A Random Reversible Watermarking Schefme for Relational Data

Qiang Liu[1,2,3], Hequn Xian[1,2,3*], Jiancheng Zhang[3,4] and Kunpeng Liu[4]

[1]College of Computer Science and Technology, Qingdao University, Qingdao, China.
[2]State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
[3]Shandong Computer Science Cente, Jinan, China.
[4]Shandong Zhengzhong Information Technology Co, Jinan, China.

*Corresponding author(s). E-mail(s): xianhq@126.com.com;
Contributing authors: qduliuq@163.com; zhangjc@sdas.org;
liukp@sdas.org;

Abstract

In the era of Big Data, relational data is at risk of piracy and misuse when distributed, shared and used. The use of digital watermarking technology is a reliable way to protect the copyright of relational data. In order to protect the copyright of relational data and recover the original data, many reversible watermarking schemes have been proposed in recent years, but most of them cannot extract the watermark information completely under severe attacks. To address this problem, a randomized reversible watermarking scheme is proposed.Watermark embedding algorithm, watermark integrity checking algorithm, watermark detection algorithm and data recovery algorithm were designed. The watermark capacity is increased by embedding multiple watermarks in selected tuples, and the randomness of the watermark information distribution is increased by embedding unequal proportions of watermarks in different tuples. In extracting the watermark, the attacked bits are discarded to improve the accuracy of watermark detection. In addition, only a partition with complete watermark information is selected for watermark extraction. This not only improves the speed of watermark extraction,

but also avoids the risk of key leakage from other partitions. The experimental results show that the complete watermark information can be extracted even when more than 90% of the tuples are under attack.

Keywords: relational data, reversible watermark, copyright, multiple verification

1 Introduction

At present, due to the widespread use of big data and cloud computing, there is an increasing amount of data available on the Internet[1]. These data are circulated and used in various forms on the Internet, such as audio, video, images, text and relational data. This data is constantly creating new value in the process of circulation and use. Data can be easily copied, modified and distributed through public channels, which makes it more easily available for misuse. In the Internet age, information misuse has become a more frequent data security matter than information corruption or leakage[2]. How to protect data security and prove data ownership in an open and shared environment has become an urgent issue.

Digital watermarking is a technique used for copyright protection and prevention of data tampering in multimedia data, such as images, audio, video, natural languages, and relational databases[3–7]. Database watermarking is a technique that has been proposed recently for database copyright protection. A data owner embeds a specific message into separate and discrete relational data, embeds his unique identification into the data by a watermark embedding algorithm before distributing the data, and then distributes the data through a public channel. After obtaining the data, a malicious attacker will use various attacks to destroy the original watermark in the data in order to interfere with the proof of copyright. Common attacks include tuple addition, tuple alteration and tuple deletion attacks. The data owner extracts a unique copyright mark from the stolen data for the purpose of copyright proofing.

The criteria for evaluating database watermarking schemes usually include robustness, embedding capacity and imperceptibility. The robustness of a scheme refers to the ability of the data owner to extract watermark information from the data even after the watermark has been subjected to multiple attacks; the embedding capacity refers to the number of bits of watermark that can be embedded in a given amount of data; and imperceptibility usually refers to the fact that the added watermark information does not cause significant distortion to the data, i.e. the user does not experience any change before or after the watermark. These three metrics influence and constrain each other. For example, the fewer the number of watermark bits embedded in the data, the greater the imperceptibility of the watermark; and the fewer the number of watermark bits, the greater the likelihood that an attacker will corrupt the watermark through an attack, and the less robust the watermarking scheme.

The robustness of watermarking can be enhanced by simply increasing the watermarking capacity, but this will result in a severe degradation of the imperceptibility of the watermark. For this purpose, we propose a dynamic, randomly distributed watermarking scheme for relational data, building on existing schemes. Without changing the watermarking capacity, we embed the watermarked bits into the relational data in a more randomly distributed manner, embedding multiple watermarks in selected tuples, and embedding an unequal number of watermarks in each tuple. This approach improves the watermark capacity and watermark robustness. The scheme includes watermark embedding, watermark detection, watermark integrity detection algorithm and data recovery algorithm.

The main contributions of this paper are: (1) Improving the randomness of watermark embedding and increasing the attack resistance of watermark by embedding an unequal number of watermark information in different tuples. (2) By randomly selecting different proportions of multiple attributes on different tuples to embed the watermark, we increase the randomness of the watermark distribution. In addition, we improve the accuracy of watermark detection, which is achieved by discarding the attacked bits during the watermark extraction process. Moreover, we select partitions with complete watermark information for watermark extraction, which enhances the performance of the detection algorithms.

2 Relation work

In the past, researchers have used watermarking techniques to protect the copyright of various multimedia data, e.g. images, video and audio. Based on the digital watermarking of multimedia data, researchers have proposed digital watermarking schemes for relational data, taking into account the characteristics of relational data.

In 2002, Agrawal and Kiernan proposed the first watermarking scheme for relational data[7]. The authors used a key to select special bits of certain attributes and embed some special values into them, which together form the watermark. After that, Sion et al. proposed a different watermarking scheme[8]. This scheme uses a key to rearrange and repartition the tuples to embed the watermark information by changing the distribution characteristics of the data. However, this scheme has poor resistance to tuple deletion attacks. In 2003, Prof. Niu et al. proposed a scheme to embed meaningful strings into data[9]. This scheme embeds a matching relation in the selected attribute value of the selected tuple, and the value of the watermarked bits is confirmed by verifying the existence of the matching relation when detecting the watermark. In 2008, Shehab et al. reduced digital watermarking to an optimization problem with constraints and proposed a digital watermarking scheme using genetic algorithms to reduce data distortion[10]. In 2009, Xiang et al. designed a relational database watermarking scheme which can resist primary key attacks. The

authors used Hemming codes and majority voting to eliminate the interference caused by the attack and enhance the robustness of the scheme.

These intentionally introduced special values will inevitably cause a certain degree of distortion to the data and in some cases will not meet the usability requirements. In 2006, Zhang et al. proposed the first reversible relational database watermarking scheme(HSW), which constructs histograms by exploiting the differences between attribute values and extends the histogram technique to achieve reversibility of database watermarking[11]. In the same year,Zhang et al. designed a reversible watermarking scheme for relational data by exploiting the reversible nature of exclusive or operations[12]. However, this technique cannot be used against attacks which target large numbers of tuples. In 2008, Gupta and Pieprzyk used the differential extended watermarking technique DEW to achieve reversible watermarking of relational data[13], but the robustness of this scheme is poor. In 2010, Farfoura and Horng proposed a prediction error extended watermarking technique (PEEW)[14], where the authors used a predictor to select the watermarked bits and features in the embedded data. In 2013, K. Jawad et al. first used genetic algorithms in database watermarking and designed a reversible watermarking schemebased on genetic algorithms and differential extended watermarking techniques (GADEW)[15]. The scheme uses genetic algorithm to select the optimal attributes to reduce data distortion and increase the watermarking capacity, which improves the robustness of the scheme. In 2015, Iftikhar et al. used genetic algorithms and a data analysis method from information theory to deal with the watermarking problem (RRW)[16]. They used genetic algorithms to generate optimal watermarks to reduce data distortion. However, the generation of the optimal watermark requires a large amount of computation time, and therefore the efficiency of the algorithm is too low when dealing with large amounts of data. In the same year, Franco-Contreras J et al. proposed a robust watermarking algorithm based on circular histograms by using circular histograms to modify data in plain text domains[17]. In 2017, Imamoglu M B et al. proposed a new reversible watermarking scheme for relational data (FFADEW) by combining differential extension techniques and the Firefly algorithm (FFA)[18]. The Firefly algorithm is another evolutionary algorithm that the authors use to select the optimal attribute value pairs to achieve minimal distortion. In 2018, Hu et al. proposed a genetic algorithm based histogram shift watermarking scheme (GAHSW)[19]. The authors used a genetic algorithm to partition the tuples and then used a histogram shift method to embed the watermark. This method improves the robustness of watermarking while reducing data distortion. In 2019, Li et al. proposed a low-distortion reversible database watermarking method based on histogram gaps (HGW)[20]. The method reduces data distortion without weakening the robustness of watermarking. Compared to GAHSW, this method reduces data distortion without weakening watermarking robustness. In 2020, Li et al. improved the histogram shifting scheme by proposing a non-redundancy shifting-based method. It changed the histogram shifting method to reduce

the distortion caused by watermarking and slightly improve the usability of the data[21].In 2022, Xiang et al. proposed a robust watermarking algorithm based on order-preserving encryption and circular histograms[22]. Based on the circular histogram watermarking technique, the authors used order-preserving encryption (OPES) to encrypt the data, avoiding the risk of privacy leakage without affecting the database in normal access.

The above digital watermarking schemes for relational data show that researchers in relational data are striving to find a balance between the three evaluation criteria of watermarking, improving the robustness and embedding capacity of watermarking, and reducing the impact of watermarking on data, i.e., improving the imperceptibility of watermarking. In this paper, we propose a dynamic, randomly distributed watermarking scheme for relational data that improves the robustness of watermarking without changing the watermarking capacity, to address the problem that the robustness of existing watermarking schemes and the watermarking capacity cannot be combined.

3 Preliminaries

3.1 XOR Encryption

The Exclusive Or(xor) operation is a common tool in cryptography and his mathematical notation is $\oplus$. It is often used in encryption algorithms because this operation is reversible, i that is:

$$A \oplus B = M, M \oplus B = A \tag{1}$$

In Equation 1, A is the plaintext, B is the key, and M is the ciphertext. Based on this feature, the owner of the data can combine the watermark information with the original data for the purpose of embedding and extracting the watermark in the database. Furthermore, as the operation is reversible, the original data can be recovered by simply repeating the watermark embedding operation.

3.2 One-way hash function

The one-way hash function $H()$ is a tool in cryptography for processing a message M of any length into a value h of fixed length, i.e. $h = H(M)$. It has the following characteristics: i) given M, it is easy to compute h, ii) given h, it is hard to compute M such that $H(M') = h$, and iii) given M, it is hard to find another message M' such that $H(M) = H(M')$. SHA is currently a dominant one-way hash function.

The Message Authentication Code (MAC) is a one-way hash function that depends on a key. Let $H(K, PK)$ be a MAC, a box In our scheme, we use the primary key r.key of the tuple and the key as input to the MAC function, and the resulting result is used to determine the location of the watermark. The

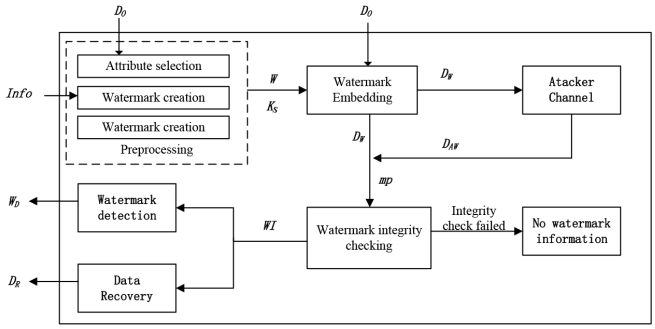

Fig. 1: System architecture

specific function used is:

$$H(K, PK) = H(PK|H(K|PK)) \tag{2}$$

where '|' represents concatenation.

4 SCHEME

This section describes a reversible watermarking scheme that can be used on relational databases. The main structure of the scheme is shown in Figure 1.

The scheme mainly includes the following five main stages: (1) preprocessing stage; (2) watermark embedding stage; (3) watermark integrity detection stage; (4) watermark extraction stage; (5) data recovery stage. The watermark preprocessing stage is mainly to do some preparation work before embedding. The first step is to encode the copyright identifier of the data owner into a watermark string that can be embedded in the database.The second step is to filter the attribute columns that are suitable for embedding the watermark. The last step is to generate a random function that will be used to determine the proportion of watermarks embedded in each tuple. The main task of the watermark embedding phase is to embed the watermark information into the data according to the key, to obtain the database with the watermark, and to return the auxiliary data. The watermark integrity check phase is to check the water-mark integrity of each partition. The watermark extraction phase extracts the watermark information from the stolen watermarked data to achieve the problem of copyright proof. The data recovery phase removes the watermark from the data and recovers the original data.

Definition 1. Watermark Integrity,WI(Watermark Integrity), is an indication of the degree of integrity of the watermark for each partition, and to a certain extent reflects the degree to which the data has been compromised.to

a certain extent, reflects the degree to which the data has been compromised. $WI = [f_1, f_2, f_3, , f_N](f_i \in 0, 1)$.

We will select only one of the partitions with a full watermark for detection during watermark extraction. The use of watermark completeness improves the efficiency of watermark detection and ensures that the data owner can still extract the complete watermark from the data in the case of a serious attack on the database.

4.1 Preprocessing

Before embedding the watermark, the data needs to be preprocessed. The main tasks in the preprocessing phase are: (1) selecting the appropriate attributes for watermark embedding; (2) creating watermark information for embedding; and (3) determining a random function that determines the proportion of attributes to be watermarked for each tuple.

4.1.1 Attribute selection

Not all attributes are suitable for embedding attributes, and for some attributes, small changes can have a significant impact on data quality. We first select multiple attribute columns from the database as candidate attribute columns that can be used as identifiable features, and embedding a watermark in an attribute has less of an impact on data quality. The candidate attributes are then reordered and numbered in ascending order, and the reordering enhances robustness against attribute attacks.

4.1.2 Watermark Creation

Watermark information is not only a carrier of the data owner's copyright information, but also evidence of the owner's copyright claim. The unique identification information is converted into a binary sequence and embedded as a watermark in the data. We choose to embed the same watermark in different partitions, so that only the partition with the most complete watermark is selected for copyright declaration when the watermark is detected. The watermark generation formula is as follows:

$$W = Info \text{ xor Rand} \tag{3}$$

$Info$ is the identifier of the data owner and *Rand* is the random number generated. By encrypting $Info$ with an alias, the data owner's information can be effectively prevented from being leaked.

4.1.3 Determine random function

we embed an unequal number of watermarks in each tuple, the number of watermarks embedded in each tuple is chosen by the F function. The input to the F function is the tuple's primary key and key, and the output is a random integer of $[0, Max]$ to ensure that the number of watermarks embedded is

random. The data owner can choose any generation function, in this case the following function is chosen:

$$F(x, y, z) = H(x|H(x|y))\%z \tag{4}$$

Table 1 The meaning of the symbols in the scheme.

Symbol	Description	Symbol	Description
D_O	Original database	D_W	Databases with embedded watermarks
D_R	Restored database	D_{AW}	Databases under attack
$1/\omega$	Proportion of attribute embedding	λ	Number of data partitions
σ	Random number greater than the length of the watermark	$1/\eta$	Proportion of tuples with watermarks embedded
ξ	Length of watermarked information	ν	Number of bits in the lowest significant bit
K_P	Data partition key	$K_S[i]$	Watermark embedding key for partition i
r	Example tuple of data	A_j	The jth attribute of the data
$bit[k]$	The kth bit of the LSB	$W[l]$	kth bit of watermark information
$bitw[k]$	Extraction of the kth bit of the LSB	$W_D[l]$	The lth bit of the detected watermark
Max	Proportional limit for embedding watermark attributes	LSB	The low significant bit of the data
mp	Auxiliary data	mpw	Auxiliary data for the data to be tested
$mp[i]$	Auxiliary data for partition i	A_j	jth attribute of the data
$r.key$	Primary key for tuple r	$count[l]$	Array of majority vote marks
WI	Marker array	$bits$	Bits to be embedded
f_i	Markers for partition i	$bitsl$	Information about the watermarked bits stored

4.2 Watermark embedding phase

The main task of the watermark embedding phase is to embed the watermark in the database in an invisible way, while ensuring data availability. After embedding the watermark, the data owner stores the auxiliary data $bitsl$ and uses them during the watermark integrity check, watermark extraction and data recovery phases. The information stored in the auxiliary array mp consists of the original data bit value $bit[k]$ and the subscript position l of the watermark. $bit[k]$ is selected as shown in Figure 2, and bits are generated by selecting one of the LSB and dissociating $bit[k]$ with $W[l]$. $bitls$ are generated

by processing the bits, and the final stored value $bitls$ is calculated according to the following equation5.

$$bitsl = l + \sigma * bits \tag{5}$$

l is the embedded watermark is the subscript position of W. σ is any random number greater than the watermark length ξ. When extracting the stored information, if the value of $bitsl$ is greater than ξ , the value of bits is 1, otherwise it is 0. When the value of $bitsl$ is less than ξ, the value of l is bitsl, otherwise l is calculated by Equation 2.

$$l = bits - \sigma * bits \tag{6}$$

The specific watermarking process is shown in algorithm 1: Here we take an

Algorithm 1 Watermark embedding algorithm

Input: D, K_P, K_S, W
Output: D_W, mp

```
1: for each tuple r ∈ D do
2:     i = H(r.key, K_P, MSB) mod λ
3:     if H(r.key, K_S[i]) mod η =0 then
4:         for each A_j in r do
5:             ω=F(r.key, K_S[i], Max)
6:             if H(r.key, K_S[i], MSB) mod ⌈ω⌉ = 0 then
7:                 W_index l = H(r.key, K_S[i]) mod ξ
8:                 bit_index k = H(r.key, K_S[i]) mod υ
9:                 bits = bit[k] xor W[l]
10:                bitsl = l + σ * bits
11:                Stroe(mp[i], bitsl)
12:                Update(bit[k], bits)
13:            end if
14:        end for
15:    end if
16: end for
17: return D_W, mp
```

arbitrary tuple r as an example. Line 2 determines the partition of r. The primary key of the tuple and the key K_P are hashed and then modulo the partition of r. Line 3 determines whether the tuple is watermarked or not, and $1/\eta$ is the parameter that controls the tuple embedding ratio. Line 4 determines for each attribute whether the attribute is embedded in a watermark or not. Lines 5 and 6 generate a random number, and attributes in the tuple with a ratio of $1/\omega$ will be watermarked, with the value of ω determined by the key, the primary key of the tuple, and Max. This approach increases the

watermarking capacity and the randomness of the watermark embedding. If you do not increase the watermark capacity, simply adjust the size of the Max parameter. Line 7 selects one bit from the watermark string as the embedded bit, ν being the length of the watermark. Line 8 selects the bit position for embedding, ξ is the number of least significant bits. Line 9 combines the watermarked bits with the original information by means of an exception or operation to generate a new bit. Line 10 combines the embedded watermark bits with the subscript of the watermark where it is located to generate the auxiliary data *bitsl* with the embedded watermark and watermark position information via Equation 1. $store(mp[i], bitsl)$ means to store the *bitls* into the auxiliary array element mp[i], mp[i] represents the auxiliary data for partition i. $Update(bit[k], bits)$ in line 10 updates the kth bit of the least significant bit to bits. Finally, the algorithm returns the embedded watermarked data D_W and the auxiliary data *mp*.

4.3 Watermark integrity detection Phase

To ensure that the owner of the data can detect its watermark in case of a serious attack, we check the watermark integrity of all partitions. The watermark completeness check uses the $Check(mp_W[i], mp[i])$ function to check the watermark completeness of partition i. This function checks ξ watermark positions in the ith partition. If the function detects at least one occurrence of watermark information in ξ watermark positions, the watermark information of the partition is considered to be complete and returns 1; otherwise it returns 0. We have discarded the attacked watermark bits in the watermark extraction stage, so when each watermark bit is detected at least once in partition i, the correct result can be obtained by majority voting mechanism. When extracting the watermark, the watermark is extracted for the partition with complete watermark information. The watermark integrity checking algorithm is shown in Algorithm 2.

Lines 1 to 8 of Algorithm 2 are similar to the water-mark embedding algorithm. Only line 3 has an additional line to determine the partition i to which tuple r belongs. If the watermark integrity token $WI[i]$ of partition i is already 1, the partition is no longer checked. Line 9 calculates the value of *bitwsl* using equation 1. Lines 10-12 determine if *bitwsl* is stored in $mp[i]$, and if so, store it in $mpw[i]$. Lines 13-15 determine the length of $mpw[i]$, and if the length is an integer multiple of ξ, perform a $Check(mpw[i], mp[i])$. This can greatly improve the detection efficiency by periodically checking whether the watermark is complete. When the token array $WI[i]$ of partition i is set to 1, no subsequent tuples of that partition are checked. The algorithm only detects all tuples when the data does not contain watermark information. When all partitions have a marker array of 0, the data does not contain any copyright information.

Algorithm 2 Watermark integrity checking algorithm

Input: D_W, K_P, K_S, mp
Output: WI

for each tuple $r \in D_W$ **do**
  $i = H(r.key, K_P) \bmod \lambda$
  if $H(r.key, K_S[i]) \bmod \eta = 0$ **then**
    for each A_j in r **do**
      $\omega = \mathrm{F}(r.key, K_S[i], Max)$
      if $\mathrm{H}(r.key, K_S[i], MSB) \bmod \lceil \omega \rceil = 0$ **then**
        W_{index} $l = H(r.key, K_S[i]) \bmod \xi$
        bit_{index} $k = H(r.key, K_S[i]) \bmod \upsilon$
        $bitwsl = l + \sigma * bit[k]$
        if $bitwsl$ in $mp[i]$ **then**
          $Store(mp_w[i], bitwsl)$
        end if
        if $mpw[i].length\%\xi = 0$ **then**
          $WI[i] = Check(mpw[i], mp[i])$
        end if
      end if
    end for
  end if
end for
return WI

4.4 Watermark extraction stage

The owner of the data extracts the watermarked information from the stolen data for the purpose of copyright proof. This requires a higher degree of robustness in watermarking. Even in the case of a serious attack on the data, the data owner is still able to extract the complete watermark information from the data. We use majority voting in the watermark extraction process to reduce the impact of the attack and improve the robustness of the watermark extraction, the results of which are shown in Table 2. The watermark extraction algorithm is shown in Algorithm 3.

Table 1: Majority Voting.

Voting result	0	0	1	1
Result1	0	0	0	1
Result2	0	0	1	1
Result3	0	1	1	1

Algorithm 3 Watermark extraction algorithm

Input: D_W, K_P, K_S, mp, WI
Output: W_D

1: *get* $f_i = 1, f_i \in WI$
2: *Initialize count* $= 0$
3: **for** each tuple $r \in D$ **do**
4: $i = H(r.key, K_P)$ mod λ
5: **if** i == f_i **then**
6: **if** $H(r.key, K_S[i])$ mod η =0 **then**
7: **for** each A_j in r **do**
8: $\omega = \mathrm{F}(r.key, K_S[i], Max)$
9: **if** $\mathrm{H}(r.key, K_S[i], MSB)$ mod $\omega = 0$ **then**
10: W_{index} $l = H(r.key, K_S[i])$ mod ξ
11: bit_{index} $k = H(r.key, K_S[i])$ mod υ
12: $Get(mp[i], bitsl)$
13: $bits = (bitsl - l)/\sigma$
14: $bitws = bits$ xor $W[l]$
15: $W_D[l] = bit_W[k]$ xor $bitws$
16: **if** $W_D[l] = 1$ **then**
17: $count[l] + +$
18: **else**
19: $count[l] - -$
20: **end if**
21: **end if**
22: **end for**
23: **end if**
24: **end if**
25: **end for**
26: **for** n=0 to $\xi - 1$ **do**
27: **if** $count[n] > 0$ **then**
28: $W_D[n] = 1$
29: **else**
30: $W_D[n] = 0$
31: **end if**
32: **end for**
33: **return** W_D

In line 1, a partition with complete watermark information is selected, and only the tuples of this partition are watermarked next. Line 2 initializes the count variable. Lines 3 and 4 determine the partition to which the tuple belongs. Line 6 determines whether the tuple is a tuple with a watermark. Lines 6 to 11 search for the location of the watermark embedding. Line 12 $Get(bits, mp[i])$ extracts the *bitls* from the auxiliary data of partition i, mp[i]. Line 13 calculates the value of the bits stored in the *bitsl*. Line 14 calculates the embedded watermark information. If $WD[l]$ is 1, the count value is added to 1,

if $W_D[l]$ is 0, the count value is subtracted from 1. Lines 25 to 31 calculate the final vote result and finally return the detected watermark information W_D. W_D is encrypted and is decrypted using key to get Info. Info is the copyright information of the owner of the decrypted data.

4.5 Data recovery phase

When the data after watermarking does not meet the demand, the data owner licenses the key and supporting files to the data user and the original data is recovered using the key and data recovery algorithm. The data recovery algorithm is shown in Algorithm 4.

Algorithm 4 Data recovery algorithms

Input: D_W D_{AW}, K_P, K_S, mp, W
Output: D_R

1: **for** each tuple $r \in D$ **do**
2: $i = H(r.key, K_P) \bmod \lambda$
3: **if** $H(r.key, K_S[i]) \bmod \eta = 0$ **then**
4: **for** each A_j in r **do**
5: $\omega = \mathrm{F}(r.key, K_S[i], Max)$
6: **if** $\mathrm{H}(r.key, K_S[i], MSB) \bmod \lceil \omega \rceil = 0$ **then**
7: W_{index} $l = H(r.key, K_S[i]) \bmod \xi$
8: bit_{index} $k = H(r.key, K_S[i]) \bmod \upsilon$
9: $Get(bitsl, mp[i])$
10: $bits = (bitsl - l)/\sigma$
11: $bitwsl = l + bit[k]$
12: **if** *bits in* $mp[i]$ **then**
13: $bitws = bits$ xor $W[l]$
14: $Update(bit_W[k], bitws)$
15: **end if**
16: **end if**
17: **end for**
18: **end if**
19: **end for**
20: **return** D_R

Lines 1 to 8 are the same as the watermark embedding algorithm. Line 9 gets the value of the stored *bitsl* from the auxiliary array. Lines 11 and 12 compute the values of *bits* and *bitwsl*. If the *bitwsl* is in the auxiliary data $mp[i]$, line 13 is an alias to the bits to get the original *bitws*. Line 14 uses $Update(bit_W[s], bitws)$ to update the value of $bit_W[k]$ to *bitws*, removing the watermark information. Finally, the recovered data is returned to D_R.

5 Experimental analysis

This section evaluates various aspects of the scheme. The aim of the experiments is to verify the accuracy and robustness of the scheme. The experiments include: (1) watermark capacity and data availability experiments (2) robustness experiments, and (3) algorithm performance experiments. The experimental environment is: 2.0 GHz Intel Core CPU, 16 GB RAM, Ubuntu 20.04LTS operating system, IDEA2021.1.3 development environment, and MariaDB 10.3.29 database. The experimental data were obtained using the "Forest Cover" dataset provided by the University of California(kdd.ics.uci.edu/databases/covertype/covertype.html). The dataset contains a total of 581012 tuples and 54 attributes. We selected 100000 data items and 10 attributes from this data set and transformed them to be the experimental data. Experimental parameters: number of data tuples n = 100000, number of data partitions μ= 10, watermark tuple embedding ratio $1/\eta$= 1/5, F-function parameter Max = 5. The experimental results are the average of 5 experiments.

5.1 Watermark capacity and data availability experiment

Watermarking capacity is a measure of a water-mark's resistance to attack. The more watermarked bits of information in a given amount of data, the more resistant the watermark will be to malicious attacks. At the same time,the larger the watermarking capacity, the greater the distortion of the data.

5.1.1 Data availability experiments

Watermarking inevitably causes distortions in the data, and the larger the watermark, the more severe the distortion. Therefore, we compared the statistical information (mean and variance) of the data before and after watermarking. As shown in Table 3, the effect of water-marking on the mean value of the data is on the order of 1 in 1000. Only the A3 attribute showed the largest change, with an increase in mean value of 0.0055. The effect on variance was also on the order of 0.01, with the largest effect on the A6 attribute reaching 0.012.

5.1.2 Watermark capacity experiment

If only one bit of watermark information is embedded in each tuple, then the number of watermarked tuples is the capacity of the watermark, i.e. the watermark capacity is N/ν. Using the control variable method, Figure 2 shows the change in the number of watermarks embedded when the number of tuples is changed at $1/\eta = 1/5$. Figure 2 shows the change in the number of embedded watermark bits by varying the value of $1/\eta$ when the number of tuples is 100000. Since our scheme embeds watermarks in the attributes of the selected tuple $1/\eta$ ratio, and the value of is randoml generated by the F function. Mathematical reasoning shows that there are about N * (lnν + C)/(η*Max) number of bits of information embedded in the data, (C is the Euler constant).

Table 2: Effect on Various Statistical Measure.
(Mean μ_D, Variance σ_D, Mean μ_{DW}, Variance σ_{DW})

Features	D	DW	D	DW
A1	28.6204	28.6205	5.3535	5.3550
A2	1.3816	1.3823	1.0770	1.0782
A3	0.1180	0.1235	0.0042	0.0069
A4	2.6058	2.6063	4.1118	4.1122
A5	2.3524	2.3542	0.1817	0.1828
A6	33.4425	33.4427	315.7284	315.7401
A7	2.1823	2.1848	0.0438	0.0464
A8	2.2545	2.2567	0.0277	0.0296
A9	1.3930	1.3933	0.0972	0.0985
A10	35.8962	35.8962	317.3522	317.3572

Figure 4 shows the experimental results for the specific number of watermarks for different tuples. The experimental results are consistent with the inference results within the error tolerance.

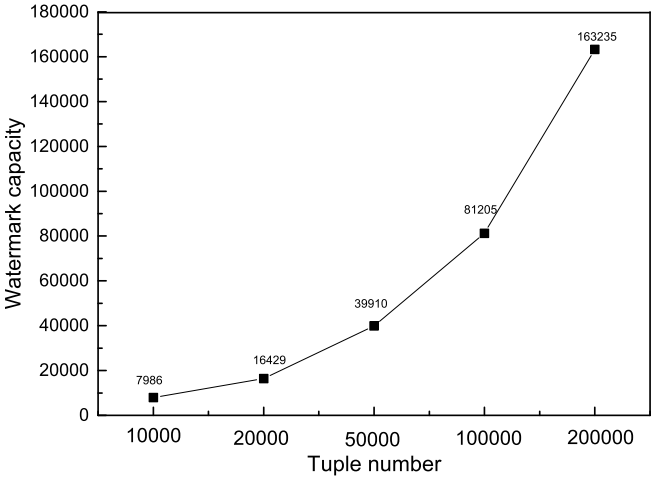

Fig. 2: Watermark capacity analysis1

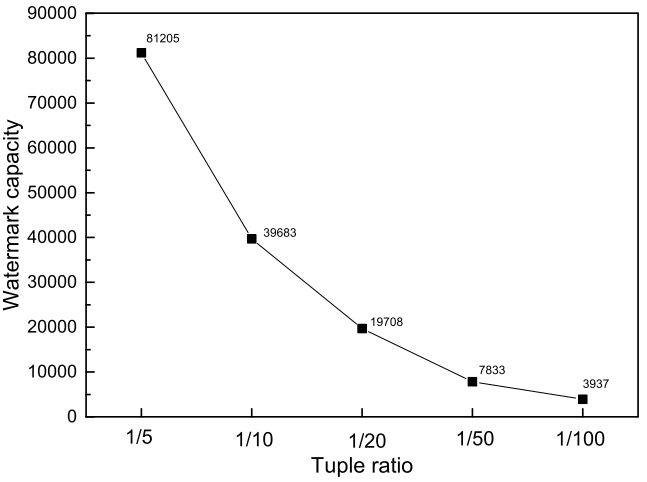

Fig. 3: Watermark capacity analysis2

5.2 Robustness experiments

In this section, a number of different types of attacks are performed on the watermarked data. These attacks include (1) tuple addition attacks, (2) tuple modification attacks, and (3) tuple deletion attacks. We also compare our scheme with several recent reversible database watermarking schemes, such as DEW, GADEW, PEEW, RRW. Since the watermark detection rate criteria of GAHSW and HGW schemes are not the same as the previous ones, this scheme cannot be compared with them.

5.2.1 Tuple adding attacks

In this type of attack, the attacker adds a number of new tuples to the watermarked data set in an attempt to interfere with the watermark detection. The attacker may insert a number of randomly generated tuples into the watermarked data.

Watermark detection: As shown in Figure 4, when the number of tuples is increased by the same amount as the original data, the detection accuracy of our scheme still remains 100%. At this point, the detection accuracy of DEW is already less than 88% and the accuracy of PEEW is only 98%.

Data recovery: As shown in Figure 5, after inserting 100% of the tuples, 100% of the watermarked data can be recovered accurately. This is because the tuple addition attack does not destroy any original data or watermark, and the hash function and auxiliary data can pinpoint the location of the added watermark.

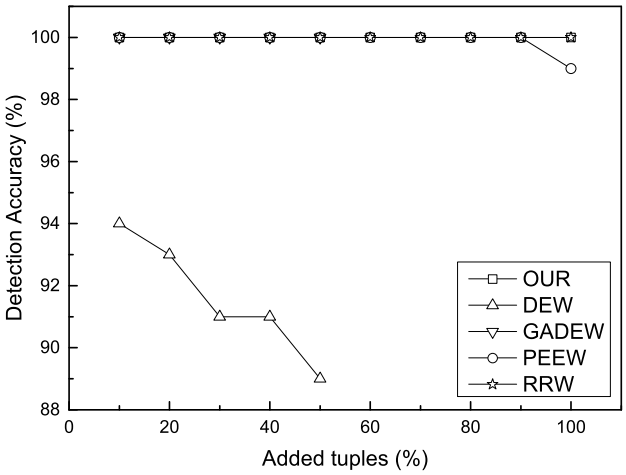

Fig. 4: Watermark detection accuracy after tuples alteration attack

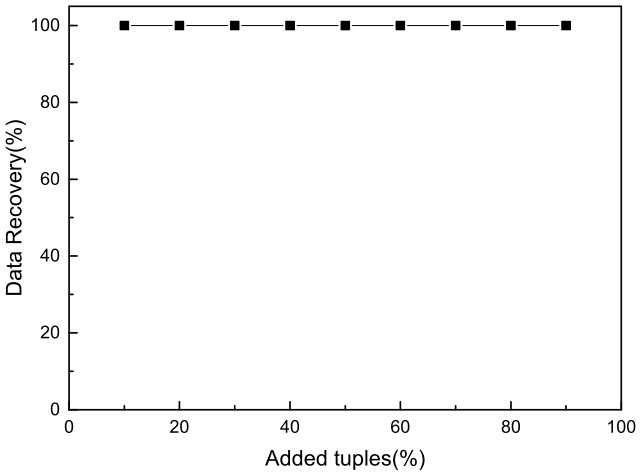

Fig. 5: Watermark detection accuracy after tuples alteration attack

5.2.2 Tuple alteration attacks

In this type of attack, the attacker changes some tuples at random. Here we perform a bit-inversion attack on all attributes of a randomly selected tuple to interfere with the watermark detection.

Watermark detection: As shown in Figure 6, when 90% of the tuples are alterated, our scheme still maintains 100% accuracy in watermark detection, while only RRW maintains 100% accuracy in the other schemes.

Data recovery: As shown in Figure 7, it is almost impossible to perform a complete data recovery on data knowing that it has been subjected to a tuple alteration attack. The experimental results are generally consistent with the results of the tuple deletion attack, and the scheme is still able to fully recover data that has not been attacked.

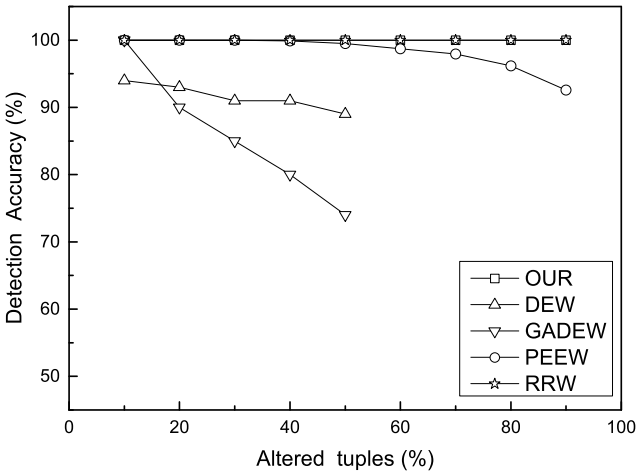

Fig. 6: Watermark detection accuracy after tuples alteration attack

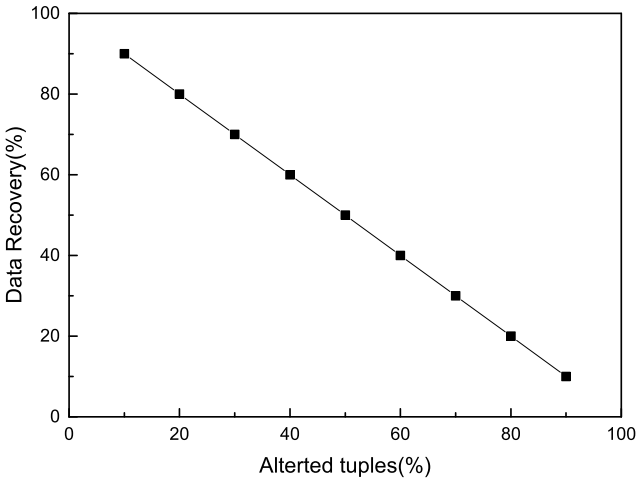

Fig. 7: Data recovery accuracy after tuples alteration attack

5.2.3 Tuple deletion attacks

In this type of attack, the attacker removes a certain number of tuples at random, trying to interfere with watermark detection by removing tuples containing watermark information.

Watermark detection: As shown in Figure 8, when 90% of the tuples are deleted, our scheme is still able to maintain 100% watermark detection accuracy. When a large number of tuples are deleted, the detection accuracy of GADEW, PEEW, GAHSW and other schemes, except RRW, decreases significantly.

Data recovery: As can be seen from Figure 9, the watermarked data can be recovered accurately. This is because the tuple deletion does not destroy the remaining part of the original data, and the remaining data can still be watermarked and restored to its original state.

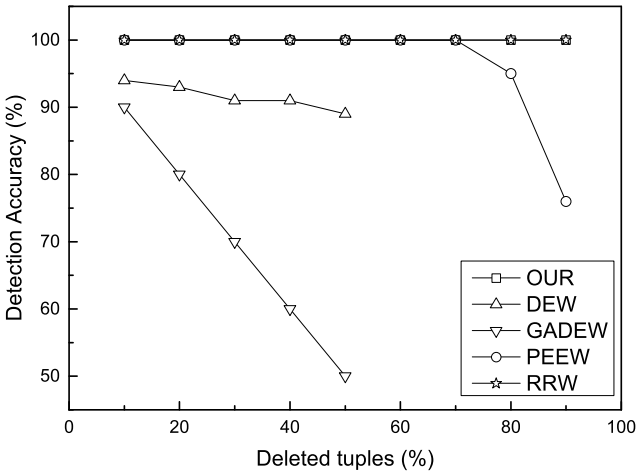

Fig. 8: Watermark detection accuracy after tuples deletion attack

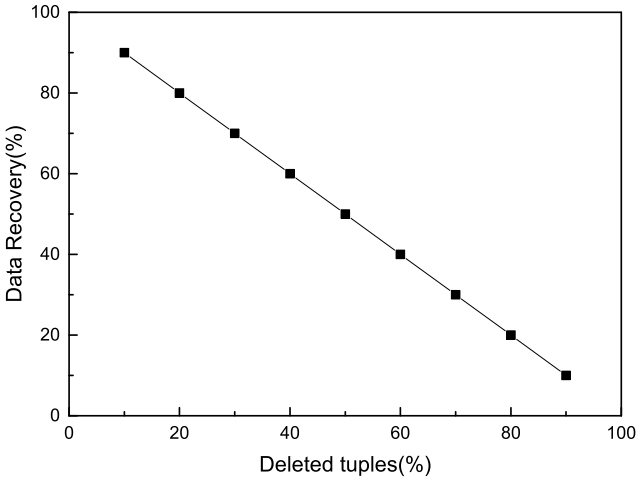

Fig. 9: Data recovery accuracy after tuples deletion attack

5.3 Performance experiments

In this section, we only compare the watermark embedding operation with the database read/write time. Since the watermark integrity checking and watermark extraction algorithms are too short, the time complexity of the data recovery algorithm is identical to that of the watermark embedding algorithm and will not be compared here. We compare the operation of reading n tuples and then updating n/5 with the watermark embedding operation. As shown in Figures 10 and 11, for tuple n = 10,000, the watermarking operation results in an additional 26% execution time compared to database reads and writes. This additional execution time is due to the JVM virtual machine initialization, data partitioning, selecting attributes, embedding the watermark and storing auxiliary data. However, this percentage of time decreases as the number of tuples increases, dropping to 14% at tuple n = 20,000. As the number of tuples continues to increase, the extra time share decreases further.

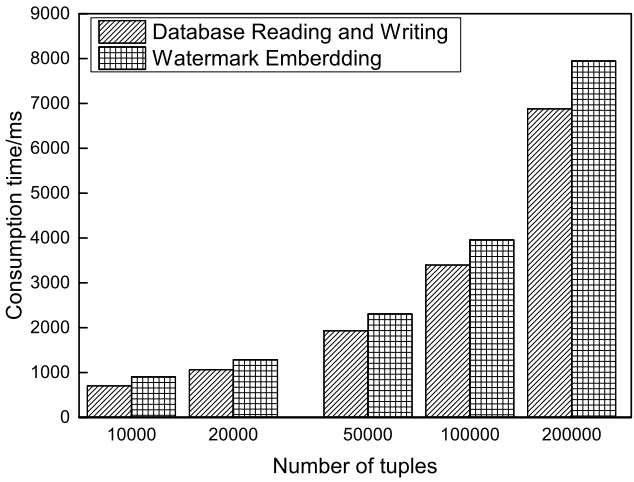

Fig. 10: Time performance analysis

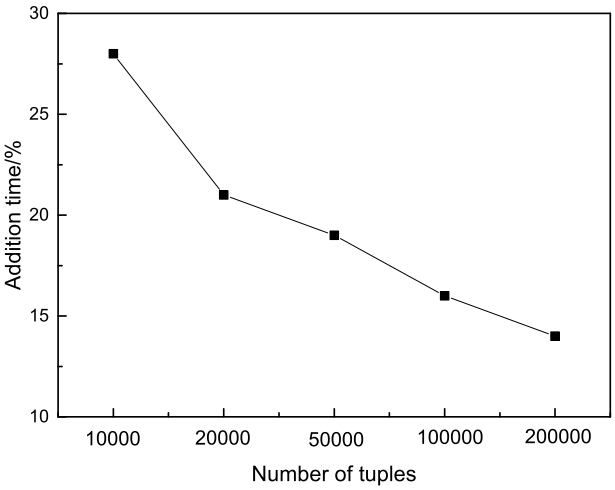

Fig. 11: Aditional time consuming analysis

6 Experimental analysis

Aiming at the problem that the watermark information extracted under serious attack is incomplete, a randomized reversible watermarking scheme based on LSB modification is proposed Watermark embedding algorithm, watermark integrity checking algorithm, watermark detection algorithm and data recovery algorithm are designed The watermark capacity is improved by embedding multiple watermarks in the selected tuples, and the randomness of watermark information distribution is increased by controlling different tuples to embed unequal proportion of watermarks When extracting the watermark, the accuracy of water-mark detection is improved by discarding the attacked bits Finally, through the experimental analysis of the algorithm performance, it can be seen that our scheme can resist serious tuple attack and the watermark capacity Compared with dew, gadew, RRW, peew and RRW, it is proved that this scheme has stronger anti attack ability and can meet the requirements of most application scenarios.

References

[1] Liu, Y.-C., Ma, Y.-T., Zhang, H.-S., Li, D.-Y., Chen, G.-S.: A method for trust management in cloud computing: Data coloring by cloud watermarking. International journal of automation and computing **8**(3), 280–285 (2011)

[2] Cheng, L., Liu, F., Yao, D.: Enterprise data breach: causes, challenges, prevention, and future directions. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **7**(5), 1211 (2017)

[3] Wong, P.W., Memon, N.: Secret and public key image watermarking schemes for image authentication and ownership verification. IEEE transactions on image processing **10**(10), 1593–1601 (2001)

[4] Saadi, S., Merrad, A., Benziane, A.: Novel secured scheme for blind audio/speech norm-space watermarking by arnold algorithm. Signal Processing **154**, 74–86 (2019)

[5] Venugopala, P., Sarojadevi, H., Chiplunkar, N.N.: An approach to embed image in video as watermark using a mobile device. Sustainable Computing: Informatics and Systems **15**, 82–87 (2017)

[6] Brassil, J.T., Low, S., Maxemchuk, N.F.: Copyright protection for the electronic distribution of text documents. Proceedings of the IEEE **87**(7), 1181–1196 (1999)

[7] Agrawal, R., Kiernan, J.: Watermarking relational databases. In: VLDB'02: Proceedings of the 28th International Conference on Very Large Databases, pp. 155–166 (2002). Elsevier

[8] Sion, R., Atallah, M., Prabhakar, S.: Rights protection for categorical data. IEEE transactions on knowledge and data engineering **17**(7), 912–926 (2005)

[9] NIU, X.-m., Zhao, L., HUANG, W.-j., ZHANG, H.: Watermarking relational databases for ownership protection. ACTA ELECTONICA SINICA **31**(S1), 2050 (2003)

[10] Shehab, M., Bertino, E., Ghafoor, A.: Watermarking relational databases using optimization-based techniques. IEEE transactions on Knowledge and Data Engineering **20**(1), 116–129 (2008)

[11] Zhang, Y., Yang, B., Niu, X.-M.: Reversible watermarking for relational database authentication. Journal of Computers **17**(2), 59–66 (2006)

[12] Zhang, Y., Niu, X.-M.: Reversible watermark technique for relational databases. ACTA ELECTONICA SINICA **34**(S1), 2425 (2006)

[13] Gupta, G., Pieprzyk, J.: Reversible and blind database watermarking using difference expansion. International Journal of Digital Crime and Forensics (IJDCF) **1**(2), 42–54 (2009)

[14] Farfoura, M.E., Horng, S.-J., Wang, X.: A novel blind reversible method

for watermarking relational databases. Journal of the Chinese Institute of Engineers **36**(1), 87–97 (2013)

[15] Jawad, K., Khan, A.: Genetic algorithm and difference expansion based reversible watermarking for relational databases. Journal of Systems and Software **86**(11), 2742–2753 (2013)

[16] Iftikhar, S., Kamran, M., Anwar, Z.: Rrw—a robust and reversible watermarking technique for relational data. IEEE Transactions on knowledge and data engineering **27**(4), 1132–1145 (2014)

[17] Franco-Contreras, J., Coatrieux, G.: Robust watermarking of relational databases with ontology-guided distortion control. IEEE transactions on information forensics and security **10**(9), 1939–1952 (2015)

[18] Imamoglu, M.B., Ulutas, M., Ulutas, G.: A new reversible database watermarking approach with firefly optimization algorithm. Mathematical Problems in Engineering **2017** (2017)

[19] Hu, D., Zhao, D., Zheng, S.: A new robust approach for reversible database watermarking with distortion control. IEEE Transactions on Knowledge and Data Engineering **31**(6), 1024–1037 (2018)

[20] Li, Y., Wang, J., Ge, S., Luo, X., Wang, B.: A reversible database watermarking method with low distortion. Mathematical Biosciences and Engineering **16**(5), 4053–4068 (2019)

[21] Li, Y., Wang, J., Luo, X.: A reversible database watermarking method non-redundancy shifting-based histogram gaps. International Journal of Distributed Sensor Networks **16**(5), 1550147720921769 (2020)

[22] Xiang, S., Ruan, G., Li, H., He, J.: Robust watermarking of databases in order-preserving encrypted domain. Frontiers of Computer Science **16**(2), 1–9 (2022)