Partitioning-insensitive Watermarking Approach for Distributed Relational Databases

Sapana Rani, Dileep Kumar Koshley, and Raju Halder

Indian Institute of Technology, Patna,
{sapana.pcs13, dileep.pcs15, halder}@iitp.ac.in

Abstract. This paper[1] introduces an efficient watermarking approach for distributed relational databases, which is generic enough to support database outsourcing and hybrid partitioning. Various challenges, like partitioning and distribution of data, existence of replication etc., are addressed effectively by watermarking different partitions using different sub keys and by maintaining a meta-data about the data distribution. Notably, the embedding and detection phases are designed with the aim of making embedded watermarks partitioning-insensitive. That means, database partitioning and its distribution do not disturb any embedded watermark at all. To the best of our knowledge, this is the first proposal on watermarking of distributed relational databases supporting database outsourcing, its partitioning and distribution in a distributed setting.

Keywords: Watermarking, Distributed Databases, Security

1 Introduction

Distributed Database System has always been an efficient solution to manage large-scale databases over computer networks [2]. In recent years, adapting cloud-based technology to avail its efficient and cost-effective services are continuously gaining paramount attention from both academia and industry. For instance, cloud-based Database-as-a-Service (DBaaS), such as Amazon Relational Database Service (RDS) [3], Microsoft SQL Azure [4], etc., is now-a-days appealing for organizations to outsource their databases. In particular, a shared platform (e.g., database server hardware and software) is provided to host multiple outsourced databases, leading to a scalable, elastic and economically viable solution. Clients can easily deploy their own databases in the cloud to avail all required services without investing much on resources [5]. In a cloud-based distributed database system, data owners outsource their databases to a cloud-based service provider that partitions and distributes them among multiple servers interconnected by a communication network.

Unfortunately, outsourcing valuable databases to third-party service providers, without taking proper precaution, may increase the possibility of certain typical attacks, such as copyright infringement, data tampering, integrity violations, piracy, illegal redistribution, ownership claiming, forgery, theft, etc [6].

[1] This work is a revised and extended version of [1].

Database watermarking has emerged as a promising technique to countermeasure the above-mentioned threats [6]. This embeds some kind of information (known as watermark) into the data using a secret key, and later extracts the same, on demand, to reason about the suspicious data. For example, suppose a watermark W is embedded into an original database using a private key K which is known only to the owner. On receiving any suspicious database, the owner may perform a verification process using the same private key K by extracting and comparing the embedded watermark (if present) with the original watermark information W.

In the context of centralized database systems, a wide range of works on the watermarking of centralized databases has been proposed over the past decades [7–17]. Specifically, they are based on random bit flipping [8, 9], fake tuple insertion [18], random bit insertion [17, 19], tuple reordering [13], binary image generation [15, 20], generation of local characteristics like range, digit and length frequencies [21], matrix operations [22], etc. In general, they are categorized into *distortion-based* and *distortion-free* depending on whether the embedded watermark distorts the underlying database content or not. These techniques are designed specifically to address the security risks in centralized databases only, making them completely unadaptable to the scenario where databases are partitioned and distributed over a network. As already mentioned, popular applications where people often face such issues include, for example, cloud-based Database-as-a-Service (DBaaS), such as Amazon Relational Database Service (RDS), Microsoft SQL Azure, etc. In general, the use of centralized database watermarking approaches directly in a distributed setting gives rise to various challenges, including (1) distribution of data, (2) existence of replication, (3) preservation of watermarks while performing partitioning and distribution by third party, (4) robustness, (5) efficient key management, etc. These challenges become more prominent when data-partition and data-distribution are done by untrusted third parties (who are different from data owners) as for example in case of cloud-based database as a service model.

To the best of our knowledge, till now there is no significant contribution in case of distributed relational database watermarking, especially for distributed cloud-based database-as-a-service scenarios. Although two related works in this direction are found in [23, 24], however they have not considered any core properties of distributed scenario during watermark embedding and detection. Moreover, the proposal does not consider any kind of relational databases and their partitioning over the distributed environment. Authors in [23], although title refers, have not addressed any challenge in distributed database scenario.

Motivating from the above concerns, in [1] we have introduced a preliminary proposal on distributed database watermarking that supports database outsourcing and hybrid partitioning. Although we referred AHK algorithm [9] as partition-level watermarking algorithm, this may have several limitations: AHK algorithm marks only one attribute in a tuple at a time. Therefore, with n vertical partitions, the algorithm should execute the AHK algorithm n times, allowing only one attribute to get marked in a particular partition. In fact, the

prime challenge here is to decide which particular part of the tuple will belong to which particular partition.

In this paper, we further strengthen the proposal by designing a novel watermarking algorithm which overcome the above mentioned limitations. The watermarking algorithm, in a single execution, can decide to which partition a part of the tuple actually belongs. In other words, re-executing of the complete watermarking algorithm is not required for each vertical partition. Additionally, in a particular partition, we can decide a fraction of attributes (rather than a single attribute) to be marked randomly. The combination of attributes to be marked is decided by secret parameter, increasing the robustness of the approach.

In particular, our main contributions in this paper are:

- Proposing efficient watermarking approach for distributed relational databases, which is generic enough to support database outsourcing and hybrid partitioning.
- Effective treatment to the above-mentioned challenges by watermarking different partitions using different sub-keys and by maintaining a meta-data about the data distribution, without revealing any secret to the third-party.
- Efficient key management using Mignotte's k out of n secret sharing scheme [25], improving the robustness of the scheme.
- The design of embedding and detection phases with the aim of making embedded watermarks partitioning-insensitive. That means, database partitioning and its distribution do not disturb any embedded watermark at all.
- Experimental evaluation on benchmark datasets to establish the effectiveness of our approach in presence of various attacks.

The structure of the paper is as follows: Section 2 discusses various works in literature related to database watermarking. Sections 3 and 4 describe the proposed watermark embedding and detection techniques respectively, by illustrating with suitable examples. Analysis on the experimental evaluation results is reported in section 5. A detail comparative study *w.r.t.* the literature is reported in section 6. Finally we conclude our works including the future plans in section 7.

2 Related Works

This section briefly discusses the state-of-the-art on the database watermarking techniques in the literature.

The idea to secure a database of map information (represented as a graph) by digital watermarking technique was first given by Khanna and Zane in 2000 [26]. The first watermarking scheme for relational databases was given by Agrawal et al in 2002 [9]. They embed the watermark in the least significant bits (LSB) of a particular bit position of some of the selected attribute of some of the selected tuples based on the secret parameters. Started with these pioneer works, a wide range of watermarking techniques for centralized database has been proposed [7–17].

Broadly, the existing approaches are categorized into *distortion-based* [8–11, 16, 17] and *distortion-free* [7, 12–15] watermarking techniques, based upon whether distortion occurs or not in the underlying data. The distortion introduced when embedding watermarks should not affect the usability of the data. Authors in [8] have applied data flow analysis to detect variant and invariant part in the database and subsequently watermarked the invariant part. Authors in [10] proposed a reversible-watermarking technique which allows to recover the original data from the distorted watermarked data. Image as watermark is embedded at bit-level in [11]. Approaches in [16, 17] are based on database-content - the characteristics of database data is extracted and embedded as watermark into itself. A recent survey by Xie et al. is reported in [27] with special attention to the distortion-based watermarking. Unlike distortion-based techniques, the distortion-free watermarking techniques generate watermark from the database itself. In [13, 20], hash value of the database is extracted as watermark information. Approaches in [7, 14, 15] are based on the conversion of database relation into a binary form to be used as watermark. Authors in [15] used the Abstract Interpretation for verifying integrity of relational databases. A comprehensive survey on various types of watermarks and their characteristics, possible attacks, and the state-of-the-art can be found in [6].

Watermarking schemes can also be classified as *robust* [28, 29] and *fragile* [13, 17, 21, 22]. Generally, the digital watermarking for integrity verification is called fragile watermarking as compared to robust watermarking for copyright protection. Recently proposed fragile watermarking techniques include [30, 31]. In [30], the proposed technique establishes a one-to-one relationship between the secret watermark and the relative order of tuples in a group. Watermark generation in [31] is based on local characteristics of the relation itself such as frequencies of characters and text length.

Recently in [32], we have proposed watermarking for large scale relational databases by adapting the potential of MapReduce [33] paradigm. The experimental results demonstrated a significant improvement in watermarking cost for distortion free watermarking with respect to the existing sequential algorithms.

To the best of our knowledge, till now there is no significant contribution in case of watermarking of distributed relational database systems. Although two related works in this direction are found in [23, 24], however they have not considered any core properties of distributed scenario during watermark embedding and detection. To be more precise, the authors in [24] proposed a real-time watermarking technique for digital contents which are distributed among a group of parties in hierarchical manner. One such example is the distribution of digital works over the Internet involving several participants from content producers to distributors to retailers and finally to customers. The main idea is to perform multilevel watermarking in order to detect attack possibly occurred at any particular level. Unfortunately, their proposal has not considered any kind of relational databases and their partitioning over distributed environment. The major drawback in [24] is that the data owner has to extract all the watermarks

from top to bottom in the hierarchy during verification. Authors in [23], although title refers, have not addressed any challenge in distributed database scenario.

3 Watermarking Technique for Distributed Databases

This section proposes a generic watermarking technique for distributed databases. The proposal is based on the scenario where database owner outsources data to a third party, assuming that the third party has the required resources to manage it. Some of the challenges addressed by the proposed technique are: (1) Distribution of data, (2) Existence of replication, (3) Non-disturbance of the embedded watermark during partitioning and distribution, (4) Robustness, etc.

The watermark embedding phase consists of the following steps:

3.1 Step 1: Initial exchange of partition information

Data owner will initiate this process to exchange some basic information with the third party in order to obtain some initial information about the partitioning and distribution of the database.

Let *DB_schema* be a relational database schema. Let *INF* be a set of specifications on the database and its associated applications, which must be preserved after partitioning and distribution by a third party. For example, *INF* may include confidentiality and visibility constraints [34], user access information [2], query behaviours [35], etc. To start this process, the data owner provides *DB_schema* and *INF* to the third party. As a result, the third party will send back to the owner a *partition overview* of the database. This *partition overview* includes information about the set of partitions to be followed in future by the third party.

Let us formalize the *partition overview*: Let *R_schema* be a schema of a database relation belonging to *DB_schema*. The horizontal partitioning of *R_schema* is formally represented by $\langle R\_schema, f_h \rangle$ where f_h is a partial function defined over the set of all attributes A in *R_schema* (i.e. $f_h : A \nrightarrow 2^{\Phi}$ where Φ is the set of all possible well-formed formulas defined on A in first order logic) [36]. In other words, f_h which is expressed in first order predicate formulas on attributes, represents properties of database tuples. The horizontal partitioning of tuples in an instance of *R_schema* is performed based on the satisfiability of f_h. Given $\Phi = \{\phi_1, ..., \phi_m\}$, since there are at most $2^{|\Phi|}$ number of horizontal partitions depending on the satisfiability of predicates Φ, we will follow the following convention: a horizontal partition is represented by h, where h is the decimal conversion of truth values of $\{\phi_1, ..., \phi_m\}$ obtained based on their satisfiability by its tuples. For example, given two properties ϕ_1 and ϕ_2, if a tuple t satisfies $\neg\phi_1 \wedge \neg\phi_2$ then t is assigned to the partition-0 (which is decimal equivalent of truth value "00"). Similarly, if t satisfies $\phi_1 \wedge \neg\phi_2$ then t is assigned to the partition-2 (as the decimal equivalent of truth value "10" is 2) and so on. In the

similar way, we also formalize the vertical partitioning as $\langle R\_schema, f_v \rangle$, where $\wp(A)$ is the power set of A and $f_v(A) \subseteq \wp(A)$.

Observe that the definitions of f_h and f_v depend on *INF* in order to satisfy it. Therefore, in general, the hybrid partitioning is formally defined as $\langle R\_schema, f_h, f_v \rangle$. The partition overview ψ of DB_schema satisfying *INF* is formally defined as

$$\psi \triangleq \{ \langle R\_schema, f_h, f_v \rangle \mid R\_schema \in DB\_schema \}$$

This is worthwhile to mention here that our approach is suitable for static partitioning and infrequent dynamic partitioning [37], where in the later case a re-watermarking is necessary to make the detection partition-independent.

Example 1. Let us illustrate this by a running example. Consider the database relation "T" depicted in Table 1.

	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
t_1	1	123	100	20	15	16	21	35	11	100	15
t_2	2	785	200	29	15	16	28	38	12	150	12
t_3	3	456	300	50	11	160	21	35	22	20	13
t_4	4	320	400	36	155	167	20	35	21	170	14
t_5	5	453	500	40	151	126	27	35	24	160	17

Table 1. Relation "T"

Let us assume that *INF* consists of the following security specifications:

1. Confidentiality constraints, $C = \{A_2 \wedge A_7 \wedge A_8\} \wedge \{A_3 \wedge A_9 \wedge A_{10}\}$. This means that, none of the partitions should contain either A_2, A_7 and A_8 in combination or A_3, A_9 and A_{10} in combination.
2. Visibility constraints, $V = \{ A_1 \wedge A_2 \wedge A_3 \} \vee \{A_6 \wedge A_9 \wedge A_{10}\}$. This means that, there should be at least one partition that contains either A_1, A_2 and A_3 in combination or A_6, A_9 and A_{10} in combination.

Data owner sends T_schema, the schema of relation "T", along with the set of specifications INF to the third party. The third party generates the horizontal partitions F_1 and F_2 after applying $f_h : A \rightarrow \phi$ where $A = \texttt{attribute}(T\_schema) = \{A_0, A_1, ..., A_{10}\}$ and $\phi : (A_3 \leq \texttt{average}(A_3)) \wedge (A_8 \leq \texttt{average}(A_8))$. Since we have only one predicate, we can have at most two horizontal partitions F_1 and F_2, where F_1 satisfies ϕ and F_2 doesn't satisfy ϕ. Satisfying *INF* [34], third party then partitions F_1 and F_2 vertically by applying a suitable function f_v as

$$f_v(F_1) = \{F_{11}, F_{12}\} \qquad and \qquad f_v(F_2) = \{F_{21}, F_{22}\}$$

where

$$F_{11} = \langle \{A_0, A_1, A_2, A_3, A_4, A_5\}, A_3 \leq \texttt{average}(A_3) \rangle$$
$$F_{12} = \langle \{A_0, A_6, A_7, A_8, A_9, A_{10}\}, A_8 \leq \texttt{average}(A_8) \rangle$$
$$F_{21} = \langle \{A_0, A_1, A_2, A_3, A_4, A_5\}, A_3 > \texttt{average}(A_3) \rangle$$
$$F_{22} = \langle \{A_0, A_6, A_7, A_8, A_9, A_{10}\}, A_8 > \texttt{average}(A_8) \rangle.$$

Finally, the third party sends back this partition overview $\psi = \{F_{11}, F_{12}, F_{21}, F_{22}\}$ to the data owner.

3.2 Step 2: Watermarking by data owner

Given a partition overview ψ (provided by the third party) and a secret key K, the data owner embeds watermark into the original database relation R. To this aim, the data owner performs the following two steps:

- *Key Management*: Obtain a set of n different sub-keys $\{K_i \mid i = 1, 2, \ldots, n\}$ from K where n represents the number of partitions obtained from the partition overview ψ (denoted $|\psi|$), and
- *Watermark Embedding*: Embed the watermark W into R using n sub-keys.

Let us describe each step in detail:

Key Management. Since our aim is to make the watermark partitioning-insensitive, the prime challenge here is to select private key K properly and to watermark the database by using K in such a way that partitioning of the database R by third party must not affect the embedded watermark. Watermarking by the data owner considering the future partitioning (yet to be done by third party) leads to following four possibilities:

- Same Watermark, Same Key: Embedding same watermark into different partitions using same key.
- Different Watermark, Same Key: Embedding different watermarks into different partitions using same key.
- Same Watermark, Different Key: Embedding same watermark into different partitions using different keys.
- Different Watermark, Different Key: Embedding different watermarks into different partitions using different keys.

In first and second case, if the watermark is revealed at one site, the key will be exposed and watermarks at other sites also become vulnerable. In the last case, applying different watermarks along with different keys will be a tedious job. Therefore, in our approach, we consider the third scenario, i.e. "Same Watermark, Different Key", in which if somehow the watermark is extracted at one site, it will not expose the watermarks embedded into other database-partitions at other sites. Moreover, this serves the purpose of making watermark detection partition-independent as well. In particular, to achieve our objective, we consider k out of n secret sharing schemes [25, 38].

Various k out of n secret sharing schemes are already proposed in the literature such as: Shamir's scheme [38], Mignotte's scheme [25], etc. It states that given a secret K and n shares, any set of k shares acts as the threshold from which the secret can be recovered. In other words, any set of (k - 1) shares is not enough to reveal K. However in Shamir's scheme, the attacker gets a range of numbers to guess about the secret key even with (k - 1) keys. In our approach,

Algorithm 1 KEY-COMPUTATION

Input : Partition overview ψ, Secret key K
Output : Shares $\{K_i \mid i = 1, 2, \ldots, n\}$ of the secret key K

1: Let $n = |\psi|$ and k be a threshold, where $|\psi|$ represents the number of partitions.
2: Choose n pairwise co-prime integers $m_1, m_2, ..., m_n | (m_1 \times ... \times m_k) > (m_{n-k+2} \times ... \times m_n)$.
3: Select secret key K such that $\beta < K < \alpha$ where $\alpha = (m_1 \times ... \times m_k)$ *and* $\beta = (m_{n-k+2} \times ... \times m_n)$.
4: **for** each $i \in 1$ to n **do**
5: Compute shares of secret key as $K_i = K \bmod m_i$
6: **end for**
7: Return $\{K_i \mid i = 1, 2, \ldots, n\}$.

we use Mignotte's scheme as this leads to small and compact shares [39]. Algorithm 1 provides detail steps of the Mignotte's scheme to obtain n shares of secret key. Here $n = |\psi|$ indicates the number of partitions. We have a secret key K which is partitioned into different shares, $\{K_i \mid i = 1, 2, \ldots, n\}$ that are used in watermarking of various partitions. Observe that this reduces the challenges in managing and distributing large number of independent keys for all database-partitions in distributed settings.

Example 2. Let us illustrate this using the running example. Consider the partition overview $\psi = \{F_{11}, F_{12}, F_{21}, F_{22}\}$ in Example 1. We require four different keys for watermarking of these four partitions. Considering the threshold k equal to 3, the owner has to assume four pairwise co-prime integers such that the product of k smallest numbers should be greater than the product of k - 1 biggest numbers. Suppose they are: $m_1 = 7$, $m_2 = 17$, $m_3 = 3$, $m_4 = 19$. Since $m_1 \times m_2 \times m_3 = 357$ and $m_3 \times m_4 = 57$, it satisfies the condition $m_1 \times m_2 \times m_3 > m_3 \times m_4$. A secret K should be chosen between the range of these two products, let it be $K = 131$. Secret shares are calculated for all n by using $K_i = K \; mod \; m_i$ as follows:

$$K_1 = K \; mod \; m_1 = 131 \; mod \; 7 = 5$$
$$K_2 = K \; mod \; m_2 = 131 \; mod \; 17 = 12$$
$$K_3 = K \; mod \; m_3 = 131 \; mod \; 3 = 2$$
$$K_4 = K \; mod \; m_4 = 131 \; mod \; 19 = 17$$

These set of secret shares or sub-keys K_1, K_2, K_3, K_4 will be used to watermark the relation T at partition-level based on partition overview ψ.

In the rest of the paper, we use the terms "secret share" and "sub-key" synonymously.

Watermark Embedding. In this subsection, we present watermark embedding step by the data owner using partition overview and the set of sub-keys. The primary objective here is to embed same watermark on multiple partitions using

different sub-keys. Let us formalize the distributed watermark embedding, as below:

$$\begin{aligned}
&\texttt{DistWM_Embed}(R, \psi, W, K) \\
&\quad = \bigcup_{i \in 1 \ldots |\psi|} \texttt{WM_Embed}(R^i, W, K_i) \\
&\quad = \bigcup_{i \in 1 \ldots |\psi|} R_w^i \\
&\quad = R_w
\end{aligned}$$

Data owner watermarks the database relation R using shares $\{K_i \mid i = 1, 2, ..., |\psi|\}$, obtained from the secret key K in Algorithm 1. Suppose R^i represents i^{th} partition in the partition overview ψ. Observe that R^i is watermarked using the share K_i. Once watermarked, data owner then outsources all the watermarked relations R_w in the database to the third party.

We formalize our watermark embedding algorithm in Algorithm 2. The descriptions of various notations to be used in the algorithms are depicted in Table 2. Observe that all these parameters are secret to the data owner.

Symbol	Description
γ	fraction of tuples marked during embedding
β	no of bits to be extracted to make the watermark
ℓ	no of attributes available for marking
η	fraction of attributes to be marked
α	detectability level
ξ	no. of least significant bit available for marking in an attribute.

Table 2. Notations used in Algorithm 2

Algorithm 2 takes database relation R, partition overview ψ and secret key shares (in matrix form) as input, and gives watermarked relation R_w as output. Step 2 in the algorithm checks if a tuple should be marked or not. A tuple t is considered for embedding, if modulus of the hash of t's primary key by γ is zero. A suitable example of hash function is MD5 [40]. Step 3 calls `CHECK` function to determine the horizontal partition in which a tuple t belongs to. Given a set of predicates $\phi_1, \ldots, \phi_m$, the `CHECK` function checks which predicates in first order logic are satisfied by t and returns a horizontal partition id h that is the decimal conversion of truth values of $\{\phi_1, ..., \phi_m\}$ based on their satisfiability. After getting the horizontal partition id h, the embedding key $K_{h,v}$ for each vertical partition $\texttt{VP}_v \in \psi$ is computed at Step 5 from the key matrix $K_{2^{|\phi|} \times |\texttt{VP}|}$. In Step 6, the owner compress its original watermark, which is to be embedded, in a length of β bits based on the hash-based computation. Finally, the embedding procedure `Apply_Watermark` is called to watermark the partition by its corresponding sub-key.

The `Apply_Watermark` procedure takes as input a secret key K, watermark wm and the list of attribute values belonging to a particular partition. In Steps 24 and 25, the number of attributes n to be marked is calculated by multiplying the

Algorithm 2 `WM_Embed`

Input : Database relation R, Partition overview ψ, Secret keys matrix $K_{2^{|\phi|} \times |\mathsf{VP}|}$
Output : Watermarked relation R_w

1: **for** each tuple $t \in R$ **do**
2: **if** hash($t.PK$) mod $\gamma = 0$ **then** /* $t.PK$ is the primary key of t */
3: Horizontal partition id $h =$ `CHECK`$(t, \langle \phi_1, ..., \phi_m \rangle)$
4: **for** each vertical partition $\mathsf{VP}_v \in \mathsf{VP}$ **do**
5: Watermark key $wm_{key} = K_{h,v}$
6: $mark =$ `compress`(hash($owner's\ watermark$), β)
7: List $L = [t.PK, t.A_x, ..., t.A_y], \forall A_x, ..., A_y \in \mathsf{VP}_v$
8: `Apply_Watermark`(wm_{key}, mark, L)
9: **end for**
10: **end if**
11: **end for**

12: **function** `CHECK`(TUPLE t, PROPERTIES $\langle \phi_1, ..., \phi_m \rangle$)
13: Create a boolean array b of length m
14: **for** $i = 1; i <= m; i = i + 1$ **do**
15: **if** $t \models \phi_i$ **then**
16: $b[i] = 1$
17: **else**
18: $b[i] = 0$
19: **end if**
20: **end for**
21: return `decimal`(b)
22: **end function**

23: **procedure** `Apply_Watermark`(KEY K, WATERMARK wm, $L = [t.PK, t.A_x, ..., t.A_y]$)
24: no of attributes available for marking, $\ell = |L| - 1$ /* primary key $t.PK$ is unavailable for marking */
25: no of attributes to be marked, $n = \lceil \ell \times \eta \rceil$ /* η is the fraction of attributes to be marked */
26: total number of attribute combinations for embedding, $c = \binom{\ell}{n}$
27: the combination of attributes to be marked, $q =$ hash($K \parallel t.PK$) mod c /*each combination contains the list of n attributes */
28: **for** each attribute $a \in q^{th}$ combination **do**
29: $bit\_index\ b = hash(K \parallel t.PK)$ mod ξ
30: Replace b^{th} LSB bit of a with b^{th} bit of wm
31: **end for**
32: **end procedure**

number of attributes ℓ available for watermarking and the predefined fraction η of attributes to be marked. In order to randomize the set of attribute values to be marked for different tuples, Step 26 computes the total number of combination of attributes. For example, if the total number of attributes available for marking (i.e., ℓ) is 5 and we want to mark 30% (i.e., $\eta = 0.3$) of attributes, then the number of attributes to be marked $n = \lceil \ell \times \eta \rceil = 2$. Therefore, total number of combinations each containing 2 attributes out of 5 is $\binom{\ell}{n} = \binom{5}{2} = 10$. Step 27 chooses q^{th} combination, and in Steps 28-31 all attributes in q^{th} combination are marked by replacing their b^{th} least significant bit with the b^{th} least significant bit of the watermark wm, where b is the bit index computed by modulus operation of primary key hash value with the number of least significant bit available for marking, ξ.

A situation may arise where partition size is too small. In such case, the tuples in the partition can be divided into multiple groups and similarly the

watermark of larger length can be divided into multiple parts, thus enabling to embed one part in one group of tuples.

Example 3. Consider the running example. Given $\psi = \{F_{11}, F_{12}, F_{21}, F_{22}\}$ and the secret sub keys $K_1 = 5, K_2 = 12, K_3 = 2, K_4 = 17$. These keys are stored in 2×2 matrix {{5, 12}, {2, 17}}. Given the partition overview ψ computed in Example 1, its clear that total number of horizontal partition is two and vertical partitions is also two. For tuple t_1, the `CHECK` function returns $h = 0$ because first tuple doesn't satisfy the predicate $\phi : (A_3 \leq \texttt{average}(A_3)) \wedge (A_8 \leq \texttt{average}(A_8))$. Therefore for vertical partition $\{A_0, A_1, A_2, A_3, A_4, A_5\}$, secret key 5 will be used for watermark embedding. Let us assume that the watermark $mark$ = "10" and List $L = [1, 123, 100, 20, 15, 16]$. `APPLY_WATERMARK` procedure determines the number of attributes available for marking $\ell = |L| - 1 = 6 - 1 = 5$ (A_0 is the primary key and hence unavailable for watermarking). Suppose $\eta = 0.3$, then number of attributes to be marked is $n = \lceil \ell \times \eta \rceil = \lceil 5 \times 0.3 \rceil = 2$. Since we are having 5 attributes, out of which 2 have to be marked, hence the number of possible combinations of 2 attributes is $c = \binom{5}{2} = 10$. Assuming that the 1^{st} combination containing $\langle 1, 2 \rangle$ is chosen by step 27. That is, attributes A_1 and A_2 need to be marked. Let $\xi = 2$ and the *bit_index* b computed for these attributes are 0 and 1 respectively. For embedding the watermark wm, data owner replaces the 0^{th} and 1^{st} LSB bit of A_1 (=123) and A_2 (=100) respectively by the corresponding LSB bit of wm, resulting into 122 and 100 respectively. Similarly another vertical partition $\{A_0, A_6, A_7, A_8, A_9, A_{10}\}$ in t_1 is also marked following the similar steps. This continues for the remaining t_2, t_3, t_4 and t_5.

The watermarked relation is shown in Table 3. Data represented in bold denotes watermarked values. Data owner outsources this watermarked relation to the third party for partitioning and distribution.

	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}
t_1	1	**122**	**100**	20	15	16	21	**36**	11	**101**	15
t_2	2	785	200	**25**	**14**	16	28	**39**	12	**146**	12
t_3	3	456	**300**	50	**9**	160	21	35	22	**22**	**13**
t_4	4	**322**	**401**	36	155	167	**20**	**34**	22	170	14
t_5	5	**453**	**500**	40	151	126	27	**35**	**27**	160	17

Table 3. Watermarked Relation "T_w"

3.3 Step 3: Partitioning and distribution by third party

Once the third party receives the watermarked database relation R_w from the data owner, the third party partitions and distributes it as per the partition overview ψ computed before. A partition may be either a subset of attributes (vertically partitioned) or a subset of tuples with common properties (horizontally partitioned) or both (hybrid partitioning). In addition, the third party maintains a metadata table that contains information about the data distribution over the servers. The metadata information consists of partition ID P_i, property description of the data in the partition in the form of first-order formula, the server ID S_j where partition P_i is located, etc.

Example 4. Considering the partition overview ψ, third party first applies function f_h to horizontally partition "T_w" relation into partitions F_1 and F_2 and then f_v to vertically partition into $F_{11}, F_{12}, F_{21}, F_{22}$. The functions f_h and f_v are the same functions as computed during partition overview creation. The resulting watermarked partitions (shown in Table 4) are finally distributed to different servers. The metadata for our running example is shown in Table 5.

A_0	A_1	A_2	A_3	A_4	A_5
1	**122**	**100**	20	15	16
2	785	200	**25**	**14**	16

(a) F_{11}

A_0	A_6	A_7	A_8	A_9	A_{10}
1	21	**36**	11	**101**	15
2	28	**39**	12	**146**	12

(b) F_{12}

A_0	A_1	A_2	A_3	A_4	A_5
3	456	**300**	50	**9**	160
4	**322**	**401**	36	155	167
5	**453**	**500**	40	151	126

(c) F_{21}

A_0	A_6	A_7	A_8	A_9	A_{10}
3	21	35	22	**22**	**13**
4	**20**	**34**	22	170	14
5	27	**35**	**27**	160	17

(d) F_{22}

Table 4. Partitioning and distribution by third party

Partition ID	Partition Description		Server ID
	Schema	Properties	
P_1	$\{A_0, A_1, A_2, A_3, A_4, A_5\}$	$A_3 \leq \texttt{avg}(A_3)$	S_3
P_2	$\{A_0, A_6, A_7, A_8, A_9, A_{10}\}$	$A_8 \leq \texttt{avg}(A_8)$	S_1
P_3	$\{A_0, A_1, A_2, A_3, A_4, A_5\}$	$A_3 > \texttt{avg}(A_3)$	S_2
P_4	$\{A_0, A_6, A_7, A_8, A_9, A_{10}\}$	$A_8 > \texttt{avg}(A_8)$	S_4

Table 5. Metadata

4 Partition-level Watermark Detection

The data owner initiates the detection process if she suspects any of the attacks on her database partitions or on part of it. The main issue here is to know the actual key that was used at the time of watermark embedding. For this purpose, the data owner communicates with the third party to obtain a partition ID P_i and the corresponding server ID S_j based on the matching of the suspicious database data with the property description of P_i in the metadata table.

Once the partition ID P_i is obtained, the owner will use the Mignotte's scheme [25] to obtain the key K_i from secret key K and i^{th} co-prime integer m_i. Algorithm 3 formalizes the watermark detection phase. It takes the suspicious partition F and the secret key K_i as the input, and gives as output the reasoning whether verification is successful or not. Step 3 checks whether the tuple was marked at the time of embedding. Step 4 increments the value of total count, i.e. the total number of tuples marked.

Steps 5-12 follow similar steps as in the case of embedding algorithm and identify the marked positions in the tuples to extract the embedded watermark bits. On matching the extracted watermark with the original watermark in step 13, the match count is incremented in the next step. Finally step 19 checks whether the match count crosses the threshold τ, and if so, the watermark detection is considered as successful.

Algorithm 3 `WM_Detect`

Input : suspicious partition F, secret key K_i
Output : detection result either as "success" or "fail"

1: total count = match count = 0
2: **for** each tuple $t^{'} \in F$ **do**
3: **if** hash($t.PK$) mod $\gamma = 0$ **then**
4: total count = total count + 1
5: $\ell^{'}$ = (no. of attributes in F) - 1 /* $t.PK$ is unavailable for marking */
6: $n^{'} = \lceil \ell^{'} \times \eta \rceil$ // η is the fraction of attributes marked during embedding
7: total number of attribute combinations, $c^{'} = \binom{\ell^{'}}{n^{'}}$
8: $q^{'}$ = hash($K_i \parallel t.PK$) mod $c^{'}$ /* each combination contains the list n attributes */
9: **for** each attribute $a \in q^{'}th$ combination **do**
10: $bit\_index\ b = hash(K_i \parallel t.PK)$ mod ξ
11: b^{th} LSB bit of $W^{'} = b^{th}$ LSB bit of a
12: **end for**
13: **if** $(W^{'} = W)$ **then**
14: match count = match count + 1
15: **end if**
16: **end if**
17: **end for**
18: threshold τ = (total count $\times$ α) /* α is the detectability level */
19: **if** match count $\geq \tau$ **then**
20: verification = "success"
21: **else**
22: verification = "fail"
23: **end if**

Example 5. In the running example let us consider the partition F_{12} in Table 4 as a suspicious one. The data owner will ask third party for the database partition ID (P_i) and the corresponding server ID (S_j). Third party refers to the meta-data in Table 5 to compare the property of suspicious data and replies back with the information $P_i = 2$ and $S_j = S_4$ to data owner. Now the data owner will get the corresponding i^{th} co-prime number m_i (i.e. m_2) and obtains K_i (i.e. K_2) following the Algorithm 1. From the running example 2, $K_2 = K$ mod m_2 $= 131\ mod\ 17 = 12$. To detect the watermark, the data owner invokes Algorithm 3 passing suspicious partition F_{12} and the key $K_2 = 12$ as inputs. Using secret parameters γ, η, let the data owner computes the marked combination in steps 2-8 as A_7, A_9. From these two attribute values, owner will extract two bits '1' and '0' and reconstruct the watermark $W^{'}$ = "10". As $W = W'$, match-count will be increased by 1. Finally if the number of match-count crosses the threshold, the detection is successful.

As an alternative, watermark can also be detected by the application of query preserving approach [35]. Since all servers are equally likely to be suspect, the owner can send an identical query to all the servers. Based on the responses, data owner assigns probabilities to the servers of being suspicious. Further based on a fixed threshold probability, the data owner will get a set of suspicious server IDs. Now the data owner asks the third party for the database partition IDs (P_i's) corresponding to those suspicious servers. Third party then refers to the metadata and replies with a set of corresponding database partition IDs without accessing the schema and properties stored in it. After getting the database

partition IDs, owner will get the corresponding key K_i from algorithm 1. Owner now hit and try among these keys to extract the watermark from the suspicious partition.

Example 6. After getting the suspicious partition F_{12} in Table 4, data owner generates a query and sends it to all the servers. Based on the responses from the servers, let the probabilities assigned to servers S_1, S_2, S_3, S_4 are 0.6, 0.4, 0.5, 0.8 respectively. Assuming the threshold probability as 0.6, the possible suspects are S_1 and S_4, the data owner asks for the corresponding database partition IDs from third party. Subsequently the third party refers to the metadata and replies back the database partition IDs, i.e. 4 and 2. Now the owner will get keys K_4 and K_2 as follows: $K_4 = K \bmod m_4 = 131 \bmod 19 = 17$, $K_2 = K \bmod m_2 = 131 \bmod 17 = 12$. Owner now hit and try among these keys to extract the watermark from the suspicious partition F_{12}.

5 Experimental Analysis

We have performed experiments on a benchmark dataset, namely Forest Cover Type data set[2]. This dataset contains 581012 tuples, each having 10 integer attributes, 1 categorical attribute, and 44 boolean attributes.

No of tuples	$\|\psi\|$	ℓ	ξ	γ	Total_count	Time (msec)
581012	2	5	3	50	11851	429396
581012	4	5	3	50	11851	381082
581012	6	5	3	50	11851	447483
581012	8	5	3	50	11851	490723

Table 6. Results of Watermark Embedding

Partition		**Updation**		**Detection**			
partition	No of tuples	Percent updated	Total_count	Match_count	τ	Time (msec)	Detect
partition 1	581012	0	11851	11851	3555	207675	✓
		30		11099		196237	✓
		60		10635		213598	✓
		90		9813		218389	✓
		99		9469		197051	✓
partition 2	581012	0	11851	11851	3555	264892	✓
		30		10965		256773	✓
		60		10311		234568	✓
		90		9707		201922	✓
		99		9400		278427	✓

Table 7. Detection after partitioning and update attack in case of 2 partitions

We have added an extra attribute *id* to the dataset which serves as primary key, and used all 10 numerical attributes in our experiments. We implemented our algorithms in Java and executed on the system featured with Intel Core i3 processor (2.50 GHz), Windows Operating System, and 4 GB RAM.

In the beginning, we watermark the original dataset using Algorithm 2 with the secret keys obtained from Mignotte's scheme [25] as depicted in Algorithm

[2] University Of California-Irvine KDD Archive: `kdd.ics.uci.edu/databases/covertype/covertype.html`

Partition		Deletion		Detection			
partition	No of tuples after deletion	Percent deleted	Total_count	Match_count	τ	Time (msec)	Detect
partition 1	581012	0	11851	11851	3555	207675	✓
	406710	30		8330		195313	✓
	232406	60		4654		88920	✓
	58103	90		1185		24504	×
	5812	99		137		2534	×
partition 2	581012	0	11851	11851	3555	264892	✓
	406710	30		8330		136118	✓
	232406	60		4773		87027	✓
	58103	90		1221		24504	×
	5812	99		140		2534	×

Table 8. Detection after partitioning and deletion attack in case of 2 partitions

Partition		Updation		Detection			
partition	No of tuples	Percent updated	Total_count	Match_count	τ	Time (msec)	Detect
partition 1	475050	0	9802	9802	2940	201103	✓
		30		9231		188743	✓
		60		8746		185909	✓
		90		8230		215241	✓
		99		8030		201969	✓
partition 2	475050	0	9802	9802	2940	159107	✓
		30		9152		195906	✓
		60		8669		181189	✓
		90		8068		219924	✓
		99		7858		192249	✓
partition 3	105962	0	2049	2049	614	46215	✓
		30		1878		39467	✓
		60		1823		40348	✓
		90		1669		41553	✓
		99		1523		39351	✓
partition 4	105962	0	2049	2049	614	40439	✓
		30		1883		38346	✓
		60		1847		37627	✓
		90		1684		39729	✓
		99		1542		37283	✓

Table 9. Detection after partitioning and update attack in case of 4 partitions

Partition		Deletion		Detection			
partition	No of tuples after deletion	Percent deleted	Total_count	Match_count	τ	Time (msec)	Detect
partition 1	475050	0	9802	9802	2940	201103	✓
	332536	30		6858		121951	✓
	190021	60		3922		71518	✓
	47506	90		995		22023	×
	4752	99		105		1888	×
partition 2	475050	0	9802	9802	2940	159107	✓
	332536	30		6855		118881	✓
	190021	60		3925		69884	✓
	47506	90		998		20360	×
	4752	99		102		1919	×
partition 3	105962	0	2049	2049	614	46215	✓
	74175	30		1445		31315	✓
	42386	60		834		21066	✓
	10598	90		221		4513	×
	1061	99		34		652	×
partition 4	105962	0	2049	2049	614	40439	✓
	74175	30		1448		30403	✓
	42386	60		830		18176	✓
	10598	90		228		4236	×
	1061	99		39		586	×

Table 10. Detection after partitioning and delete attack in case of 4 partitions

1, by varying the number of partitions (i.e., 2, 4, 6, and 8) obtained by applying both horizontal and vertical partitioning.

The fraction of attributes to be marked in a partition, η is taken as 0.5, i.e. 50% of the attributes available for marking are actually marked. Then, we simulate update and deletion attacks on various partitions. In the detection pro-

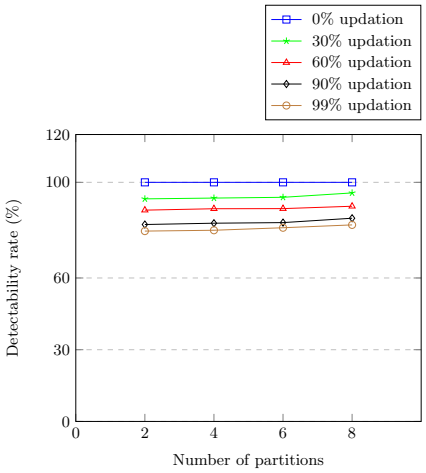

Fig. 1. Watermark detection rate after update attack

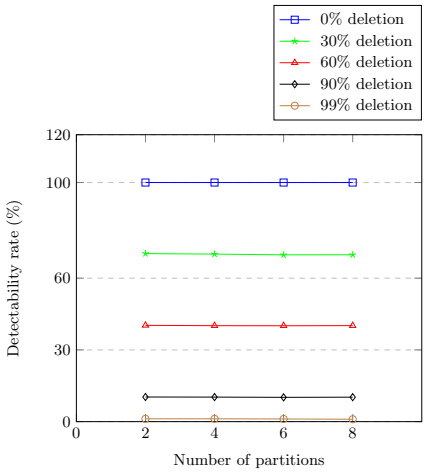

Fig. 2. Watermark detection rate after delete attack

Partition		**Updation**		**Detection**			
partition	No of tuples	Percent updated	Total_count	Match_count	τ	Time (msec)	Detect
partition 1	105963	0	2050	2050	615	59568	✓
		30		1941		39518	✓
		60		1880		38871	✓
		90		1743		41046	✓
		99		1732		40563	✓
partition 2	105963	0	2050	2050	615	35941	✓
		30		1935		40411	✓
		60		1850		40839	✓
		90		1731		43736	✓
		99		1663		47170	✓
partition 3	137742	0	2753	2753	826	60376	✓
		30		2611		53781	✓
		60		2496		52510	✓
		90		2320		50883	✓
		99		2254		48535	✓
partition 4	137742	0	2753	2753	826	53254	✓
		30		2612		46241	✓
		60		2488		48631	✓
		90		2322		47973	✓
		99		2254		48751	✓
partition 5	337307	0	7048	7048	2114	121809	✓
		30		6573		120490	✓
		60		6221		137168	✓
		90		5843		126628	✓
		99		5706		116504	✓
partition 6	337307	0	7048	7048	2114	113988	✓
		30		6546		118265	✓
		60		6130		123761	✓
		90		5751		150714	✓
		99		5649		144750	✓

Table 11. Detection after partitioning and update attack in case of 6 partitions

cess, we use same set of secret parameters as that of embedding phase. We have taken a fixed detectability level $\alpha = 0.3$ to measure the success detectability, i.e. if match count is greater than threshold τ (30% of the total count) then we consider the detection as successful. Table 6 depicts the watermark embedding results (watermark embedding time in millisecond) for various number of partitions in partition-overview.

Let us now discuss the watermark detection after update and delete attacks. For all partitions, we have performed the experiments on $Count = 581012$ tuples by taking $\gamma = 50$. Let us first define detectability rate:

$$detectability\ rate = (Match\_count/Total\_count)\ \times\ 100$$

The results of detection process after update attack in case of 2 partitions is shown in Table 7.

Partition		**Deletion**		**Detection**			
partition	No of tuples	Percent deleted	Total_count	Match_count	τ	Time (msec)	Detect
partition 1	105963	0	2050	2050	615	59568	✓
	74176	30		1443		33902	✓
	42387	60		835		17680	✓
	10598	90		220		4140	×
	1061	99		33		463	×
partition 2	105963	0	2050	2050	615	35941	✓
	74176	30		1443		26535	✓
	42387	60		835		18117	✓
	10598	90		220		4609	×
	1061	99		33		585	×
partition 3	137742	0	2753	2753	826	60376	✓
	96421	30		1949		35758	✓
	55098	60		1156		23975	✓
	13776	90		290		5979	×
	1379	99		26		679	×
partition 4	137742	0	2753	2753	826	53254	✓
	96421	30		1952		38365	✓
	55098	60		1148		23478	✓
	13776	90		281		5589	×
	1379	99		26		464	×
partition 5	337307	0	7048	7048	2114	121809	✓
	236116	30		4880		85868	✓
	134924	60		2814		51280	✓
	33732	90		696		15996	×
	3375	99		72		1439	×
partition 6	337307	0	7048	7048	2114	113988	✓
	236116	30		4872		81899	✓
	134924	60		2809		47013	✓
	33732	90		694		13973	×
	3375	99		72		1583	×

Table 12. Detection after partitioning and delete attack in case of 6 partitions

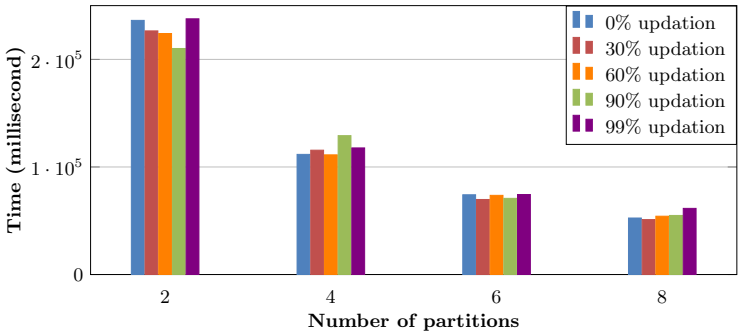

Fig. 3. Average detection time after update attack

Partition		Updation		Detection			
partition	No of tuples	Percent updated	Total_count	Match_count	τ	Time (msec)	Detect
partition 1	189609	0	3997	3997	1199	69628	✓
		30		3811		65827	✓
		60		3596		66698	✓
		90		3396		68332	✓
		99		3314		63773	✓
partition 2	189609	0	3997	3997	1199	68353	✓
		30		3762		68743	✓
		60		3547		74104	✓
		90		3327		67199	✓
		99		3229		63773	✓
partition 3	137750	0	2761	2761	828	50717	✓
		30		2974		47441	✓
		60		2518		49458	✓
		90		2353		49884	✓
		99		2302		48918	✓
partition 4	133750	0	2761	2761	828	49279	✓
		30		2644		49108	✓
		60		2501		49194	✓
		90		2351		53799	✓
		99		2292		49842	✓
partition 5	147691	0	3044	2997	913	53668	✓
		30		2901		53158	✓
		60		2736		59440	✓
		90		2557		56258	✓
		99		2491		57356	✓
partition 6	147691	0	3044	2994	913	53468	✓
		30		2882		51086	✓
		60		2684		51199	✓
		90		2516		55550	✓
		99		2435		66929	✓
partition 7	105962	0	2049	2049	614	37305	✓
		30		1985		35643	✓
		60		1866		44264	✓
		90		1765		48460	✓
		99		1702		55030	✓
partition 8	105962	0	2049	2049	614	38014	✓
		30		1947		38188	✓
		60		1884		39523	✓
		90		1779		39979	✓
		99		1719		39089	✓

Table 13. Detection after partitioning and update attack in case of 8 partitions

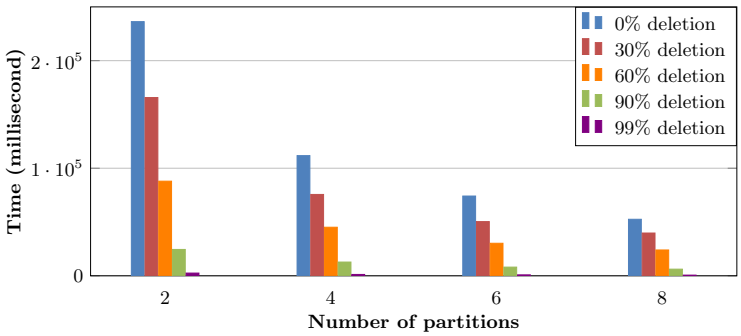

Fig. 4. Average detection time after delete attack

The watermark detectability rate after update attack is depicted in Figure 1. It is to be noted that "Total_count" denotes the number of tuples marked during embedding before updation and "Match_count" represents the number of times we are able to extract our embedded watermark successfully from various partitions after update attack. We have taken the results by randomly updating

Partition		Deletion		Detection			
partition	No of tuples	Percent deleted	Total_count	Match_count	τ	Time (msec)	Detect
partition 1	189609	0	3997	3997	1199	69628	✓
	132728	30		2764		54844	✓
	75845	60		1610		30165	✓
	18962	90		395		8098	×
	1898	99		26		701	×
partition 2	189609	0	3997	3997	1199	68353	✓
	132728	30		2760		48426	✓
	75845	60		1608		28362	✓
	18962	90		395		7749	×
	1898	99		26		649	×
partition 3	137750	0	2761	2761	828	50717	✓
	96426	30		1945		38874	✓
	55101	60		1110		27907	✓
	13776	90		295		5809	×
	1379	99		26		554	×
partition 4	133750	0	2761	2761	828	49279	✓
	96426	30		1946		36597	✓
	55101	60		1095		22382	✓
	13776	90		298		6022	×
	1379	99		25		632	×
partition 5	147691	0	3044	2997	913	53668	✓
	103385	30		2121		43292	✓
	59078	60		1207		24967	✓
	14771	90		302		6245	×
	1478	99		33		786	×
partition 6	147691	0	3044	2994	913	53468	✓
	103385	30		2117		39825	✓
	59078	60		1204		22150	✓
	14771	90		302		6175	×
	1478	99		33		632	×
partition 7	105962	0	2049	2049	614	37305	✓
	74175	30		1445		29214	✓
	42386	60		834		19664	✓
	10598	90		221		4725	×
	1061	99		34		570	×
partition 8	105962	0	2049	2049	38014	614	✓
	74175	30		1445		27050	✓
	42386	60		834		16519	✓
	10598	90		221		4628	×
	1061	99		34		548	×

Table 14. Detection after partitioning and deletion attack in case of 8 partitions

30%, 60%, 90% and 99% tuples of each partition. We can observe that, setting α to 0.3, even after updating 99%, we are able to detect our embedded watermark in some cases. However, owner can tune this value to make a trade off between false positives and false negatives.

The detection results after delete attack in case of 2 partitions is depicted in Table 8. Similarly the detection results after update and delete attacks in case of 4 partitions are shown in Table 9 and Table 10 respectively. Tables 11 and 12 depict the detection results for 6 partitions after update attack and delete attack respectively. Similarly Table 13 and Table 14 depict the detection results for 8 partitions after update attack and delete attack respectively.

Similarly the watermark detection rate after delete attack is shown in Figure 2. The results have been taken by randomly deleting 30%, 60%, 90% and 99% tuples of each partition. The average detection times for all partitions after update attack and delete attacks are shown in Figure 3 and Figure 4.

Figures 5, 6, 7 and 8 represent the detection rate for 2, 4, 6 and 8 partitions respectively.

Based on these experimental results, we establish the following observations:

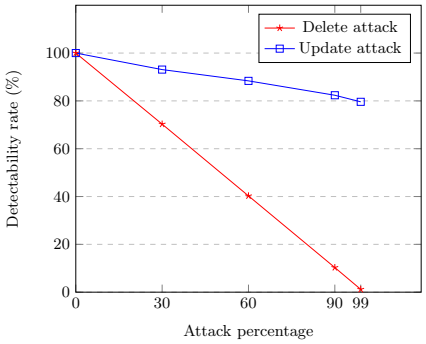

Fig. 5. Percentage of detection for 2 partition

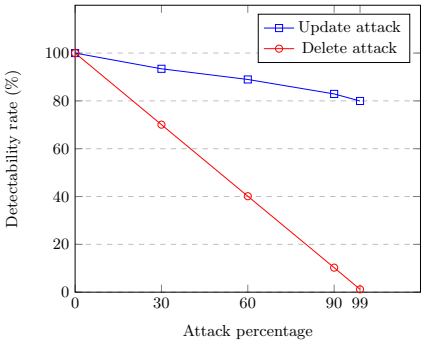

Fig. 6. Percentage of detection for 4 partition

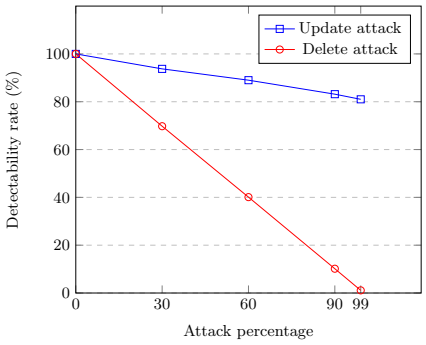

Fig. 7. Percentage of detection for 6 partition

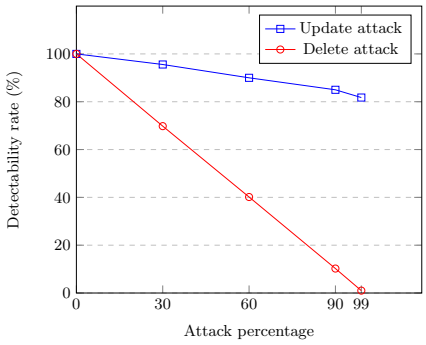

Fig. 8. Percentage of detection for 8 partition

- The watermark is successfully detected even after the attacker updates 99% of the tuples in some cases, considering $\alpha = 0.3$. Therefore, detection success will be increased as we decrease α.
- For 0% updation or deletion, we have 100% detectability rate for all partitions, which is always true.
- Figure 1 and 2 show that detectability rate increases as we increase the number of partitions.
- Figure 1 and 2 show that detectability rate decreases as we increase the percentage of updation and deletion.
- It is clear from Figure 3 and Figure 4 that for the same percentage of updation or deletion, the average detection time decreases as we increase the number of partitions.
- According to Figures 5, 6, 7 and 8, we observe that the detectability rate decreases on increasing the percentage of updation and deletion.

6 Discussions *w.r.t.* the Literature

A wide range of watermarking techniques [7–17] for centralized database has been proposed. Unlike all these, our scenario is based on the cloud-based dis-

tributed database system where data owners outsource their databases to a cloud-based service provider who eventually partitions and distributes them among multiple servers interconnected by a communication network.

In the context of distributed database watermarking, let us briefly describe two existing approaches found in the literature:

El-Bakry et al. [23] The proposed technique, for the purpose of watermarking, changes the structure of relational database schema by adding a new record. This new record is generated by applying a secret function on the original data of each field with the help of a secret key. Though the title refers, they have not addressed any challenge in distributed database scenario. In fact, the major technical contributions have not considered any distributed database scenario at all.

Razdan et al. [24] The watermarking technique is proposed for digital contents which are distributed among a group of parties in hierarchical manner. For example, distribution of digital works over the internet involving several participants from content producers to distributors to retailers and finally to customers. The proposed technique inserts unique watermarks at each transaction stage to provide a complete audit-trail. This hierarchical watermarking of digital contents imposes difficulties during the watermark extraction process as the data owner has to extract all the watermarks from top to bottom.

This is worthwhile to mention here that the authors in [23, 24], however, have not considered any core properties of distributed scenario during watermark embedding and detection. Their proposals do not even consider any kind of database partitioning over the distributed environment.

Advantages of our approach: In this paper, we have proposed a partition-independent database watermarking approach in distributed environment. Since we have embedded same watermark in all the database partitions using different key, the vulnerability of different partitions is minimized. Even if somehow the watermark at one partition is revealed, it will not affect the watermarks embedded in other partitions. The watermark detection can be done on each partition independently. For the purpose of key management we have used k-out of-n secret sharing algorithm [25] which makes our watermark more robust.

Disadvantages of our approach: The initial exchange of partition information between the data owner and the third party service provider induces an overhead in our proposal. The proposed framework relies on the assumption that the service provider always agrees to the previously computed partition-overview at a later stage of partitioning and distribution.

Despite these overheads, our watermarking approach best suites in the cloud computing scenario where data owners outsource their watermarked databases to the third party service providers. The key management scheme makes the detection process partition independent and also an attack at one partition doesn't

affect the other partitions at all. The experimental analysis shows that detectability rate increases as we increase the number of partitions since the match count also increases. As obvious, the detectability rate decreases as we increase the percentage of updation and deletion. For 0% updation or deletion, we have 100% detectability rate for all partitions, which is always true.

7 Conclusions and Future plan

In this paper, we proposed a novel watermarking technique for distributed databases that supports hybrid partitioning. The algorithms are designed to be partitioning-insensitive. The key management scheme that we have considered makes the watermark more robust against various attacks, as if anyhow some partitions are attacked it will not affect any watermark in other database-partitions. The experimental results show the strength of our approach by analyzing the detection rate with respect to random modification and deletion attack. To the best of our knowledge, this is the first work on watermarking of databases in distributed setting that supports database outsourcing and its partitioning and distribution. The future work aims to extend it to the case of big data in cloud computing environment.

ACKNOWLEDGMENT

This work is partially supported by the Council of Scientific and Industrial Research (CSIR), India.

References

1. Rani, S., Koshley, D.K., Halder, R.: A watermarking framework for outsourced and distributed relational databases. In: International Conference on Future Data and Security Engineering, Springer (2016) 175–188
2. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer Science & Business Media (2011)
3. Amazon Relational Database Service. `https://aws.amazon.com/rds/.`
4. Microsoft Azure SQL Database. `https://azure.microsoft.com/en-in/services/sql-database/`
5. Curino, C., Jones, E.P., Popa, R.A., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., Zeldovich, N.: Relational cloud: A database-as-a-service for the cloud. (2011)
6. Halder, R., Pal, S., Cortesi, A.: Watermarking techniques for relational databases: Survey, classification and comparison. J. UCS **16**(21) (2010) 3164–3190
7. Bhattacharya, S., Cortesi, A.: A generic distortion free watermarking technique for relational databases. In: International Conference on Information Systems Security, Springer (2009) 252–264
8. Rani, S., Kachhap, P., Halder, R.: Data-flow analysis-based approach of database watermarking. In: Advanced Computing and Systems for Security. Springer (2016) 153–171

9. Agrawal, R., Haas, P.J., Kiernan, J.: Watermarking relational data: framework, algorithms and analysis. The VLDB journal **12**(2) (2003) 157–169
10. Gupta, G., Pieprzyk, J.: Database relation watermarking resilient against secondary watermarking attacks. In: International Conference on Information Systems Security, Springer (2009) 222–236
11. Zhou, X., Huang, M., Peng, Z.: An additive-attack-proof watermarking mechanism for databases' copyrights protection using image. In: Proceedings of the 2007 ACM symposium on Applied computing, ACM (2007) 254–258
12. Halder, R., Cortesi, A.: A persistent public watermarking of relational databases. In: ICISS, Springer (2010) 216–230
13. Li, Y., Guo, H., Jajodia, S.: Tamper detection and localization for categorical data using fragile watermarks. In: Proceedings of the 4th ACM workshop on Digital rights management, ACM (2004) 73–82
14. Li, Y., Deng, R.H.: Publicly verifiable ownership protection for relational databases. In: Proceedings of the 2006 ACM Symposium on Information, computer and communications security, ACM (2006) 78–89
15. Bhattacharya, S., Cortesi, A.: Distortion-free authentication watermarking. In: International Conference on Software and Data Technologies, Springer (2010) 205–219
16. Zhang, Y., Niu, X., Zhao, D., Li, J., Liu, S.: Relational databases watermark technique based on content characteristic. In: Innovative Computing, Information and Control, 2006. ICICIC'06. First International Conference on. Volume 3., IEEE (2006) 677–680
17. Guo, H., Li, Y., Liu, A., Jajodia, S.: A fragile watermarking scheme for detecting malicious modifications of database relations. Information Sciences **176**(10) (2006) 1350–1378
18. Pournaghshband, V.: A new watermarking approach for relational data. In: Proceedings of the 46th annual southeast regional conference on XX, ACM (2008) 127–131
19. Kamran, M., Suhail, S., Farooq, M.: A robust, distortion minimizing technique for watermarking relational databases using once-for-all usability constraints. IEEE Transactions on Knowledge and Data Engineering **25**(12) (2013) 2694–2707
20. Bhattacharya, S., Cortesi, A.: A distortion free watermark framework for relational databases. In: ICSOFT (2). (2009) 229–234
21. Khan, A., Husain, S.A.: A fragile zero watermarking scheme to detect and characterize malicious modifications in database relations. The Scientific World Journal **2013** (2013)
22. Camara, L., Li, J., Li, R., Xie, W.: Distortion-free watermarking approach for relational database integrity checking. Mathematical problems in engineering **2014** (2014)
23. El-Bakry, H., Hamada, M.: A developed watermark technique for distributed database security. Computational Intelligence in Security for Information Systems 2010 (2010) 173–180
24. Razdan, R.: Real-time, distributed, transactional, hybrid watermarking method to provide trace-ability and copyright protection of digital content in peer-to-peer networks (March 7 2001) US Patent App. 09/799,509.
25. Mignotte, M.: How to share a secret. In: Workshop on Cryptography, Springer (1982) 371–375
26. Khanna, S., Zane, F.: Watermarking maps: hiding information in structured data. In: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2000) 596–605

27. Xie, M.R., Wu, C.C., Shen, J.J., Hwang, M.S.: A survey of data distortion watermarking relational databases. IJ Network Security **18**(6) (2016) 1022–1033
28. Khanduja, V., Chakraverty, S., Verma, O.P., Singh, N.: A scheme for robust biometric watermarking in web databases for ownership proof with identification. In: International Conference on Active Media Technology, Springer (2014) 212–225
29. Khanduja, V., Verma, O.P., Chakraverty, S.: Watermarking relational databases using bacterial foraging algorithm. Multimedia Tools and Applications **74**(3) (2015) 813–839
30. Kamel, I., AlaaEddin, M., Yaqub, W., Kamel, K.: Distortion-free fragile watermark for relational databases. International Journal of Big Data Intelligence **3**(3) (2016) 190–201
31. Alfagi, A.S., Manaf, A.A., Hamida, B., Olanrewajub, R.: A zero-distortion fragile watermarking scheme to detect and localize malicious modifications in textual database relations. Journal of Theoretical and Applied Information Technology **84**(3) (2016) 404
32. Rani, S., Koshley, D.K., Halder, R.: Adapting mapreduce for efficient watermarking of large relational dataset. In: Trustcom/BigDataSE/ICESS, 2017 IEEE, IEEE (2017) 729–736
33. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM **51**(1) (2008) 107–113
34. De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragments and loose associations: Respecting privacy in data publishing. Proceedings of the VLDB Endowment **3**(1-2) (2010) 1370–1381
35. Agrawal, S., Narasayya, V., Yang, B.: Integrating vertical and horizontal partitioning into automated physical database design. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ACM (2004) 359–370
36. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and reasoning about systems. Cambridge university press (2004)
37. Rodríguez, L., Li, X.: A dynamic vertical partitioning approach for distributed database system. In: Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on, IEEE (2011) 1853–1858
38. Shamir, A.: How to share a secret. Communications of the ACM **22**(11) (1979) 612–613
39. Iftene, S.: General secret sharing based on the chinese remainder theorem with applications in e-voting. Electronic Notes in Theoretical Computer Science **186** (2007) 67–84
40. Schneier, B.: Applied cryptography: protocols, algorithms, and source code in C. john wiley & sons (2007)