A Random Reversible Watermarking Scheme for Relational Data

Qiang Liu[1,2], Hequ Xian[1,2(✉)], Jiancheng Zhang[3,4], and Kunpeng Liu[4]

[1] College of Computer Science and Technology, Qingdao University, Qingdao, China
xianhq@126.com
[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[3] Shandong Computer Science Center, Jinan, China
zhangjc@sdas.org
[4] Shandong Zhengzhong Information Technology Co., Ltd., Jinan, China
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 random reversible watermarking scheme is proposed. Watermark embedding algorithm, watermark integrity checking algorithm, watermark detection algorithm and data recovery algorithm are 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

There is currently an increasing amount of data on the internet due to the widespread use of big data and cloud computing [1]. This data is stored on the internet in various forms such as audio, video, images, text, and relational data. As it circulates and used, this data constantly creates new value. These data can

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 413–430, 2023.
https://doi.org/10.1007/978-3-031-25538-0_22

be easily copied, modified and distributed through public channels, which makes them more vulnerable to misuse. Information misuse has become a more frequent data security issue 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 technology used for copyright protection of multimedia data, such as images, audio, video, natural languages, and relational databases [3–7]. Database watermarking is a recently proposed technique for database copyright protection. Before distributing the data, Data owners embed their unique copyright mark into the data via a watermark embedding algorithm and then distributed the data through public channels. After obtaining the data, the malicious attacker will use various attacks to destroy the watermark in the data 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 copyright proofing.

In this paper, we propose a dynamic random distributed watermarking scheme for relational data, which improves the robustness of watermarking while reducing data distortion. Experimental results show that our scheme is better than the previous schemes in terms of robustness and data availability.

The subsequent sections of the paper are structured as follows: In Sect. 2, some related researches are provided. In Sect. 3, the proposed scheme is described in detail. In Sect. 4, experiments and results are discussed concisely. At last, the paper is concluded in Sect. 5.

2 Related Work

In 2002, Agrawal and Kiernan proposed the first watermarking scheme for relational data [7]. The author uses a key to select special bits for specific attributes and embeds some particular values forming the watermark. After that, Sion et al. proposed a different watermarking scheme [8]. This scheme uses a key to rearrange and repartition the tuples. The authors achieve the embedding of 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 tuple chosen, 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].

These intentionally introduced particular 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 that 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 scheme based on genetic algorithms and differential extended watermarking techniques (GADEW) [15]. The scheme uses a genetic algorithm to select the optimal attributes to reduce data distortion and increase the watermarking capacity, which improves the scheme's robustness. In 2015, Iftikhar et al. used genetic algorithms and data analysis methods from information theory to deal with the watermarking problem (RRW) [16]. They used genetic algorithms to generate optimal watermarks to reduce data distortion. However, developing the optimal watermark requires a large amount of computation time. 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 cutting 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 enhance the usability of the data [21]. In 2022, Xiang et al. proposed a robust watermarking algorithm based on order-preserving encryption scheme(OPES) and circular histograms [22].

The above schemes for relational data show that researchers are trying to improve the robustness and capacity of watermarking while reducing the impact of watermarking on the data. It seems that the robustness of watermarking and data availability cannot be achieved at the same time. To solve this problem, we propose a random reversible watermarking scheme in this work.

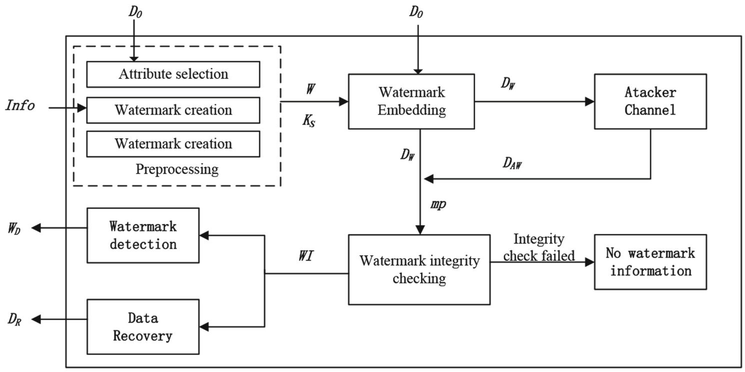

Fig. 1. System architecture.

3 Scheme

The main structure of the scheme is shown in Fig. 1. The scheme mainly includes the following five main phases: (1) preprocessing; (2) watermark embedding; (3) watermark integrity detection; (4) watermark extraction; (5) data recovery.

The watermark preprocessing is mainly to do some preparatory work before embedding watermark. The first step is encoding the data owner's copyright information into a watermark sequence that can be embedded in the database. The second step is filtering out the attribute columns suitable for embedding the watermark. The final step is to generate a random function to determine the proportion of each tuple in which the watermark will be embedded. The main task of the watermark embedding phase is to embed the watermark information into the data according to the key, and finally obtain the database with the watermark and return the auxiliary data. The watermark integrity check phase checks each partition's watermark integrity. The watermark extraction phase extracts the watermark information from the stolen watermarked data and implements the issue of proof of copyright. The data recovery phase removes the watermark from the data and restores the original data. Notation used in the scheme is given in Table 1.

Definition 1. Watermark Integrity, WI(Watermark Integrity), indicates the watermark's degree of integrity for each partition and. In some ways, it reflects the extent to which data has been corrupted. $WI = [f_1, f_2, f_3, ..., f_N](f_i \in 0, 1)$. Where f_i is the identifier of the complete watermark of partition i.

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. It ensures that in case of severe attacks in the database, the data owner can still extract the complete watermark from the data.

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 least 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
$bit_W[k]$	The kth watermarked bit of the LSB	$W_D[l]$	The lth bit of the detected watermark
Max	Proportional limit for embedding watermark attributes	LSB	The least significant bit of the data
mp	Auxiliary data	MSB	The most significant bit of the data
$mp[i]$	Auxiliary data for partition i	mpw	Auxiliary data for the data to be tested
$r.key$	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
bit_o	The original bits that be recovered	$bitwsl$	The value of the $bitsl$ in the checked data
Mean	Mean value of the attribute	Std	Standard deviation of attributes

3.1 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; (3) determining a random function.

Attribute Selection. Not all attributes are suitable for embedding attributes, and for some attributes, small changes can significantly impact data quality. We select multiple attribute columns from the database as candidate attributes that can be used as identifiable features, and embedding a watermark in an attribute fewer impacts data quality. The candidate attributes are then reordered and numbered in ascending order, and the reordering enhances robustness against attribute attacks.

Watermark Creation. The 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. Then the binary sequence is encoded with the data as watermark information. We embed the full watermark into the different partitions. We simply select the partition with the most complete watermark for the copyright claim. The watermark creation equation is shown in Eq. (1):

$$W = Info \text{ xor Rand} \tag{1}$$

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

Determine Random Function. We embed an unequal number of watermarks in each tuple. The F function chooses the number of watermarks embedded in each tuple. 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 selected by Eq.(2):

$$F(x, y, z) = H(x|H(x|y))\%z \tag{2}$$

3.2 Watermark Embedding

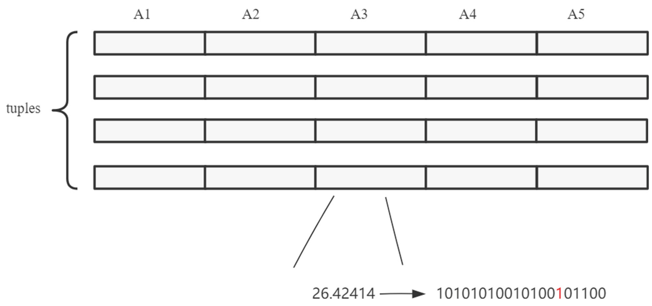

Fig. 2. Least significant bits

The main task of the watermark embedding phase is to embed the watermark into the database in an invisible way. 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 Fig. 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 Eq. (3).

$$bitsl = l + \sigma * bits \tag{3}$$

l is the subscript position of the embedded watermark W. *sigma* is any random number bigger than the watermark length ξ. When extracting the stored information, if the value of *bitsl* is bigger than ξ or not, the value of bits is 1 or 0. When the value of *bitsl* is less than ξ, the value of l is *bitsl*. Otherwise, l is calculated by Eq. (4). The specific watermarking process is shown in Algorithm 1.

$$l = bits - \sigma * bits \tag{4}$$

Algorithm 1. Watermark embedding algorithm

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

```
for each tuple r ∈ D do
    i = H(r.key, K_P, MSB) mod λ
    if H(r.key, K_S[i]) mod η =0 then
        for each A_j in r do
            ω=F(r.key, K_S[i], Max)
            if H(r.key, K_S[i], MSB) mod ⌈ω⌉ = 0 then
                W_index l = H(r.key, K_S[i]) mod ξ
                bit_index k = H(r.key, K_S[i]) mod υ
                bits = bit[k] xor W[l]
                bitsl = l + σ * bits
                Stroe(mp[i], bitsl)
                Update(bit[k], bits)
            end if
        end for
    end if
end for
return D_W, mp
```

Here we take an 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. 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 into a watermark bit or not. Lines 5 and 6 generate a random number and attributes in the tuple with a ratio of $1/\omega$ will be watermarked. The value of ω is 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. We can adjust the watermark capacity and control the data distortion by adjusting Max. 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 using an exception or operation to generate a new bit. Line 10 connects the embedded watermark bits with the subscript of the watermark where it is located to create the auxiliary data $bitsl$ with the embedded watermark and watermark position information via Eq. (3). Then, $store(mp[i], bitsl)$ means to store the $bitsl$ into the auxiliary array element $mp[i]$, $mp[i]$ represents the auxiliary data for partition i. $Update(bit[k], bits)$ in line 12 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.

3.3 Watermark Integrity Detection

To ensure that the data owner can detect its watermark, 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 partition's watermark information is considered 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 the 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.

Algorithm 2. Watermark integrity checking algorithm

Input: D_W, K_P, K_S, mp
Output: WI
1: **for** each tuple $r \in D_W$ **do**
2: $i = H(r.key, K_P)$ mod λ
3: **if** $H(r.key, K_S[i])$ mod η =0 **then**
4: **for** each A_j in r **do**
5: $\omega = \mathrm{F}(r.key, K_S[i], Max)$
6: **if** $\mathrm{H}(r.key, K_S[i], MSB)$ mod $\lceil \omega \rceil = 0$ **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: $bitwsl = l + \sigma * bit[k]$
10: **if** $bitwsl$ in $mp[i]$ **then**
11: $Store(mpw[i], bitwsl)$
12: **end if**
13: **if** $mpw[i].length\%\xi = 0$ **then**
14: $WI[i] = Check(mpw[i], mp[i])$
15: **end if**
16: **end if**
17: **end for**
18: **end if**
19: **end for**
20: **return** WI

Lines 1 to 8 of Algorithm 2 are similar to the watermark 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 Eq. 3. 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 marker 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 copyright information.

3.4 Watermark Extraction

The data owner extracts the watermarked information from the data for copyright claim. This requires the watermark to be robust. Even in the case of a severe attack on the data, the data owner can still extract the complete watermark information from the data. We use majority voting in the watermark extraction process to reduce the attack's impact and improve the accuracy of watermark detection. The watermark extraction algorithm is shown in Algorithm 3.

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: $i = H(r.key, K_P) \bmod \lambda$
5: **if** i == f_i **then**
6: **if** $H(r.key, K_S[i]) \bmod \eta = 0$ **then**
7: **for** each A_j in r **do**
8: $\omega = \mathrm{F}(r.key, K_S[i], Max)$
9: **if** $\mathrm{H}(r.key, K_S[i], MSB) \bmod \omega = 0$ **then**
10: $W_{index}\ l = H(r.key, K_S[i]) \bmod \xi$
11: $bit_{index}\ k = H(r.key, K_S[i]) \bmod \upsilon$
12: $Get(mp[i], bitsl)$
13: $bits = (bitsl - l)/\sigma$
14: $bitws = bits \text{ xor } W[l]$
15: $W_D[l] = bit_W[k] \text{ xor } bitws$
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 watermark extracted 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 7 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 a key to get $Info$. Info is the copyright information of the owner of the decrypted data.

3.5 Data Recovery

When the quality of the data is not sufficient to meet the demand, the data will be useless. The data owner licenses the key and supporting files to the data user, who then uses the key and data recovery algorithm to recover the original data. 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:                 **if** $bits\ in\ mp[i]$ **then**
12:                     $bit_o = bits$ Xor $W[l]$
13:                     $Update(bit_w[k], bit_o)$
14:                 **end if**
15:             **end if**
16:         **end for**
17:     **end if**
18: **end for**
19: **return** D_R

Lines 1 to 8 are same as the watermark embedding algorithm. Line 9 gets the value of the stored *bitsl* from the auxiliary array. Lines 10 compute the values of *bits*. If the *bits* is in the auxiliary data $mp[i]$, line 12 is an alias to the *bits* to get the original bit_o. Line 13 uses $Update(bit_W[s], bit_o)$ to update the value of $bit_W[k]$ to bit_o, removing the watermark information. Finally, the recovered data is returned to D_R.

4 Experimental Analysis

This section evaluates various aspects of the scheme. The experiments aim to verify the accuracy and robustness of the scheme. The experiments include

(1) Statistical distortion experiments, (2) watermark capacity experiments, (3) robustness experiments. The experimental environment is: a 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 tuples 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 $Max = 5$.

4.1 Statistical Distortion Experiments

Watermarking inevitably causes distortion of the data, and the longer the watermark, the more severe the distortion. Therefore, we compared some statistical metrics of the original and watermarked databases. The results of the comparison were compared with other schemes, such as DEW, GADEW, PEEW and GAHSW.

To visualize the effect of watermarking on data distortion, we calculate the mean, standard deviation and mean absolute error(MAE) before and after watermark embedding data. And the value of MAE is calculated from Eq. (5).

$$MAE = \frac{\sum_{i=1}^{n} |A_i - A_i^w|}{n} \tag{5}$$

where A_i is the original attribute, A_i^w is its watermarked version, and n is the total number of attributes for all tuples in the database.

To make the experimental results more precise, the results were averaged from 10 experiments. Table 2 shows the experimental results for each attribute of the database, and the results are compared with other schemes in the table. To visualize the changes in mean and variance, the difference in mean (DM) and difference in standard deviation (DS) for each attribute are given in Table 3. DM and DS are calculated by Eq. (6) and Eq. (7):

$$DM = |Mean_D - Mean_{DW}| \tag{6}$$

$$DS \ = |Std_D - Std_{DW}| \tag{7}$$

where $Mean_D$ and Std_D represent the original database's mean and standard deviation values, $Mean_{DW}$ and Std_{DW} represent the mean and standard deviation of the watermarked database. In addition, the MAE values for the different scenarios are shown in the last row of Table.

As shown in Table 3, the variation in our scheme's mean and standard deviation between the original and watermarked databases is minimal. The maximum

Table 2. Results of statistical distortion of the database.

Attributename	Original database Mean	Original database Std	DEW Mean	DEW Std	GADEW Mean	GADEW Std	PEEW Mean	PEEW Std
Elevation	2862.037	231.375	2862.037	231.375	2862.018	231.442	2862.037	231.375
Aspec	138.163	103.779	138.403	103.946	137.712	103.635	138.829	105.149
Slope	11.803	6.534	11.819	7.074	11.788	6.542	11.818	6.546
H_D_To_Hydrology	260.580	202.774	260.580	202.774	260.292	202.366	261.844	203.507
V_D_To_Hydrology	35.242	42.627	48.462	66.603	25.593	57.792	41.540	48.110
H_D_To_Roadways	3344.252	1776.873	3346.659	1781.056	3342.698	1779.658	3352.709	1780.346
Hillshade_9am	218.234	20.933	218.321	21.071	217.984	21.145	218.552	21.242
Hillshade_Noon	225.451	16.653	225.522	16.966	225.359	17.070	225.669	17.869
Hillshade_3pm	139.301	31.184	139.387	31.228	139.255	31.440	139.425	31.985
H_D_To_Fire_Points	3589.618	1781.439	3591.923	1786.310	3588.353	1781.782	3700.371	2071.255

Attributename	GAHSW Mean	GAHSW Std	OUR Mean	OUR Std
Elevation	2862.040	231.462	2862.071	231.375
Aspec	138.206	103.847	138.167	103.779
Slope	11.835	6.571	11.834	6.498
H_D_To_Hydrology	260.659	202.739	260.588	202.777
V_D_To_Hydrology	35.242	42.627	35.277	42.623
H_D_To_Roadways	3344.255	1776.972	3344.256	1776.875
Hillshade_9am	218.243	20.984	218.270	20.922
Hillshade_Noon	225.528	16.625	225.454	16.652
Hillshade_3pm	139.360	31.219	139.333	31.175
H_D_To_Fire_Points	3589.660	1781.533	3589.622	1781.438

variation caused is 0.035, which is negligible compared to DEW, GADEW, and PEEW. Although the variation in mean and standard deviation for the GAHSW scenario is also tiny, it is still worse than the results in our scheme. It is important to note that DEW and PEEW change 0 for some attributes, such as the first attribute shown in Table 3. This is because the watermark is not embedded in these attribute columns. In general, the smaller the variation in the mean and standard deviation, the smaller the impact of the watermarking scheme on the data quality. The same is also applicable to the MAE for quantifying attribute distortion. The MAE values for DEW, GADEW, and PEEW are not ideal, with their values being 28.395, 7.867, and 133.125, respectively. This means that these schemes introduce relatively large distortions into the data, which severely affect its usability of the data. The MAE value of GAHSW is the lowest, at 0.775. Although this value is much lower than the other schemes, it is still higher than the value in our proposed scheme. Although the improvement is minor, it is still an improvement. Thus, the experiments show that our proposed scheme is highly effective and significantly outperforms the other schemes in terms of statistical distortion performance.

Table 3. Results of difference in Mean, difference in Std and MAE

	DEW		GADEW		PEEW		GAHSW		OUR	
Attributename	DM	DS	DM	DS	DM	DS	DM	DS	DM	DS
Elevation	0	0	0.019	0.067	0	0	0.003	0.087	0.035	0
Aspec	0.240	0.167	0.451	0.144	0.666	1.37	0.043	0.068	0.004	0
Slope	0.016	0.540	0.015	0.008	0.015	0.012	0.032	0.037	0.031	0.036
H_D_To_Hydrology	0	0	0.288	0.408	1.264	0.733	0.079	0.035	0.008	0.003
V_D_To_Hydrology	13.220	23.976	9.649	15.165	6.298	5.483	0.872	0.503	0.035	0.004
H_D_To_Roadways	2.407	4.183	1.554	2.785	8.457	3.473	0.003	0.099	0.004	0.002
Hillshade_9am	0.071	0.313	0.092	0.417	0.218	1.216	0.077	0.028	0.036	0.011
Hillshade_Noon	0.086	0.044	0.046	0.256	0.124	0.801	0.059	0.035	0.004	0.001
Hillshade_3pm	0.087	0.138	0.250	0.212	0.318	0.309	0.009	0.051	0.032	0.009
H_D_To_Fire_Points	2.305	4.871	1.265	0.343	110.753	289.816	0.042	0.094	0.003	0.001
MAE	28.395		7.867		133.125		0.775		0.624	

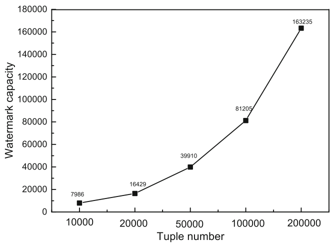

Fig. 3. Watermark capacity under different tuples

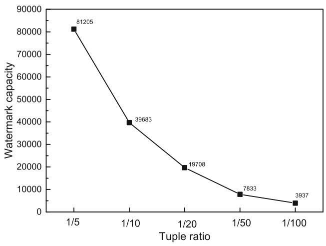

Fig. 4. Watermark capacity in different proportions

4.2 Watermark Capacity Experiment

Watermark capacity is a measure of the ability of a watermark to resist attacks. The more bits of information a watermark has in a certain amount of data, the more resistant the watermark will be against malicious attacks. However, the larger the watermark capacity, the greater the distortion of the data.

Figure 3 shows the change in watermark capacity under different number of tuples at $1/\eta = 1/5$. Figure 4 shows the difference 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, the value is randomly 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). The experimental results are consistent with the inference results within the error tolerance.

4.3 Robustness Experiments

In this section, different attacks are performed on the watermarked data. These attacks include (1) tuple addition attack, (2) tuple alteration attack, and

(3) tuple deletion attack. We also compare our scheme with recent reversible database watermarking schemes, such as DEW, GADEW, PEEW, and RRW. Since the watermark detection rate criteria of GAHSW schemes are not the same as the previous ones, this scheme can't be compared.

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

Watermark detection: As shown in Fig. 5, when the same amount increases the number of tuples as the original data, the detection accuracy of our scheme 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 Fig. 6, 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.

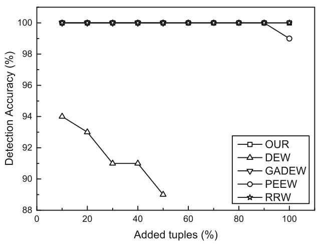

Fig. 5. Watermark detection accuracy after tuples addition attack

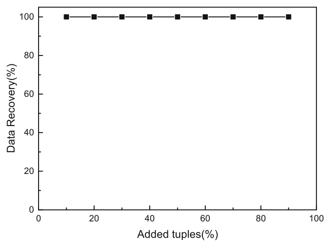

Fig. 6. Data recovery accuracy after tuples addition attack

Tuple Alteration Attack. 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 Fig. 7, when 90% of the tuples are alternated, our scheme still maintains 100% accuracy in watermark detection, while only RRW maintains 100% accuracy in the other schemes.

Data recovery: As shown in Fig. 8, it is almost impossible to perform 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 recover data that has not been attacked fully.

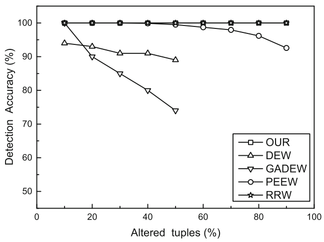

Fig. 7. Watermark detection accuracy after tuples alteration attack

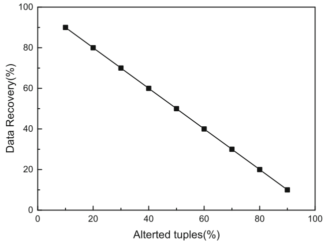

Fig. 8. Data recovery accuracy after tuples alteration attack

Tuple Deletion Attack. In this 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 Fig. 9, when 90% of the tuples are deleted, our scheme can still maintain 100% watermark detection accuracy. When many tuples are deleted, the detection accuracy of GADEW, PEEW, GAHSW, and other methods, except RRW, decreases significantly.

Data recovery: As seen in Fig. 10, the original 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.

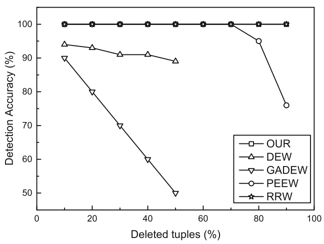

Fig. 9. Watermark detection accuracy after tuples deletion attack

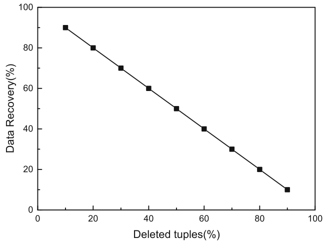

Fig. 10. Data recovery accuracy after tuples deletion attack

5 Conclusion

Aiming at the problem that the watermark information extracted under severe attack is incomplete, a random 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. The randomness of watermark information distribution is increased by controlling different tuples to embed an unequal proportion of watermarks. When extracting the watermark, the accuracy of watermark detection is improved by discarding the attacked bits. Finally, the scheme's statistical distortion, watermark capacity, and robustness experiments are analyzed. We can draw some conclusions. Firstly, our scheme can resist severe tuple attacks. Secondly, the impact of embedded watermarks on data availability is small. Compared with DEW, GADEW, RRW, PEEW, and GAHSW, our scheme has better attack resistance and causes less data distortion, which 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. Int. J. Autom. Comput. **8**(3), 280–285 (2011)
2. Cheng, L., Liu, F., Yao, D.: Enterprise data breach: causes, challenges, prevention, and future directions. Wiley Interdisc. Rev. Data Mining Knowl. Discov. **7**(5), 1211 (2017)
3. Wong, P.W., Memon, N.: Secret and public key image watermarking schemes for image authentication and ownership verification. IEEE Trans. Image Process. **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 Process. **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. Sustain. Comput.: Inform. Syst. **15**, 82–87 (2017)
6. Brassil, J.T., Low, S., Maxemchuk, N.F.: Copyright protection for the electronic distribution of text documents. Proc. 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. Elsevier (2002)
8. Sion, R., Atallah, M., Prabhakar, S.: Rights protection for categorical data. IEEE Trans. Knowl. Data Eng. **17**(7), 912–926 (2005)
9. Niu, X., Zhao, L., Huang, W., Zhang, H.: Watermarking relational databases for ownership protection. Acta Electron. Sin. **31**(S1), 2050 (2003)
10. Shehab, M., Bertino, E., Ghafoor, A.: Watermarking relational databases using optimization-based techniques. IEEE Trans. Knowl. Data Eng. **20**(1), 116–129 (2008)
11. Zhang, Y., Yang, B., Niu, X.-M.: Reversible watermarking for relational database authentication. J. Comput. **17**(2), 59–66 (2006)
12. Zhang, Y., Niu, X.-M.: Reversible watermark technique for relational databases. Acta Electron. Sin. **34**(S1), 2425 (2006)
13. Gupta, G., Pieprzyk, J.: Reversible and blind database watermarking using difference expansion. Int. J. Digital Crime 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. J. Chin. Inst. Eng. **36**(1), 87–97 (2013)
15. Jawad, K., Khan, A.: Genetic algorithm and difference expansion based reversible watermarking for relational databases. J. Syst. Softw. **86**(11), 2742–2753 (2013)

16. Iftikhar, S., Kamran, M., Anwar, Z.: Rrw-a robust and reversible watermarking technique for relational data. IEEE Trans. Knowl. Data Eng. **27**(4), 1132–1145 (2014)
17. Franco-Contreras, J., Coatrieux, G.: Robust watermarking of relational databases with ontology-guided distortion control. IEEE Trans. Inf. Forensics Secur. **10**(9), 1939–1952 (2015)
18. Imamoglu, M.B., Ulutas, M., Ulutas, G.: A new reversible database watermarking approach with firefly optimization algorithm. Math. Problems in Eng. **2017**(2), 1–14 (2017)
19. Hu, D., Zhao, D., Zheng, S.: A new robust approach for reversible database watermarking with distortion control. IEEE Trans. Knowl. Data Eng. **31**(6), 1024–1037 (2018)
20. Li, Y., Wang, J., Ge, S., Luo, X., Wang, B.: A reversible database watermarking method with low distortion. Math. Biosci. Eng. **16**(5), 4053–4068 (2019)
21. Li, Y., Wang, J., Luo, X.: A reversible database watermarking method non-redundancy shifting-based histogram gaps. Int. J. Distrib. Sens. Netw. **16**(5), 1550147720921769 (2020)
22. Xiang, S., Ruan, G., Li, H., He, J.: Robust watermarking of databases in order-preserving encrypted domain. Front. Comp. Sci. **16**(2), 1–9 (2022). https://doi.org/10.1007/s11704-020-0112-z