the comparison object value to a. Due to the nature of the order-preserving encoding, the root node's OPC value (converted into decimal) is 2^{L-1}. Then compare a to 2^{L-1}. If $a < 2^{L-1}$, then a is in the left subtree, and the value of the root node is updated to $2^{L-1} - 2^{L-2}$; otherwise, if $a \geq 2^{L-1}$, meaning that a is in the right subtree, and the value of the root is updated to $2^{L-1} + 2^{L-2}$. Repeatedly, a is further compared with the root value of the subtree. The recursive process continues until the corresponding value of a is located in the database, or the maximum value less than a is located. After locating the corresponding value, it returns the OPC corresponding to a. In the original SQL statement, a is changed to the corresponding binary form. Since the order of the data can be identified by the order of OPC, the original comparison operation can be converted into an operation comparing a against the corresponding OPC columns.

VI. Evaluation

A. Evaluation Methodology and Metrics

HOPE-L is a lossless database watermarking method by combining homomorphic encryption, order-preserving encryption and information hiding. We mainly evaluate the advantages of HOPE-L from four perspectives: functional completeness, usability, efficiency, and robustness.

Functional Completeness. All key functions of HOPE-L, including the HOPE encryption, databased warkmarking embedding and extraction, can performed smoothly and seamlessly in the implemented database environment.

Usability. After applying HOPE-L, the database can still execute basic SQL queries that are fundamentally supported by HOPE. Note that such SQL queries are applied to the encrypted database. We present the result of a typical SQL query that needs to be compared. The results of other SQL queries are omitted due to space limitation.

Efficiency. HOPE-L introduces multiple computationally expensive operations. Such operations mainly include database encryption, database watermark embedding, database watermark extraction, query on encrypted database, and database decryption. We evaluate the execution time incurred by these operations, and quantify the time-efficiency of HOPE-L.

Robustness. Robustness is a desirable property of database watermarking schemes, as normal SQL operations may unintentionally interfere with embedded watermarks. More importantly, attackers may use these normal SQL operations to destroy the watermarks. Therefore, it is vital for database watermarking schemes to make watermark robust against common database attacks.

There are 6 common database attacks that cause damage to database watermark, including the subset deletion attack, subset addition attack, subset modification attack, conspiracy attack, hybrid attack, and reversible attack. We use both analytical and experimental approaches to evaluate the degree of HOPE-L's robustness and its resistance against common attacks on database watermark.

In particular, we define the character error probability CEP of the extracted the watermark as a metric for evaluating the robustness. CEP quantifies the ratio of error characters to the entire watermark, as shown in Eq. (6), where N represents the number of bits of the watermark extraction error character, and L represents the length of the watermark.

$$CEP = \frac{N}{L} \tag{6}$$

B. Completeness

We create a database table named 'data' in a database named 'sqltestdb'. The original plaintext data in the table is illustrated in Table III, where we show only the first 500 rows of tuples. Note that for simplicity, the tuples in the table 'data' consists of 5 attributes: `ID, x, y, z, total`.

TABLE III: Original database

ID	x	y	z	total
1	90	90	90	270
2	80	80	80	240
...	...	...	...	...
500	92	71	68	231

Database Encryption. The process of database encryption (by applying HOPE-L) involves construction of an OPC for the numerical columns, by applying the order-preserving encryption function `OPC`, and encryption of the plaintext data, by applying the Paillier homomorphic encryption function.

We show the results after encryption in Table IV. Note that in the tuple data resulted by the homomorphic encryption, each data becomes a large decimal integer. For brevity, we omit the digits in the middle of the each large decimal integer.

In Table IV, the upper part consists of the encrypted result of the original plaintext data, and the lower part consists of the order-preserving codes for the corresponding data. These codes correspond to the numerical order of the plaintext data.

TABLE IV: Encrypted database

ID	e_x	e_y	e_z	e_{total}
1	$62 \cdots 5$	$27 \cdots 5$	$19 \cdots 9$	$29 \cdots 8$
2	$96 \cdots 3$	$54 \cdots 5$	$58 \cdots 2$	$28 \cdots 4$
...	...	...	...	...
500	$55 \cdots 3$	$13 \cdots 3$	$25 \cdots 6$	$19 \cdots 0$
ID	c_x	c_y	c_z	c_{total}
1	10010000	110000100	111000000	111101000
2	100100100	100000000	101100000	101010000
...	...	...	...	...
500	110100000	010010000	011011000	100010000

Recall that for any give $d \in \mathcal{D}, e_d = E_k(d)$ where E is the Paillier encryption function, and that $c_d = \texttt{OPC}(d)$.

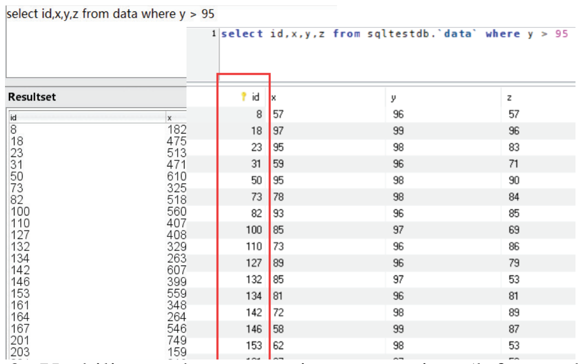
Watermark Embedding. Recall that in HOPE-L, we apply the Hash algorithm on the the database name, database table name and the currently user ID to generate a decimal watermark. In this process, the key K is required when applying the Hash function. The user has to select a specific key K for the embedding process and this key will also be used in the process of watermark extraction. Table V illustrates the encrypted database after watermark embedding.

TABLE V: Encrypted database after embedding watermark

ID	x	y	z	total
1	$292 \cdots 6670$	$404 \cdots 2507$	$439 \cdots 9698$	$252 \cdots 3319$
2	$363 \cdots 9007$	$568 \cdots 1816$	$581 \cdots 5642$	$577 \cdots 3419$
...	...	...	...	...
500	$259 \cdots 3841$	$959 \cdots 3636$	$429 \cdots 7367$	$229 \cdots 2391$

Watermark Extraction and Decryption. The process of extracting the hidden watermark of the database is opposite to the watermark embedding process.

By applying the extraction and decryption function in HOPE-L, we find that the watermark is the same as embedded, and that the decryption results are consistent with those in Table III (we omit including the results here as they are the same as those in Table III). Therefore, we conclude that the whole database encryption and decryption, watermark embedding and extraction process does not introduce any distortion into the database.



```
select id,x,y,z from data where y > 95
```

Resultset

id	x
8	182
18	475
23	513
31	471
50	610
73	325
82	518
100	560
110	407
127	408
132	329
134	263
142	607
146	399
153	559
161	348
164	264
167	546
201	749
203	159

```
select id,x,y,z from sqltestdb.`data` where y > 95
```

id	x	y	z
8	57	96	57
18	97	99	96
23	95	98	83
31	59	96	71
50	95	98	90
73	78	98	84
82	93	96	85
100	85	97	69
110	73	96	86
127	89	96	79
132	85	97	53
134	81	96	81
142	72	98	89
146	58	99	87
153	62	98	53

Fig. 2: Usability under comparison operation (left: result on encrypted database, right: result on plaintext database).

C. Usability

HOPE-L allows to perform SQL queries on an encrypted database, as described in Section V. We use the database shown in Table III as an exmaple to demonstrate that SQL

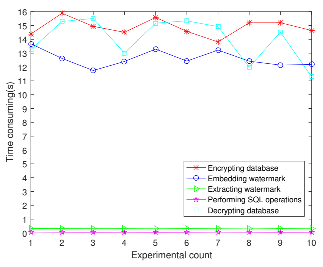


Fig. 3: Time-efficiency experiment.

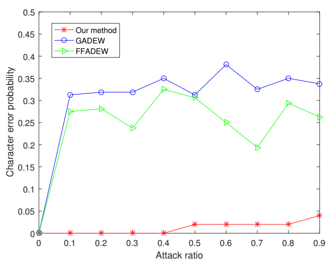


Fig. 4: Results on subset insertion attack.

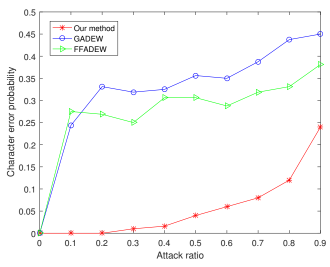


Fig. 5: Results on subset deletion attack.

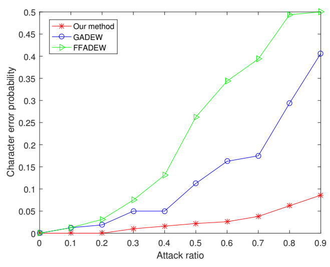


Fig. 6: Results on subset modification attack.

queries involving comparison operations can be successfully performed on an encrypted database. Other opertaions (*e.g.*, equivalence matching) can also be performed; however, due to space limitation, we omit their results.

The example SQL query we use is "`select id, x, y, z from data where y > 95`", which involves a typical numerical comparison operation. Fig. 2 compares the execution results of SQL query on the encrypted database (the left half of Fig. 2) and on the original plaintext database (the right half of Fig. 2). Both query selects only the tuples in which the y value is greater than 95. Fig. 2 shows that the query results are the same. Therefore, HOPE-L still guarantees that usability of the database.

D. Time-efficiency

HOPE-L involves 5 time-consuming phases, namely, database encryption, watermark embedding, watermark extraction, database query, and database decryption. We take 5000 database tuples, run all of the five phases multiple times. The results are shown in Fig. 3.

As can be seen from Fig. 3, database encryption, embedded watermarking and database decryption take relatively more time, but on the whole, the five stages are time-efficient. Both database encryption and decryption contain homomorphic operations, which take a long time. When embedding watermark, we keep multiplying the encryption result by the $E(0)$. Constantly doing large integer multiplication operations makes the overall calculation larger. However, the time consumption is kept within 16s, the time overhead is small, and the efficiency is high, which can meet the people's use requirements.

The extraction operation directly extracts the value of the corresponding position, and them performs voting analysis. None of them go through complex operations. The time overhead of the operation is small, about 0.3s. When performing a query operation, HOPE-L converts the corresponding query attribute name to the mark attribute name, and find the OPC in the mark attribute column, which is similar to the indexed query. The result shows that average time consumption is less than 0.1s, and the execution is very efficient.

E. Robustness

There are 6 common database attacks, as described in Section VI-A. We analyse and evaluate each of them individually. We compare HOPE-L against two existing database watermarking methods: GADEW [22] based on differential extension and genetic algorithm, and FFADEW [23] based on differential expansion and firefly algorithm. In the experiments, we adjust the attack ratio to evaluate the character error probability, by which we are able to quantify the robustness against attacks. However, only three attacks can cause damage to HOPE-L, namely, subset modification attacks, subset insertion attacks and subset deletion attacks.

Subset Insertion Attack. When extracting the watermark, we first sort the tuples according to the primary key of the database table. The primary key of the newly inserted tuples are accumulated and sorted later. The original database tuples, the primary key and key K have not changed. Therefore, the final watermark can still be extracted correctly. When a new tuple is inserted and entered into a watermark group, it may bring a wrong extraction result, but the proportion is tiny. Only when new tuples are added to the same group, and the watermark obtained from the extraction digits are same, the number of tampered watermarks can be higher than that of original watermark. Further the extraction results will be affected. Therefore, subset insertion attack has a smaller impact on watermark in a 100% attack.

The comparison experiments of the three algorithms are shown in Fig. 4. The two advanced watermarking technologies have a high probability of error. However, HOPE-L only has a slight error after reaching the attack percentage of 40% which also shows that our method is robust.

Subset Deletion Attack. Subset deletion attacks can be divided into two types: horizontal attack and vertical attack, where the attacker may randomly delete some rows (horizontal attack) or columns (vertical attack) of the database.

In the vertical attack experiments, we vary the attack ratio from 0% to 90%. We randomly delete database tuple information and compute the CEP of the extracted watermark. Random deletion of the tuples can directly delete some watermarks. The comparison results are shown in Fig. 5. We observe that when the deletion ratio is less than 30%, the watermark of HOPE-L can be extracted correctly. Then, as the ratio increases, CEP is growing substantially, but our method still performs better than GADEW and FFADEW in deletion scenario.

In the horizontal attack experiments, there are four attribute columns in the experiment database. After HOPE, there are four new attribute columns for OPC, which called mark attribute. Any mark attribute column is deleted without any influence (these columns don't contain the watermark). Any original attribute column is deleted, only a quarter of watermark copy is deleted, and other columns still retain complete watermark. Thus, randomly deleting columns in original database, it has little effect on the extraction of watermark.

Subset Modification Attack. As with the previous two experiments, we vary the attack ratio to quantify the impacts of this attack. The results are shown in Fig. 6. We observe that when the attack ratio is within 20%, no matter how the data is modified, there is no error when extracting watermarks. We further investigate the details, which shows that the same watermark has been embedded 80 times in total. After the subset modification attack, the information extraction of individual watermark bits is reduced to 62/80, meaning that 18 bits of information has been tampered. However, after the voting mechanism, there is no bit error in the extracted watermark, and CEP is 0. Until the attack ratio of modified tuple reaches 30%, the error occurs again and CEP increases to 0.01.

Under the modification attack of these proportions, we test the CEP of extracting watermark and We also make comparisons against the two existing algorithms, GADEW and FFADEW, in Fig. 6. When the attack ratio of these two algorithms reaches 30%, CEP has a significant growth. The red broken line is the experimental result of HOPE-L , When the attack rate reaches 90%, CEP is still less than 0.1. HOPE-L is more robust than the two existing algorithms obviously.

Conspiracy attack. Due to the indistinguishability of ciphertext in homomorphic encryption, the two HOPE encryption results of the same data are different. After encrypting the same database and watermark, there are two completely different databases, which cannot be statistically analyzed.

hybrid attack. Although there are similar databases, they uses different keys during encryption. A hybrid database will be chaotic, and it is difficult to get meaningful watermark information. At the same time, it is hard to obtain meaningful data through decryption. Therefore, it is meaningless for attackers to recombine databases.

Reversible attack. The watermark of HOPE-L is generated by hash algorithm of database information, which can authenticate copyright ownership. Therefore, random watermark by reversible attack has no practical significance.

VII. Conclusions

In this paper, we propose and implement HOPE-L, a lossless database watermarking method in homomorphic encryption domain, by combining database encryption and information hiding technologies. HOPE-L is the first lossless database watermarking method by combining homomorphic encryption, order-preserving encryption and information hiding. HOPE-L can protect the privacy and copyright of database content simultaneously. The watermark can authenticate the copyright, and the recipient can recover the original database without loss. The experimental results and analysis show that HOPE-L has more embedding space and no distortion. The embedded watermark has strong robustness, and the whole operation process is time-efficient. In particular, we compare HOPE-L against existing reversible database watermarking algorithms (GADEW and FFADEW). The results show that HOPE-L is superior to both algorithms in robustness and can resist common database attacks.

References

[1] Y. C. Liu, Y. T. Ma, H. S. Zhang, D. Y. Li, and G. S. Chen, "A method for trust management in cloud computing: Data coloring by cloud watermarking," *International Journal of Automation and Computing*, vol. 8, no. 3, pp. 280–285, 2011.

[2] H. Hacigumus, B. Iyer, and S. Mehrotra, "Providing database as a service," in *Proceedings 18th International Conference on Data Engineering*, 2002.

[3] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On databanks and privacy homomorphism," *Foundations of Secure Computation*, 1978.

[4] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[5] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, 1999.

[6] C. Gentry, "Fully homomorphic encryption using ideal lattices," *Proceedings of the Annual ACM Symposium on Theory of Computing*, vol. 9, pp. 169–178, 01 2009.

[7] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 563–574.

[8] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *International Conference on Advances in Cryptology-eurocrypt*, 2009.

[9] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," in *Acm Symposium on Operating Systems Principles*, 2011.

[10] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *Security and Privacy (SP), 2013 IEEE Symposium on*, 2013.

[11] M. Goljan, J. J. Fridrich, and R. Du, "Distortion-free data embedding for images," in *Information Hiding, 4th International Workshop, IHW 2001, Pittsburgh, PA, USA, April 25-27, 2001, Proceedings*, 2001.

[12] J. Tian, "Reversible data embedding using a difference expansion," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 13, no. 8, pp. 890–896, 2003.

[13] A. V. Leest, M. V. D. Veen, and F. Bruekers, "Reversible image watermarking," in *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, 2003.

[14] D. M. Thodi and J. J. Rodriguez, "Prediction-error based reversible watermarking," in *Image Processing, 2004. ICIP '04. 2004 International Conference on*, 2004.

[15] G. Xuan, J. Zhu, J. Chen, Y. Q. Shi, Z. Ni, and W. Su, "Distortionless data hiding based on integer wavelet transform," *Electronics Letters*, vol. 38, no. 25, pp. 1646–1648, 2003.

[16] R. Agrawal and J. Kiernan, "Chapter 15 - watermarking relational databases," in *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, P. A. Bernstein, Y. E. Ioannidis, R. Ramakrishnan, and D. Papadias, Eds. San Francisco: Morgan Kaufmann, 2002, pp. 155 – 166.

[17] R. Sion, M. Atallah, and S. Prabhakar, "Rights protection for relational data," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, pp. 1509– 1525, 01 2005.

[18] R. Halder and A. Cortesi, "A persistent public watermarking of relational databases," in *Information Systems Security-international Conference*, 2010.

[19] M. Shehab, E. Bertino, and A. Ghafoor, "Watermarking relational databases using optimization-based techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 1, pp. 116–129, 2007.

[20] D. Gross-Amblard, "Query-preserving watermarking of relational databases and xml documents," *Acm Transactions on Database Systems*, vol. 36, no. 1, pp. 1–24, 2011.

[21] Y. Zhang, B. Yang, and X. M. Niu, "Reversible watermarking for relational database authentication," *Journal of compute*, vol. 17, no. 2, pp. 59–65, 2006.

[22] G. Gupta and J. Pieprzyk, "Reversible and blind database watermarking using difference expansion," in *Proceedings of the 1st International Conference on Forensic Applications and Techniques in Telecommunications, Information, and Multimedia and Workshop*, M. Sorell, Ed. Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 1–6.

[23] Imamoglu, Mustafa, Bilgehan, Ulutas, Guzin, and Mustafa, "A new reversible database watermarking approach with firefly optimization algorithm," *Mathematical Problems in Engineering: Theory, Methods and Applications*, vol. 2017, no. Pt.3, p. 1387375.1, 2017.

[24] X. S. Jun, H. J. Yong, and J. University, "Database authentication watermarking algorithm in order preserving encrypted domain," *Journal of Software*, 2018.