2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) | 978-1-6654-9425-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/TrustCom56396.2022.00156

# An Attribute-attack-proof Watermarking Technique for Relational Database

Shuguang Yuan * †, Chi Chen * †, Ke Yang * †, Tengfei Yang ‡ and Jing Yu * †
* State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing 101400, China
yuanshuguang, chenchi, yangke, yujing@iie.ac.cn
‡ National Computer Network Emergency Response Technical Team/Coordination Center of China
yang.tf@foxmail.com

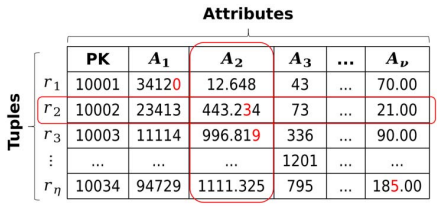
***Abstract*—Proving ownership rights on relational databases is an important issue. The robust watermarking technique could claim ownership by insertion information about the data owner. Hence, it is vital to improving the robustness of watermarking technique in that intruders could launch types of attacks to corrupt the inserted watermark. Furthermore, attributes are explicit and operable objectives to destroy the watermark. To my knowledge, there does not exist a comprehensive solution to resist attribute attack. In this paper, we propose a robust watermarking technique that is robust against subset and attribute attacks. The novelties lie in several points: applying the classifier to reorder watermarked attributes, designing a secret sharing mechanism to duplicate watermark independently on each attribute, and proposing twice majority voting to correct errors caused by attacks for improving the accuracy of watermark detection. In addition, our technique has features of blind, key-based, incrementally updatable, and low false hit rate. Experiments show that our algorithm is robust against subset and attribute attacks compared with AHK, DEW, and KSF algorithms. Moreover, it is efficient with running time in both insertion and detection phases.**

***Index Terms*—Watermarking, Relational database, Robustness, Attribute Attack**

## I. Introduction

With the arrival of the data era, business interests are mined and analyzed. While attackers also covet the values of data. Hence, copyright protection is vital for data owners or providers. For this purpose, watermarking techniques are widely used in many fields like images [1], multimedia [2], text [3], databases [4] and applications [5].

A database consists of a collection of relations. A table represents each one of relations, where columns are attributes (i.e., $A_i$) and rows are tuples (i.e., $r_j$). The figure 1 shows the table representation of a database relation. Then, watermarking techniques insert the watermarks by introducing "small" changes. The values being marked red represent watermarks. Without secret parameters, attackers cannot detect certain positions of watermarks. At the same time, watermarking techniques should be able to resist benign updates and malicious attacks. According to our survey, most existing techniques like [4] [6] [7] [8] aim to improve robustness against subset attack such that tuples are inserted/altered/deleted, e.g., the $r_2$ is deleted. In comparison, the malicious actions on attributes remain a challenge to be solved. In most techniques like [4] [9] [6] [7] [8], the issues of attribute attack that are not raised. The technique [10] just did an experiment of deleting an attribute and adding another attribute at the same location. The experiments are inadequate. Techniques [11] [10] inserting the watermarks into a single fixed attribute may survive an attribute attack. Nevertheless, that just relies on the existence of watermarked attribute. It is an unconscious result that cannot provide a determining evaluation of robustness against attribute attack. According to our experiments V, to a certain extent, some techniques [4] [12] could survive under low-intensity attribute attack. In brief, these techniques above do not raise a solution for attribute attack.




| | PK | $A_1$ | $A_2$ | $A_3$ | ... | $A_\nu$ |
|---|---|---|---|---|---|---|
| $r_1$ | 10001 | 34120 | 12.648 | 43 | ... | 70.00 |
| $r_2$ | 10002 | 23413 | 443.234 | 73 | ... | 21.00 |
| $r_3$ | 10003 | 11114 | 996.819 | 336 | ... | 90.00 |
| ⋮ | ... | ... | ... | 1201 | ... | ... |
| $r_\eta$ | 10034 | 94729 | 1111.325 | 795 | ... | 185.00 |

Fig. 1: Table representation of a database relation

The attributes or tuples are vital to conveying data values. For an attacker, it is the same to distort attributes or tuples. On the one hand, attributes are more explicit and operable objectives. The attacker may destroy attributes to distort watermarks, e.g., $A_2$ is deleted. On the other hand, the data owner may publish a part of data with some of the attributes (e.g., selling of database fragments), making it possible to be the watermarks undetectable by just changing a part of the attributes. Hence, for purposes of both benign updates and malicious attacks, it is valuable to apply a watermarking technique to solve these issues. In this paper, we aim to propose a novel attribute-attack-proof watermarking technique. Our contributions are as follows:

1. We use classification-based and secret sharing-based mechanisms to provide the ability to re-order attributes and correct watermark, respectively. Specifically, the first mechanism prevents detecting false attributes to extract error bits

*Jing Yu is corresponding author.

2324-9013/22/$31.00 ©2022 IEEE
DOI 10.1109/TrustCom56396.2022.00156

string; the second mechanism addresses the problem of lacking attributes by assigning shares of watermarks to watermark attributes.

2. We propose an efficient algorithm AAP. In order to improve the detection accuracy, it uses twice majority voting to correct extracted watermarks. At the same time, it has the advantage features of blind, key-based, incrementally updatable, and low false hit rate and is robust against subset and attribute attacks.

3. We evaluate our AAP algorithm compared with AHK [4], KSF [12] , and DEW [13] algorithms in a real-world dataset. The result of subset, attribute, and multifaceted attribute attacks prove AAP algorithm achieves our goals.

The paper is organized as follows: In Section II, we describe the related work about the robustness of existing watermarking techniques. In Section III, the problem definition of attribute attack is proposed. The overview of the scheme is described. In Section IV, the implementation of preprocessing, insertion, and detection algorithms are demonstrated. In Section V, data-driven experiments are showed. The robustness performances against subset, attribute, and multifaceted attacks are presented. In section VI, we conclude our work.

## II. Related Work

The first relational database watermarking technique was proposed by Agrawal *et al.* [4] [14]. By Hash function and secret key, it selects a fraction of tuples, attributes, and bit locations for insertion. It is known as the AHK Algorithm by using the combination of the secret key (SK) and the primary key (PK) of tuples to decide whether to mark them or not. A number of following techniques like [15] [16] [17] continue to study this technique. Cui *et al.* [15] proposed a weighted watermarking algorithm that assigns different weights to attributes. Guo *et al.* [16] proposed a twice-insertion scheme for identifying both the owner and the traitor.

In [18], Sion *et al.* proposed a method that encodes the watermark bit relies on altering the size of the "positive violators" set. That addresses data re-sorting, subset selection, and linear data changes attacks but subset deletion and alteration attack. Shehab *et al.* [19] formulated the watermarking techniques of relational database as a constrained optimization problem. They insert watermarks with constraints on partitioned tuple groups. They presented two techniques to solve the formulated optimization problem based on genetic algorithms and pattern-searching techniques. It is robust against subset attacks. In [8], Saman *et al.* developed an information technique that inserts watermarks on non-significant effect features. The attribute selection relies on mutual information to control data distortions. In [7], Javier *et al.* raised a scheme that modulates the relative angular position of the circular histogram center of mass of one numerical attribute for message insertion. The techniques above have good robustness against subset attack. However, the problem of attribute attack did not address.

Besides, most watermarking techniques use the primary key to deciding where and how to insert the watermark. The primary key is also an attribute, which stores unique values that identify each tuple in the relation. It enforces synchronization between watermark insertion and detection. The virtual primary key schemes [20] [21] aim to avoid compromising watermark detection due to the deletion or replacement of the relation's primary key. In this paper, we do not consider the problem that the primary key attribute is attacked by erasing or updating.

## III. Approach Overview

In this section, we illustrate the main components of our watermarking technique. Consider a database relation $D$, let the schema of $D$ be $D(P, A_1, ..., A_\nu)$, where $P$ is primary key and $A_1$, $A_2$, ..., $A_\nu$ are numerical attributes. The number of the attribute is $\nu$. Besides, let $r$ be a tuple in $D$, and the number of tuples in $D$ be $\eta$. A robust watermark algorithm is used to insert the watermark bits string ($W$) into the dataset $D$. Figure 2 shows the block diagram summarizing the main components of our watermarking technique.

Preprocessing aims to remember the original order of watermarked attributes and duplicate copies of a part of watermark bits string mapping, then assign then to each watermarked attribute. It has two main steps: **Attribute Classification.** In this step, optional attributes in original data are selected for watermarking. The data in each selected attribute are considered samples. Moreover, these samples are trained to identify selected attributes by a classifier. **Watermark Shares Generation.** By applying the idea of replicated secret-sharing mechanism, watermark shares $WS_1, ..., WS_n$ in bits string are generated from watermark $W$. Every $WS_i$ maps an attribute. Each one contains a part of the whole watermark bits string, being capable of correcting others' errors. Hence, if some attributes are deleted, watermark shares mapped on other attributes could fill the lacks and correct the errors.

The encoding process can be summarized in the following steps: **Watermark Insertion.** The watermark insertion algorithm takes a secret key $\mathcal{K}$, a cryptographic pseudorandom sequence generator [22] $\mathcal{S}$ and the watermark shares ($W_i$) as input and converts a dataset $D$ into watermarked dataset $D_w$. Without the knowledge of the seed ($\mathcal{K}$), it is computationally infeasible to compute the sequence because different $\mathcal{K}$ leads to different sequences using cryptographic pseudorandom sequence generator $\mathcal{S}$. The changes in the data are controlled by placing certain bounds on the least significant bit (LSB) $\xi$ of selected attributes. According to the above parameters, it uses the bit setting mechanism to alter the bit of selected tuples. Note that, for distinct attribute, it inserts bits from a particular watermark share. **Attacker Channel (Attack Model).** When the owner of data marks database $D$ to generate a watermarked database $D_w$, the attackers want to destroy the watermark by launching different types of attacks, including subset attacks and attribute attacks. Malicious attacks may take various forms: 1. Subset Deletion Attack. Attacker can delete a subset of watermarked tuples from database $D_w$. Deletion will damage tuples which may have watermarks. 2. Subset Alteration Attack. Attacker can change the value of tuples on database $D_w$. We assume the

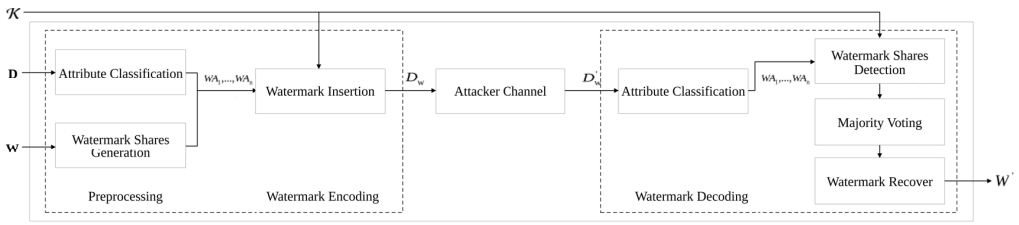



Fig. 2: Proposed Scheme

TABLE I: Notions

| | |
|---|---|
| $D$: Original dataset | $D_w$: Watermarked dataset |
| $A$: Attributes | $WA$: Watermark Attribute |
| $\eta$: Number of tuples | $\nu$: Number of attributes |
| $t$: Number of shares | $n$: Number of watermark attributes |
| $\xi$: Least significant bit | $1/\gamma$: Fraction of tuples used |
| $\mathcal{K}$: Secret key | $\mathcal{S}$: Pseudo-random squence generator |
| $\tau$: Detection threshold | $W$: Watermarks |
| $P$: Primary key attribute | $L$: Length of watermark |
| $l$: Length of watermark share | |

attacker does not know the real positions where the watermarks were inserted. 3. Subset Insertion Attack. Attacker adds similar tuples that may disturb detection. 4. Attribute Attack. Attacker may either add spurious attributes, delete existing ones or re-order the attributes randomly.

The watermark decoding process can be summarized in the following steps: **Attribute Classification.** Before detecting watermarks, the trained classifier aims to identify watermarked attributes $WA$, then recover the original order of watermark attribute $WA$. **Watermark Shares Detection.** The detection is the reverse process of insertion. The watermarked dataset $D_w$, secret key $\mathcal{K}$, pseudorandom sequence generator $\mathcal{S}$ are used to extract bits. **Majority Voting and Watermark Recover.** This step contains twice majority voting. The first majority voting corrects the extracted bits string because each bit will be inserted many times. Even being attacked, the particular bit could be corrected by comparing this in other positions. Another majority voting corrects watermark shares which helps reconstruct watermark $W^{'}$. On the receiver side, the original watermark bits string ($W$) is reconstructed by a certain number of watermark shares extracted on watermark attributes. As a result, high detection accuracy is achieved irrespective of the types of attacks in the watermarked data.

## IV. Algorithm

In this section, different phases of our algorithm are demonstrated. Table I gives the notations used in our technique.

### A. Preprocessing

The preprocessing phase consists of two main steps, Attribute Classification and Watermark Shares Generation. The attribute attack may cause the out-of-order problem or lacking problem of attributes. The solution of out-of-order attributes helps address the lack problem. Because when the lacked or malposed attributes are identified, the detection strategy can be adapted easily. These attributes could be skipped or corrected accurately. The algorithm of preprocessing is shown in Algorithm 1. The inputs are original dataset $D$, watermark attributes $WA$, watermark $W$ and parameters $n$, $t$. The watermark is an L-bits long binary bit string $b_0 b_1 b_2 ... b_{L-1}$.

*1) Attribute Classification:* We say that the candidates' attributes to be watermarked are defined as Watermark Attribute ($WA$). The $n$ denotes the number of $WA$.

**Definition 1 (Watermark Attribute).** *Given an original dataset D, let $WA$ be the optional numerical attribute to be watermarked, which $WA \subseteq A$.*

The original order of $WA$ is the information we should gain accurately before and after inserting watermarks, preventing detecting error watermarks on false watermark attributes. In order to address the problem of out-of-order attributes, machine learning is a suitable tool that turns data in different attributes into the information of attributes order. That is a classic multi-class classification task. Classification methods including Decision tree [23], Neural network [24], Linear/logistic/Softmax regression [25] can be applied. In this paper, because the novelty of classification is not our contribution, we use a simple open-source Linear classifier [1] to implement attribute classification. Data in each watermark attribute $WA_i$ are trained with label $i$ by a classifier. Once each $WA$ is classified, a classifier could compare the candidate dataset and return the label of attributes. Thus, in detection phase, the identified labels of attributes are referenced to re-sort the attribute order. Lines 1-3 show this process in algorithm 1.

*2) Watermark Shares Generation:* The second step is to generate different watermark shares for the watermark attribute. The idea of the secret-sharing technique is used, which divides a watermark $W$ into n $WS_1, ..., WS_n$ watermark shares such that no one can know the watermark $W$ unless they can collect $\geq$ $t$ shares, where $t$ is the threshold, $n$ is the number of attributes. Each $WS_i$ contains a part of watermark bits string $W$. The repetitive part of bits string between any two $WS$ can be considered the redundancy of $W$. When some attributes are deleted, watermark shares extracted from the remaining attribute can still reconstruct the $W$ by comparing, correcting repetitive parts, and supplementing the non-repetitive part of $W$. That addresses the problem of suffering from attribute deletion.

[1] https://github.com/oracle/tribuo

The challenges of generating watermark shares consist of three parts: 1) the watermark share should have the same representation as to the watermark, such that it be in bit string form as well; 2) the watermark should be distributed equally to each share. It ensures that each attribute could hold a certain amount of bits, which prevents uneven distribution from causing recovery difficulties of watermark; 3) the shares are capable of error tolerance to reconstruct watermark. The tiny changes should not lead to the failure of reconstructing the watermark $W$, then decreasing the detection accuracy. Hence, lots of classic secret sharing techniques [26] [27] can not be used in this situation. There are a example, such that Shamir's secret-sharing uses univariate polynomial $f(y) = s + r_1y + r_2y^2 + ... + r_ty^t$ to reconstruct secret $s$. Each share $s_j$ is $f(j)$. By performing Lagrange interpolation, set of $s_j$ ( the number of shares $\geq$ t) can reconstruct secret $s$. However, it is obvious that if an error $s_j^{'}$ joins reconstruction, the result $s$ is wrong because polynomial $f(y)$ returns a wrong value.

Thus, the idea of replicated secret sharing [28] [29] is used for watermark shares generation. Because when shares have some error of bits, the reconstruction of replicated secret sharing is still achieved. Note that, to a certain degree, the error could be corrected because each bit is inserted in different tuples many times.

*combination of* $C_5^2$

| index/ pieces | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $B_0$ | 1 | 1 | 0 | 0 | 0 |
| $B_1$ | 1 | 0 | 1 | 0 | 0 |
| $B_2$ | 1 | 0 | 0 | 1 | 0 |
| $B_3$ | 1 | 0 | 0 | 0 | 1 |
| $B_4$ | 0 | 1 | 1 | 0 | 0 |
| $B_5$ | 0 | 1 | 0 | 1 | 0 |
| $B_6$ | 0 | 1 | 0 | 0 | 1 |
| $B_7$ | 0 | 0 | 1 | 1 | 0 |
| $B_8$ | 0 | 0 | 1 | 0 | 1 |
| $B_9$ | 0 | 0 | 0 | 1 | 1 |

| $WS_1$ | $WS_2$ | $WS_3$ | $WS_4$ | $WS_5$ |
|---|---|---|---|---|
| | | $B_0$ | $B_0$ | $B_0$ |
| | $B_1$ | | $B_1$ | $B_1$ |
| | $B_2$ | $B_2$ | | $B_2$ |
| | $B_3$ | $B_3$ | $B_3$ | |
| $B_4$ | | | $B_4$ | $B_4$ |
| $B_5$ | | $B_5$ | | $B_5$ |
| $B_6$ | | $B_6$ | $B_6$ | |
| $B_7$ | $B_7$ | | | $B_7$ |
| $B_8$ | $B_8$ | | $B_8$ | |
| $B_9$ | $B_9$ | $B_9$ | | |

Fig. 3: Assignment of pieces on (3,5) secret sharing

The generation of watermark share is described as follows. We say the $n$ is the number of watermark attributes $WA$, $t$ is the threshold, $C_n^t$ is the abbreviation of t-combination of the set of n combination. Hence, the number of recoverable attributes combination is $|C_n^t| + ... + |C_n^{n-1}| + |C_n^n|$, i.e., when the number of attributes is greater than or equal to $t$. The $C_n^{t-1}$ is the set of maximal non-recoverable attributes combination. Then, the watermark $W$ is additively split into $|C_n^{t-1}|$(i.e., $m$) pieces $B_0, ..., B_m$. Note that each $B$ contains a $L/m$-bits long binary string. Each $B$ denotes a possibility within $C_n^{t-1}$ combination of attributes can not reconstruct watermark. For example, in the figure 3, if there exist a (3,5)-scheme, the possibility of 1,1,0,0,0 can not reconstruct watermark because it only has two "1", at least 3 is required for reconstruction. $WS_i$ contains $B_j$ ($i \in \{0, 1, ..., m\}$) which the i-column of $B_j$ is zero. Hence, $WS_1$ holds a vector of $B_4, B_5, B_6, B_7, B_8, B_9$ because the value of 1-column in $B_0, B_1, B_2, B_3$ is "1". It ensures that members of $t-1$ $WS_i$ jointly miss at least one additive share and shares is divided equally to each $WS$. Each $WS_i$ holds $|C_{n-1}^{t-1}|$ pieces which is a $|C_{n-1}^{t-1}| * L/m$-bits long binary string. All $WS$ jointly hold $t$ copies of the whole watermark $W$. Greater than or equal to $t$ $WS$ jointly have all required pieces and can thus reconstruct $W$. Lines 4-15 show this process in algorithm 1. Line 16 transforms $B_{i0}, ..., B_{im}$ to $b_{i0}b_{i1}...b_{il}$. Each $b_{il}$ denotes a bit. In encoding phase, algorithm uses a bit every time of insertion. Finally, the result of classification and watermark shares are returned.

**Algorithm 1** Preprocessing Algorithm

**Require:** $D, WA, W, n, t, l$
**Ensure:** $WS, Classifier$
1: **for** each $WA_i \in$ D **do**
2: $Classifier = train(data\ in\ WA_i\ attribute,\ label\ i)$
3: **end for**
4: $combinationVectors = combination(n, t-1)$
5: $piecesSize\ m = |combinationVectors|$
6: $bitsLength\ l = |combination(n-1, t-1)| * L/m$
7: $pieces\ B_0, ..., B_m = split\ W\ into\ m\ pieces$
8: **for** each i $\in$ n **do**
9: $WS_i = []$
10: **for** j=1 to m **do**
11: $vector \in combinationVectors[j]$
12: **if** $i \notin vector$ **then**
13: $WS_i.append(B_j)$
14: **end if**
15: **end for**
16: $WS_i = transform\ B_{i0}, ..., B_{im}\ to\ b_{i0}b_{i1}...b_{il}$
17: **end for**
18: **return** $WS, Classifier$

### B. Watermark Encoding

The insertion algorithm is shown in Algorithm 2. The inputs are original dataset $D$, watermark attributes $WA$, watermark shares $WS$, secret key $\mathcal{K}$ and parameters $l, n, \xi, \gamma$. The algorithm inserts bits of watermark shares into mapped watermark attributes. The insertion algorithm traverses the $D$ and finds the tuple satisfying $\mathcal{S}_1(\mathcal{K}, r.P)\ mod\ \gamma\ equals\ 0$. The primary key binds $\mathcal{K}$ which is computationally infeasible to compute the same value to locate the same tuple using a wrong secret key. The parameter $\gamma$ could control the insertion density. Sequentially, it determines attributes in sets of $WA$ in the same way. The $n$ is the number of watermark attribute $|WA|$. Then, a bit $x$ of $i$-th watermark share is selected by $\mathcal{S}_3(\mathcal{K}, r.P)\ mod\ l\ equals\ 0$ which prevents synchronization error because it makes inserted bit of watermark share is not fixed. The length of the watermark share is $l$. We assume that it is acceptable to change one of $|\xi|$ least significant bits in a small number of numeric values. Finally, a bit $x$ is inserted in $k$-th position amongst $\xi$ when $\mathcal{S}_4(\mathcal{K}, r.P)\ mod\ l\ equals\ 0$. Note that, in insertion algorithm, watermark shares are inserted

into $D$ using $b_0 b_1 b_2 ... b_{l-1}$. The $\mathcal{S}$ could generate a fixed sequence using the same seed. Thus, in terms of insertion and detection, it determines the same position of bit, attribute, and tuple. For watermarked tuples, without having correct $\mathcal{S}$ and comparing watermark copies, it's impossible to extract bits, then recover ownership information to claim ownership because extracted bits can't contain meaningful ownership information. Databases usually allow attributes to assume null values. The algorithm will skip the null value, leaving it unchanged. On average, a fraction $\eta/\gamma$ of tuples will be selected to insert watermarks. Each watermark share bit $x$ is inserted $\eta/(\gamma l n)$ times.

**Algorithm 2** insertion Algorithm

**Require:** $D, \mathcal{K}, WS, WA, l, n, \xi, \gamma$
1: **for** each tuple $r \in$ D **do**
2: **if** $\mathcal{S}_1(\mathcal{K}, r.P)\ mod\ \gamma\ equals\ 0$ **then**
3: $attributeIndex\ i = \mathcal{S}_2(\mathcal{K}, r.P)\ mod\ n$
4: **if** $r.WA_i\ isn't\ null$ **then**
5: $markIndex\ j = \mathcal{S}_3(\mathcal{K}, r.P)\ mod\ l$
6: $bitIndex\ k = \mathcal{S}_4(\mathcal{K}, r.P)\ mod\ \xi$
7: $markBit\ x = b_{ij} \in WS_i$
8: $Set\ Least\ significant\ bit\ k\ of\ r.WA_i\ to\ x$
9: **end if**
10: **end if**
11: **end for**
12: **return** $D$

### C. Watermark Decoding

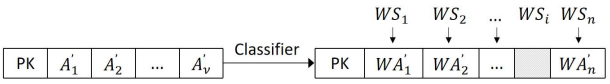



Fig. 4: Classification on attributes in $D'$

The watermark decoding contains four steps: Attribute Classification, Watermark Shares Detection, Majority Voting, and Watermark Recover. The decoding algorithm is shown in Algorithm 3. The $D_w$ of inputs is the watermarked database which may be distorted under types of attacks. The other inputs are watermark $W$, secret key $\mathcal{K}$, classifier $Classifier$ and parameters $l, t, n, \xi, \gamma$. The first step is Attribute Classification, which identifies watermark attributes $WA$. The figure 4 demonstrates this process. The candidate attributes $A'_1, ..., A'_\nu$ in $D_w$ are identified as watermark attributes $WA'_1, ..., WA'_n$. Hence, the identification and order of $WA$ are determined by the pre-trained classifier. The needless attributes will be removed, and missing attributes will be filled with a blank. The connection between each pair of watermark attributes $WA_i$ and watermark share $WS_i$ can be uniquely confirmed.

The next steps are Watermark Shares Detection, Majority Voting, and Watermark Recover. Firstly, the two-dimensional array $zeros$ and $ones$ are initial as 0. Then, the watermark bits are extracted as the insertion algorithm does in lines 8-22. When $\mathcal{S}_1(\mathcal{K}, r.P)\ mod\ \gamma\ equals\ 0$, the select tuple then

**Algorithm 3** Decoding Algorithm

**Require:** $D_w, \mathcal{K}, W, l, n, t, \xi, \gamma, Classifier$
1: $WA_1, ..., WA_n = Classifier(D_w, \{A'_1, ..., A'_\nu\})$
2: **for** i=0 to n-1 **do**
3: **for** j=0 to l-1 **do**
4: $zeros[i][j] = 0$
5: $ones[i][j] = 0$
6: **end for**
7: **end for**
8: **for** each tuple $r \in D_w$ **do**
9: **if** $\mathcal{S}_1(\mathcal{K}, r.P)\ mod\ \gamma\ equals\ 0$ **then**
10: $attributeIndex\ i = \mathcal{S}_2(\mathcal{K}, r.P)\ mod\ n$
11: **if** $isExist(WA'_i)$ **then**
12: $markIndex\ j = \mathcal{S}_3(\mathcal{K}, r.P)\ mod\ L$
13: $bitIndex\ k = \mathcal{S}_4(\mathcal{K}, r.P)\ mod\ \xi$
14: $b_{ij}\ = Least\ significant\ bit\ k\ of\ r.WA_i$
15: **if** $b_{ij} = 1$ **then**
16: $ones[i][j] = ones[i][j] + 1$
17: **else**
18: $zeros[i][j] = zeros[i][j] + 1$
19: **end if**
20: **end if**
21: **end if**
22: **end for**
23: **for** i=0 to n-1 **do**
24: $WS'_i = []$
25: **for** j=0 to l-1 **do**
26: **if** $ones[i][j] \geq zeros[i][j]$ **then**
27: $b_{ij} = 1$
28: **else**
29: $b_{ij} = 0$
30: **end if**
31: $WS'_i.setValue(j, b_{ij})$
32: **end for**
33: **end for**
34: **for** i=0 to n-1 **do**
35: $transform\ b_{i0}b_{i1}...b_{il}\ of\ WS'_i\ to\ B_{i0}B_{i1}...B_{im}$
36: **end for**
37: $majority\ voting\ of\ B_0, ..., B_m\ sequence\ of\ WS'_1, ..., WS'_n$
38: $reconstrcut\ watermark\ W'\ from\ B_0, ..., B_m$
39: **return** $\frac{match(W, W')}{|W|} * 100\%$

is determined by the attribute and position of LSB to extract a watermark share bit. Note that if the select attribute doesn't exist, this tuple will be skipped. The arrays $ones$ and $zeros$ are used to store an extracted bit, indicating the number of times that bit is extracted to be 0 and 1, respectively. If this bit has been changed by the attacker, the extracted bit may not match its original value. In generality, some error bits can't decrease detection accuracy in that each bit of watermark share will be inserted many times. As a rule, watermarking techniques should have error correction mechanisms like voting or cyclic redundancy check. Our technique has capable of error correction of watermark on bits and attributes,

respectively. Thus, secondly, by Majority Voting step on $b_{ij}$ of watermark shares, bit errors are corrected as a result of malicious attacks, and the template of $WS^{'}$ is obtained. Line 23-33 shows this step, where $b_{ij}$ is the voting result of $j$-th bit of watermark share $WS_i^{'}$. In the Watermark Recover step, each $WS^{'}$ is transformed to corresponding $B_{i0}, ..., B_{il}$ of which the particular combination is determined in Preprocessing. Then another majority voting is performed to correct possible error of $B_0, ..., B_m$ from $WS_1^{'}, ..., WS_n^{'}$. Finally, pieces $B_0, ..., B_m$ construct watermark $W^{'}$. The ownership could be claimed if the recovered $W^{'}$ matches $W$ with high accuracy.

Blind means that it is not required to have the original database or watermark in detection. In our algorithm, preprocessing introduces a machine learning technique that provides the capability of identification for watermarking attributes. And secret sharing provides an assignment scheme of watermark pieces. Original database or watermark aren't contained in the detection process. Hence, our algorithm is blind. For watermarking techniques, it assumes that the insertion and detection algorithm should be public. Ownership information must lie only in the choice of the secret key. In our algorithm, the new mechanism helps to reorder attributes before detection and correct errors after detection, respectively. That doesn't mean weak the work of the secret key that determines our algorithm is key-based. Then, we give the analysis of detecting a specific watermark in the non-watermarked dataset. Repeated independent trails are Bernoulli trails. And that has only two probabilities $p$ and $q$, where $q = 1 - p$. Let $b(k; n, p)$ be the probabilities such that there are n trials, probabilities $p$ for success, and $q$ for failure. Then

$$b(k; n, p) = \binom{n}{k} p^k q^{n-k} \tag{1}$$

The cumulative binomial probability is

$$B(k; n, p) = \sum_{i=k+1}^{n} b(i; n, p) \tag{2}$$

If the detection algorithm performs on non-watermarked data, it obtains some binary string $b_0 b_1 ... b_{l-1}$ as a potential watermark share. Let $b_i$ be extracted $\omega_i$ times ($\omega_i > 0$). The $\omega_i \approx \eta/\gamma n$, where $\eta$ is the number of tuples, $\gamma$ is the density parameter, $n$ is the number of watermark attributes. Besides, the $\tau$ is the threshold probability of majority voting for a bit. Each extract is considered as zero or one with a probability of 0.5, which is modeled as an independent Bernoulli trial. The $b_i$ is detected to be zero or one with the same probability

$$B(\lfloor \tau\omega_i \rfloor ; \omega_i, p) = \sum_{j=\lfloor \tau\omega_i \rfloor + 1}^{\omega_i} b(j; \omega_i, 0.5) \tag{3}$$

Thus, the probability of a possible watermark share is $\prod_{i=0}^{l-1} 2B(\lfloor \tau\omega_i \rfloor ; \omega_i, p)$, where the factor 2 is that each bit may be either zero or one. The number of possible binary strings is $2^l$. The number of watermark shares is $|WS|$. Thus, a binary string match a valid watermark share is $\frac{|WS|}{2^l}$. For a watermark share $WS$, the overall false hit rate is

$$FHR^{WS} = \frac{|WS|}{2^l} \prod_{i=0}^{l-1} 2B(\lfloor \tau\omega_i \rfloor ; \omega_i, p) \tag{4}$$

By the equation 4, the value is close to 0 if l $\gg log|WS|$. Hence, the false hit rate of our algorithm is low. The complexity of the insertion algorithm is O($\eta$). The complexity of decoding consists of three steps: the first is watermark share detection, the complexity is O($\eta$); the second, the complexity of majority voting is O($n * l$); the third, the complexity of correction of watermark pieces is O($n * m$), where $\eta$ is the number of tuples, $n$ is the number of watermark attributes, $l$ is the length of bits string of watermark share, $m$ is the number of pieces of the watermark. Thus, the decoding of AAP is efficient with low complexity O($\eta$+$n * l$+$n * m$).

TABLE II: Default parameters

| Parameters | $\gamma$ | $\xi$ | $\tau$ | $n$ | $t$ |
|---|---|---|---|---|---|
| Value | 50 | 3 | 50% | 10 | 3 |

## V. Experiments and Results

In this section, we report the experimental results. Experiments are conducted on Intel Core i7 with CPUs of 3.60 GHz and RAM of 16GB. Algorithms were implemented with JAVA. Experiments are performed using the Forest Cover Type (FCT) data set [2]. The FCT has 581,012 rows in which each tuple contains 10 integer attributes, 1 categorical attribute, and 44 Boolean attributes. We added an extra attribute ID as the primary key, 4 attributes of integer type. We select ten attributes for watermarking. The experiments are repeated many times, and the average result is calculated. The default parameters are described in table II. Hence, our algorithm ensures that only three remaining attributes can still recover complete ownership information. The result shows that our algorithm is efficient and robust against subset, attribute, and multifaceted attacks. It is difficult to remove the watermark under these attacks. We evaluate our technique (denoted by AAP) against the Agrawal-Kiernan's [4] technique (denoted by AHK), Yuan's technique [13] (denoted by DEM) and Kamran-Suhail-Farooq's technique (denoted by KSF).

### A. Watermarking Overhead

Two experiments assess the computational cost of insertion detection in figure 5 by varying different tuples on selected configurations and datasets. Reading the entire data from disk and writing the data to disk are not contained in execution times. The execution time of AAP and AHK algorithms is low. It can be seen that the execution times of these algorithms increase as tuples increase. For 0 $\leq$ tuples $\leq$ 500000, the execution time of AAP and AHK is less than 1s. Moreover, the execution time of DEM and KSF is high.

[2]kdd.ics.uci.edu/databases/covertype/covertype.html

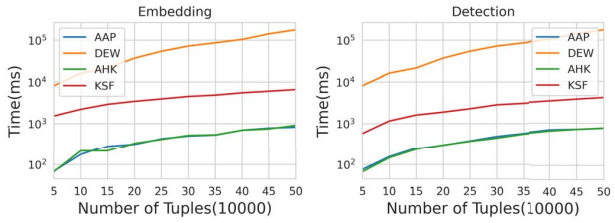



Fig. 5: Execution times of AAP algorithm

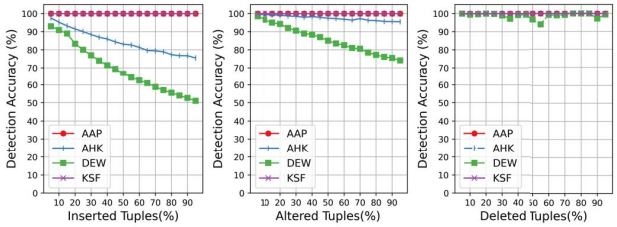



Fig. 6: Robustness of algorithms against subset attack

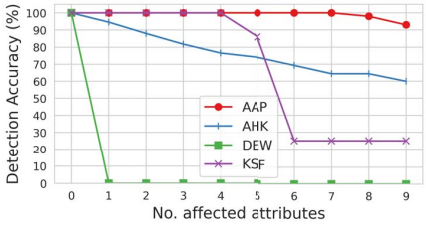



Fig. 7: Robustness of algorithms against attribute attack

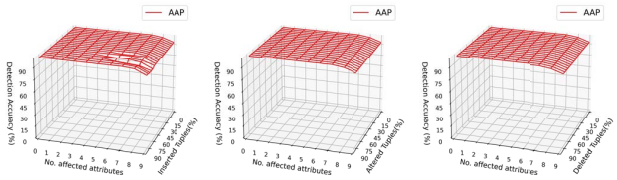



Fig. 8: Robustness of AAP algorithm against multifaceted attack

### B. Attack Analysis

Consider data owner generates a dataset $D_w$ by inserting a watermark $W$ in the dataset $D$ using our insertion algorithm. An attacker wants to destroy watermarks by attacks. At the same time, the attacker preserves data quality so that it remains useful as well. We assume that the attacker can't access to the original dataset $D$, watermark $W$, secret key $\mathcal{K}$ and parameters used in insertion algorithm ( e.g. $\gamma$, $\xi$, $\tau$, $n$, $t$). Hence, the watermark shares ($WS$) and watermark attributes ($WA$) are also inaccessible to the attacker. Besides, the knowledge about the detail of secret sharing or classification cannot help the attacker to extract watermark when leaks watermark $W$, parameters $n$, $t$ and secret key $\mathcal{K}$.

*1) Subset Attack:* For subset attacks, the attacker may randomly insert/alter/delete tuples from the watermarked dataset. Then the detection algorithm is performed. Figure 6 shows the experimental results. Our algorithm successfully extracts the watermarks with 100% accuracy even when over 95% of the tuples are inserted/altered/deleted. Besides, KSF has the same robustness with 100% accuracy even when over 95% of the tuples are inserted/altered/deleted. However, as the number of distorted tuples increases, the detection accuracy of AHK and DEW decreases. An insertion attack may cause more loss of detection accuracy for AHK and DEW algorithms. The attacker alters the attribute values intending to flip the watermark bits. AHK achieves 100% detection accuracy. DEW has a worse result in terms of alteration attacks. All algorithms are robust against deletion attack because the number of watermarks and the number of matches decrease simultaneously. The high robustness of our watermarking algorithm is due to the redundancy of watermark bits in insertion phase and majority voting in detection phase.

*2) Attribute Attack:* In the real world, these attribute attacks cause the same consequence: the detection algorithm would locate false attributes to extract watermarks. Hence, measuring the number of affected attributes is fair for the robustness test. The figure 7 shows the results. Due to the identification of watermark attributes and correction of extracted watermarks, our algorithm is robust against attribute attack even when over 9 attributes are false. The worst case is DEW. When greater than one attribute loses, detection fails with 0% accuracy. The locating of DEW is distorted under attribute attack in that it uses the watermark itself to locate the position of watermarks. The false location cause failure detection. As the number of affected attributes increases, the detection accuracy of AHK and KSF decreases. When over 5 attributes are affected, the detection accuracy of KSF drops steeply because of the failure of majority voting of data partitioning. The detection accuracy of AHK decreases in that the number of false bits increases.

*3) Multifaceted Attribute Attack:* We have also performed experiments for sophisticated attacks by combining subset attack with attribute attack. The scenario is that attacker launches a type of subset attack, then destroys some attributes. Note that it is not rare because destroying attributes is easy. To show the results clearly, the figure 8 shows the robustness of our AAP algorithm against multifaceted attack. Our algorithm is robust against multifaceted attack as well. When over 9 attributes are affected, and over 95% tuples are inserted/altered/deleted, the detection accuracy remains up to 90%. The reason behind this desirable behavior is that the majority voting corrects the errors introduced by subset attack, and secret sharing corrects the errors introduced attribute attack. At the same time, these two mechanisms do not affect each other. The figure 9 show the robustness of compared algorithms against multifaceted attack. The detection accuracy of DEW is low because of attribute attack. As the number of affected attributes increases, the detection accuracy of AHK and KSF decreases under multifaceted attack.

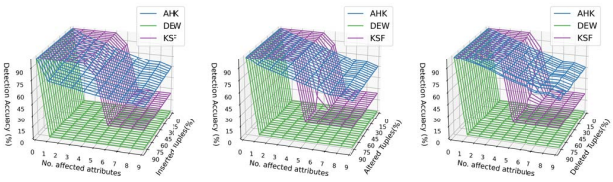



Fig. 9: Robustness of compared algorithms against multi-faceted attack

## VI. Conclusion

This paper proposes a highly robust algorithm against insertion, deletion, alteration attacks, and even attribute attacks by introducing classification, secret sharing, and majority voting mechanisms. At the same time, it is efficient and easy to implement. Note that our algorithm is restricted to numeric data. The results of our experiments on a real-world dataset substantiate our claims. In the future, we are looking to find an attribute-attack-proof solution for both normal and primary key attributes.

## Acknowledgment

This work was supported by National Science and Technology Major Project of China under the Grant No.2016ZX05047003.

## References

[1] C. Podilchuk and W. Zeng, "Image-adaptive watermarking using visual models," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 4, pp. 525–539, 1998.

[2] F. Hartung and M. Kutter, "Multimedia watermarking techniques," *Proc. IEEE*, vol. 87, no. 7, pp. 1079–1107, 1999.

[3] Y. Kim, K. Moon, and I. Oh, "A text watermarking algorithm based on word classification and inter-word space statistics," in *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*. IEEE Computer Society, 2003, pp. 775–779.

[4] R. Agrawal and J. Kiernan, "Watermarking relational databases," in *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*. Morgan Kaufmann, 2002, pp. 155–166.

[5] J. P. Stern, G. Hachez, F. Koeune, and J. Quisquater, "Robust object watermarking: Application to code," in *Information Hiding, Third International Workshop, IH'99, Dresden, Germany, September 29 - October 1, 1999, Proceedings*, ser. Lecture Notes in Computer Science, A. Pfitzmann, Ed., vol. 1768. Springer, 1999, pp. 368–378.

[6] R. Sion, M. J. Atallah, and S. Prabhakar, "Rights protection for categorical data," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 7, pp. 912–926, 2005.

[7] J. Franco-Contreras and G. Coatrieux, "Robust watermarking of relational databases with ontology-guided distortion control," *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 9, pp. 1939–1952, 2015.

[8] S. Iftikhar, M. Kamran, and Z. Anwar, "RRW - A robust and reversible watermarking technique for relational data," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 4, pp. 1132–1145, 2015.

[9] H. Wang, X. Cui, and Z. Cao, "A speech based algorithm for watermarking relational databases," in *International Symposium on Information Processing, ISIP 2008 / International Pacific Workshop on Web Mining, and Web-Based Application, WMWA 2008, Moscow, Russia, 23-25 May 2008*, F. Yu and Q. Luo, Eds. IEEE Computer Society, 2008, pp. 603–606.

[10] K. Jawad and A. Khan, "Genetic algorithm and difference expansion based reversible watermarking for relational databases," *J. Syst. Softw.*, vol. 86, no. 11, pp. 2742–2753, 2013.

[11] G. Gupta and J. Pieprzyk, "Reversible and blind database watermarking using difference expansion," *Int. J. Digit. Crime Forensics*, vol. 1, no. 2, pp. 42–54, 2009.

[12] M. Kamran, S. Suhail, and M. Farooq, "A robust, distortion minimizing technique for watermarking relational databases using once-for-all usability constraints," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 12, pp. 2694–2707, 2013.

[13] S. Yuan, J. Yu, P. Shen, and C. Chen, "Verify a valid message in single tuple: A watermarking technique for relational database," in *Database Systems for Advanced Applications - 25th International Conference, DASFAA 2020, Jeju, South Korea, September 24-27, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, Y. Nah, B. Cui, S. Lee, J. X. Yu, Y. Moon, and S. E. Whang, Eds., vol. 12112. Springer, 2020, pp. 54–71.

[14] R. Agrawal, P. J. Haas, and J. Kiernan, "Watermarking relational data: framework, algorithms and analysis," *VLDB J.*, vol. 12, no. 2, pp. 157–169, 2003.

[15] G. S. Xinchun Cui, Xiaolin Q, "A weighted algorithm for watermarking relational databases," *Wuhan Univ J Nat Sci 2007*, vol. 12, no. 1, pp. 79–82, 2007.

[16] F. Guo, J. Wang, and D. Li, "Fingerprinting relational databases," in *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), Dijon, France, April 23-27, 2006*, H. Haddad, Ed. ACM, 2006, pp. 487–492.

[17] X. Zhou, M. Huang, and Z. Peng, "An additive-attack-proof watermarking mechanism for databases' copyrights protection using image," in *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007*, Y. Cho, R. L. Wainwright, H. Haddad, S. Y. Shin, and Y. W. Koo, Eds. ACM, 2007, pp. 254–258.

[18] R. Sion, M. J. Atallah, and S. Prabhakar, "Rights protection for relational data," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 12, pp. 1509–1525, 2004.

[19] M. Shehab, E. Bertino, and A. Ghafoor, "Watermarking relational databases using optimization-based techniques," *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 1, pp. 116–129, 2008.

[20] Y. Li, V. Swarup, and S. Jajodia, "Constructing a virtual primary key for fingerprinting relational data," in *Proceedings of the 2003 ACM workshop on Digital rights management 2003, Washington, DC, USA, October 27, 2003*, M. Yung, Ed. ACM, 2003, pp. 133–141.

[21] M. Gort, E. A. Diaz, and C. F. Uribe, "A highly-reliable virtual primary key scheme for relational database watermarking techniques," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2017.

[22] B. Schneier, "Applied cryptography second edition: Protocols, algorithms and source code in c," *government information quarterly*, vol. 13, no. 3, p. 336, 1996.

[23] S. R. Safavian and D. A. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst. Man Cybern.*, vol. 21, no. 3, pp. 660–674, 1991.

[24] P. Gallinari, S. Thiria, and F. Fogelman, "Multilayer perceptrons and data analysis," in *Proceedings of International Conference on Neural Networks (ICNN'88), San Diego, CA, USA, July 24-27, 1988*. IEEE, 1988, pp. 391–399.

[25] S. K. Shevade and S. S. Keerthi, "A simple and efficient algorithm for gene selection using sparse logistic regression," *Bioinform.*, vol. 19, no. 17, pp. 2246–2253, 2003.

[26] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[27] C. Asmuth and J. Bloom, "A modular approach to key safeguarding," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 208–210, 1983.

[28] M. Ito, A. S. Nonmember, T. N. Member, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan (Part III Fundamental Electronic Science)*, vol. 72, no. 9, pp. 56–64, 1989.

[29] R. Cramer, I. Damgård, and Y. Ishai, "Share conversion, pseudorandom secret-sharing and applications to secure computation," in *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, ser. Lecture Notes in Computer Science, J. Kilian, Ed., vol. 3378. Springer, 2005, pp. 342–362.