A Persistent Public Watermarking of Relational Databases

Raju Halder and Agostino Cortesi

Dipartimento di Informatica
Università Ca' Foscari di Venezia, Italy
{halder,cortesi}@unive.it
http://www.unive.it

Abstract. In this paper, we propose a novel fragile and robust persistent watermarking scheme for relational databases that embeds both private and public watermarks where the former allows the owner to prove his ownership, while the latter allows any end-user to verify the correctness and originality of the data in the database without loss of strength and security. The public watermarking is based on a part of the database state which remains invariant under processing of the queries associated with the database, whereas the private watermarking is based on an appropriate form of the original database state, called *abstract database*, and the *semantics-based properties* of the data which remain invariant under processing of the associated queries.

Key words: Watermarking, Databases, Abstraction

1 Introduction

Most of the existing watermarking techniques [1, 4, 10, 16] in the literature are private, meaning that they are based on some private parameters (*e.g.* a secret key). Only the authorized people (*e.g.* database owners) who know these private parameters are able to verify the watermark and prove their ownership of the database in case of any illegal redistribution, false ownership claim, theft etc. However, private watermarking techniques suffer from disclosure of the private parameters to dishonest people once the watermark is verified in presence of the public. With access to the private parameters, attackers can easily invalidate watermark detection either by removing watermarks from the protected data or by adding a false watermark to the non-watermarked data. In contrast, in public watermarking techniques [9, 15], any end-user can verify the embedded watermark as many times as necessary without having any prior knowledge about any of the private parameters to ensure that they are using correct (not tampered) data coming from the original source. For instance, when a customer uses sensitive information such as currency exchange rates or stock prices, it is very important for him to ensure that the data are correct and coming from the original source.

There are many applications that need to provide both private and public watermarks so that the owner can verify any suspicious database to claim his ownership, while at the same time any end-user can verify the originality and integrity of the data without exposing any private parameters. However, the existing techniques in the literature are unable to provide both.

Digital watermarking for integrity verification is called fragile watermarking as compared to the robust watermarking for copyright protection [11]. Fragileness of public watermarking must be maintained when any end-user wants to verify the correctness of the data through it. Since the location of public watermark in the host data is public, robustness of it is a prime concern too. However, there exist no watermarking scheme in the literature that can provide both of robustness and fragileness.

The watermark verification phase in the existing techniques [4, 10, 15, 16] completely relies on the content of the database. In other words, the success of the watermark detection is content dependent. *Benign Updates* or any other intensional processing of the database content may damage or distort the embedded watermark that results into an unsuccessful watermark detection. For instance, suppose a publisher is offering a 20% discount on the price of all articles. The modification of the price information may make the watermark detection phase almost infeasible if the price values are marked at bit-level or if any information (*viz*, hash value) is extracted based on this price information and used in the embedding phase. Therefore, most of the previous techniques are designed to face $Value\ Modification\ Attacks$, but are unable to resolve the persistency of the watermark under intentional allowed modifications.

In our previous work [6], we already introduced the notion of persistent watermark and discussed how to improve the existing techniques in terms of persistency of the watermark that serves as a way to recognize the integrity and ownership proof of the database bounded with a set of queries Q while allowing the evaluation of the database by queries in Q. In this paper, we go one step further, and we propose a novel fragile and robust persistent watermarking scheme that embeds both private and public watermarks where the former allows the owner to prove his ownership, while the latter allows any end-user to verify the correctness and originality of the data in the database without loss of strength and security. The public watermarking is based on a part of the database state which remains invariant under processing of the queries associated with the database, whereas the private watermarking is based on an appropriate form of the original database state, called *abstract database*, and the *semantics-based properties* of the data which remain invariant under the processing of the associated queries.

The structure of the paper is as follows: Section 2 recalls some basic concepts. In Section 3, we propose a combined persistent public and private watermarking scheme. In Section 4, we provide a brief discussions about the complexity of our algorithms and the relations with the existing techniques in the literature. Finally, we draw our conclusions in Section 5.

2 Basic Concepts

In this Section some basic concepts are recalled from the literature [5, 6, 8].

Persistent Watermark: Given a database dB and a set of applications interacting with the dB. Let Q be the set of queries issued by these applications. We denote the database model by a tuple $\langle dB, Q\rangle$. We do not make any restrictions on the operations used in Q (SELECT, UPDATE, DELETE, INSERT).

Let the initial state of dB be d_0. For the sake of simplicity, we assume that there is a unique sequence $d_1, d_2 \ldots, d_{n-1}$ of valid states of the dB reached when executing the queries of Q. Let W be the watermark that is embedded in state d_0. The watermark W is persistent *w.r.t.* Q if we can extract and verify it blindly from any of the following $n-1$ states successfully.

Definition 1 (Persistent Watermark).
Let $\langle dB, Q\rangle$ be a database model where Q is the set of queries associated with the database dB. Suppose the initial state of dB is d_0. The processing of the queries in Q over d_0 yield to a set of valid states $d_1, \ldots, d_{n-1}$. A watermark W embedded in state d_0 of dB is called persistent w.r.t. Q if

$$\forall i \in [1..(n-1)], \quad \mathit{verify}(d_0, W) = \mathit{verify}(d_i, W)$$

where $\mathit{verify}(d, W)$ is a boolean function such that the probability of "$\mathit{verify}(d, W) =$ true" is negligible if and only if W is not the watermark embedded in d.

Static versus Non-static Database States: Consider a database model $\langle dB, Q\rangle$ where Q is the set of queries associated with the database dB. For any state $d_i,\ i \in [0..(n-1)]$, we can partition the data cells in d_i into two parts *w.r.t.* Q: *Static* and *Non-Static*. Static part contains those data cells of d_i that are not affected by the queries in Q at all, whereas the data cells in non-static part of d_i may change under processing of the queries in Q.

Let $CELL_{d_i}$ be the set of cells in state d_i of dB. We denote the set of static cells of d_i *w.r.t.* Q by $STC^Q_{d_i} \subseteq CELL_{d_i}$. For each tuple $t \in d_i$ we denote the static part of it by $STC^Q_t \subseteq STC^Q_{d_i}$. Thus, $STC^Q_{d_i} = \bigcup_{t_j \in d_i} STC^Q_{t_j}$.

Now we discuss how to identify the static and non-static part of d_i *w.r.t.* Q. As SELECT and INSERT statements in Q do not affect the existing data cells of d_i, they do not take part in determining static/non-static part at all. However, DELETE statement may delete some data cells form static or non-static part, resulting into a subset of it. Thus, if $STC^Q_{d_i}$ and $(CELL_{d_i} - STC^Q_{d_i})$ represent static and non-static part of d_i *w.r.t.* Q respectively, a subset of it remains invariant over all the n valid states $d_0, d_1, \ldots, d_{n-1}$ under processing of DELETE statements in Q. The UPDATE statements modify values of the data cells in non-static part only. Let ATT^{update} be the set of attributes of dB that are targeted by the UPDATE statements. Thus we can identify the set of cells $STC^Q_{d_i},\ i \in [0..(n-1)]$ in state d_i corresponding to the attributes not in ATT^{update}, which remains invariant over all the n valid states.

Semantics-based Properties: Given a database state $d_i,\ i \in [0..(n-1)]$ of dB associated with a set of queries Q, we can identify some semantics-based properties of the data in d_i *w.r.t.* Q. These properties include $Intra-cell\ (IC)$, $Intra-tuple\ (IT)$ or $Intra-attribute\ among-tuples\ (IA)$ properties.

Intra-cell (IC) property: In this case individual data cells of a database state represents some specific properties of interests. Let the possible values of a cell corresponding to a attribute Z be $a \leq Z \leq b$ over all the valid states, where a and b represent integer values. The IC property can be represented by $[a, b]$ from the domain of intervals.

Intra-tuple (IT) property: An IT property is a property which is extracted based on inter-relationship between two or more attribute values in the same tuple. As an example, we may consider inter-relation between two attributes *basic_price* and *total_price* of a database containing commodity information where *total_price* includes *basic_price* plus a percentage of VAT of the *basic_price*. This can be abstracted by a relational abstract domain, like the domain of octagons [13].

Intra-attribute among-tuples (IA) property: The IA property is obtained from the set of independent tuples in a relation. Examples of such property are: (i) in an employee database $\#male\ employee = \#female\ employee \pm 1$, where $\#$ denotes cardinality of a set, (ii) the average salary of male employees is greater than the average salary of female employees, (iii) the total number of female employees is greater than 3, etc. The first two can be abstracted by relational abstract domain, whereas the last one can be represented by interval $[3, +\infty]$.

We denote the set of semantics-based properties obtained this way from state d_i *w.r.t.* Q by $P^Q_{d_i}$. For each tuple $t \in d_i$ we denote the set of IC, IT properties by $P^Q_t = IC^Q_t \cup IT^Q_t \subseteq P^Q_{d_i}$. Note that IA property can not be determined at tuple level. Thus, $P^Q_{d_i} = \{\bigcup_{t_j \in d_i}(IC^Q_{t_j} \cup IT^Q_{t_j})\} \cup IA^Q_{d_i}$, where $IA^Q_{d_i}$ represents $Intra-attribute\ among-tuples\ (IA)$ property in state d_i *w.r.t.* Q. Observe that $P^Q_{d_i}$ remains invariant over all the n valid states $d_0, d_1, \ldots, d_{n-1}$.

Abstract Database: In [5, 8], we proposed a sound approximation technique for database query languages based on the Abstract Interpretation framework where the values of the concrete database are replaced by abstract values from abstract domains representing some specific properties of interests, resulting into an abstract database. We may distinguish partially abstract databases in contrast to fully abstract one, as in the former case only a subset of data in the database is abstracted. The abstract database provides a partial view of the data by disclosing properties rather than their exact content. Consider the employee database in Table 1(a) that consists of a single table *emp*. Table 1(b) depicts a partially abstract database consisting of $emp^\sharp$ which is obtained by abstracting basic and gross salaries of the employees in *emp* by elements from the domain of intervals.

Table 1: Concrete and corresponding partially abstract employee database

(a) The concrete table *emp*

eID	Name	Basic Sal (euro)	Gross Sal (euro)	Age	DNo
E001	Bob	1000	1900	48	2
E002	Alice	900	1685	29	1
E003	Matteo	1200	2270	58	2
E004	Tom	600	1190	30	2
E005	Marry	1350	2542.5	55	1

(b) The abstract table $emp^\sharp$

$eID^\sharp$	$Name^\sharp$	Basic $Sal^\sharp$ (euro)	Gross $Sal^\sharp$ (euro)	$Age^\sharp$	$DNo^\sharp$
E001	Bob	[1000, 1300]	[1900, 2470]	48	2
E002	Alice	[900, 1170]	[1685, 2190.5]	29	1
E003	Matteo	[1200, 1560]	[2270, 2951]	58	2
E004	Tom	[600, 780]	[1190, 1547]	30	2
E005	Marry	[1350, 1755]	[2542.5, 3305.25]	55	1

Definition 2 (Abstract Database). *Let dB be a database. The database $dB^\sharp = \alpha(dB)$ where α is the abstraction function, is said to be an abstract version of dB if there exist a representation function γ, called concretization function such that for each tuple $\langle x_1, x_2, \ldots, x_n\rangle \in dB$ there exist a tuple $\langle y_1, y_2, \ldots, y_n\rangle \in dB^\sharp$ such that $\forall i \in [1 \ldots n],\ x_i \in \gamma(y_i) \vee x_i \in id(y_i)$.*

Watermarking based on partially abstract databases which are obtained by abstracting the data cells in non-static part ($CELL_d - STC_d^Q$) only, results into a content-independent persistent watermark. This is because although the exact values in ($CELL_d - STC_d^Q$) may change under processing of the queries in Q, their properties represented by abstract values remain invariant.

3 Persistent public/private watermarking

In the rest of the paper, we do not restrict ourself to any particular data type of the attributes. Attributes of any type including numeric, boolean, character, or any other can play roles in the public as well as private watermarking phase. Consider a database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$ in state d associated with a set of queries Q, where PK is the primary key. We divide the attribute set $\{A_0, A_1, A_2, \ldots, A_{\beta-1}\}$ into two parts *w.r.t.* Q: *Static attribute set* $A^Q_{static} = \{A^s_0, A^s_1, \ldots, A^s_{p-1}\}$ and *Non-static attribute set* $A^Q_{var} = \{A^v_0, A^v_1, \ldots, A^v_{q-1}\}$, where $p + q = \beta$. The set of static data cells STC_d^Q corresponds to static attribute set A^Q_{static}, whereas the set of non-static data cells ($CELL_d - STC_d^Q$) corresponds to non-static attribute set A^Q_{var}. Although the primary key PK may be static in nature, we exclude it from the set A^Q_{static} and mention it separately in the rest of the paper. Of course, any change on the values of the primary key will be detected in the verification phase.

Public watermark is embedded into a known location of the host data with known methods to guarantee its public detectability. We identify most significant

bit (MSB) positions of the data cells in STC_d^Q as the location for public watermark. We avoid non-static data cells because their values keep changing under processing of the queries in Q. This ensures the persistency of the public watermark. Since the public watermark in the host data is visible to all end-users, it is highly possible that attackers try to remove or distort it. We achieve robustness of the public watermark by choosing only the most significant bit positions of the host data as the location for public watermark: any major malicious change of the static portion of the database will be detected in the verification phase. Moreover, our scheme is designed to be fragile by using a cryptographic hash value of each tuple so as to detect and locate any modification when attackers try to modify the data in the database while keeping the watermark untouched.

The **private watermarking** is based on two invariants of the database states: *semantics-based properties* and *partially abstract database*, so as to maintain the persistency of the watermark under processing of the queries associated with the database. The security of private watermarking relies on the secret key as well as the level of abstraction used. Attackers do not know which properties are used to abstract the database. In addition, private watermarking is also based on MSBs of the attribute values. We assume the secret key to be large enough to thwart Brute force attack.

It is worthwhile to mention that, unlike existing techniques [1, 16, 9, 2], the verification phase of the proposed scheme is deterministic. Since the watermarking does not introduce any distortion to the underlying data, it is distortion-free. However, in our scheme we do not allow any schema transformations.

3.1 Public Watermarking

The overall architecture of the public watermarking phase is depicted in Figure 1. It consists of a single procedure, called **GenPublicKey**. The inputs of **GenPublicKey** are the database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$ in state d associated with a set of queries Q, the signature S of the database owner which is known to all end-users, and a parameter ξ representing the number of most significant bits (MSBs) available in attributes. The procedure generates a table $B(PK, b_0, \ldots, b_{p-1})$ where PK is the primary key, p is the number of attributes in A_{static}^Q and $\forall j \in [0..(p-1)]$: b_j contains either 1 or 0. The binary table B is treated as public key and made available to all end-users. Later, when any end-user wants to verify the source of a suspicious database, he uses B as the public key to generate and verify the embedded signature S.

The algorithm of **GenPublicKey** is depicted in Figure 2. Let us describe it in details.

Let $|S|$ be the length of the signature S in binary form. We divide S into m blocks $\{S_0, S_1, \ldots, S_{m-1}\}$ each of length p, where p is the number of attributes in A_{static}^Q and $m = \lceil \frac{|S|}{p} \rceil$. If the length of the last block is less than p, we append 0s to make it of length p.

For each tuple $t \in d$, the algorithm generates an hash value h in binary form of length p from its primary key and its static part $STC_t^Q = \{t.A_0^s, \ldots, t.A_{p-1}^s\}$.

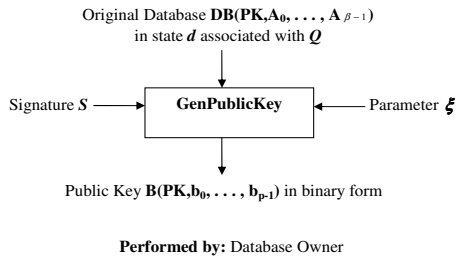

Fig. 1: Overall architecture of Public Watermarking Phase

We exclude the dynamic part of the tuples in computing hash because it keeps changing under processing of the queries. While computing hash, we assume that it is almost infeasible to generate same hash value from two different messages.

The $HASH$ function we might use takes a parameter p and generates a binary hash value of length p: we can use Merkle-Damgård's Meta method [12] where the length of the initial hash value and the length of each block of the binary string obtained from "$t.PK||t.A_0^s||\ldots||t.A_{p-1}^s$" (where $||$ stands for concatenation operation) is considered to be p.

Algo:	**GenPublicKey**
Input:	Database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$ in state d associated with a set of queries Q, Owner's signature S, Parameter ξ representing the no. of MSBs available in attributes.
Output:	A publicly available binary table $B(PK, b_0, \ldots, b_{p-1})$.
1.	Identify $A_{static}^Q = \{A_0^s, A_1^s, \ldots, A_{p-1}^s\}$
2.	Compute $m = \lceil \frac{\lvert S\rvert}{p} \rceil$, where $\lvert S\rvert$ denotes length of signature S in binary form and p=no. of attributes in A_{static}^Q
3.	Split the signature S into m blocks $\{S_0, S_2, \ldots, S_{m-1}\}$ where $\lvert S_i\rvert = p$
4.	FOR each tuples $t \in d$ DO
5.	$h = HASH(t.PK\|\|t.A_0^s\|\|\ldots\|\|t.A_{p-1}^s,\ p)$
6.	$i = PRSG(t.PK)\%m$
7.	$w = h \otimes S_i$
8.	Generate a binary tuple r in $B(PK, b_0, \ldots, b_{p-1})$ with $r.PK = t.PK$
9.	FOR $j = 0 \ldots p-1$ DO
10.	$k = HASH(t.PK\|\|t.A_j^s)\%\xi$
11.	$r.b_j = k^{th}$ MSB of $t.A_j^s \otimes w[j]$
12.	END FOR
13.	END FOR
14.	Return B

Fig. 2: Algorithm for Signature Embedding and Public Key Generation

Using pseudorandom sequence generator $PRSG$ (*e.g.* Linear Feedback Shift Register [7]) seeded by tuple's primary key, we identify which group the tuple

belongs to. If the tuple t belongs to i^{th} group, we compute $w = h \otimes S_i$ where h is the binary hash value of length p and S_i is the i^{th} block of the binary signature S. In other words, we embed i^{th} block S_i of signature S into all tuples that belong to i^{th} group. This ensures the existence of the signature during verification phase if there exist at least one marked tuple in each group after processing of DELETE operations. Observe that w is of length p.

Corresponding to tuple t, we now create a binary tuple r in $B(PK, b_0, \ldots, b_{p-1})$ whose primary key is same as that of t, *i.e.* $r.PK = t.PK$. For each static attribute $A_j^s \in A_{static}^Q$ where $j = 0, \ldots, (p-1)$, we obtain a MSB bit position k in the corresponding data cell $t.A_j^s$ by computing $k = HASH(t.PK||t.A_j^s)\%\xi$ where ξ is the number of MSBs available in A_j^s. The value of the j^{th} attribute b_j of r is, thus, r.b_j= k^{th} MSB of $t.A_j^s \otimes w[j]$.

We perform similar operations for all tuples in state d of dB, and finally we get a binary table $B(PK, b_0, \ldots, b_{p-1})$ consisting of a set of binary tuples generated this way. This binary table B is then made publicly available and treated as public key which is later used by any end-user to verify the embedded signature S.

Signature Verification Figure 3 depicts the overall architecture of signature verification phase performed by end-users. The procedure **PublicVerify** takes a suspicious database $dB(PK, A_0, \ldots, A_{\beta-1})$ in a different state d' as input, and generates an intermediate binary table $B'(PK, a_0, \ldots, a_{p-1})$. Based on this intermediate binary table $B'(PK, a_0, \ldots, a_{p-1})$ and the public key $B(PK, b_0, \ldots, b_{p-1})$ which is generated by the database owner in watermarking phase, the procedure **ExtractSig** extracts a signature S'. Finally, **MatchSig** compares S' with the original signature S. If it matches, the verification claim is true, otherwise false.

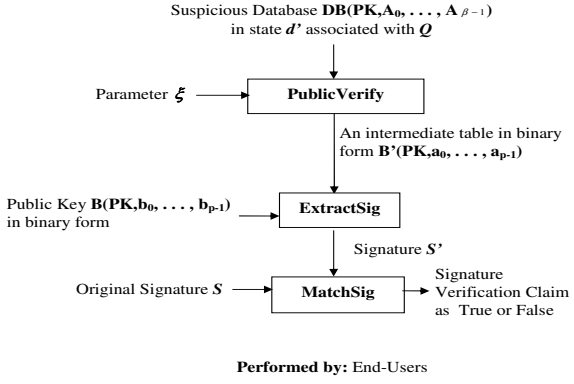

Fig. 3: Overall architecture of publicly Signature Verification phase

The algorithms of the procedures **PublicVerify** and **ExtractSig** are depicted in Figure 4 and 5 respectively. For each tuple $t' \in d'$, the algorithm

PublicVerify generates a binary tuple r' in $B'(PK,\ a_1,\ \ldots,\ a_{p-1})$ whose primary key is equal to the primary key of t', *i.e.* $r'.PK = t'.PK$. The binary values of the attributes $a_j,\ j \in [0..(p-1)]$ in r' are obtained as follows: (i) Compute binary hash value h of length p from the primary key $t'.PK$ and static part $STC^Q_{t'} = \{t'.A^s_0,\ \ldots,\ t'.A^s_{p-1}\}$ in similar way as in algorithm **GenPublicKey**, (ii) Extract k^{th} MSB from $t'.A^s_j$ in similar way as in algorithm **GenPublicKey**, (iii) Compute $a_j = k^{th}$ MSB of $t'.A^s_j \otimes h[j]$, where $h[j]$ represents j^{th} bit of h. In this way, the algorithm generates a set of binary tuples from the tuples in state d', and collection of these binary tuples forms the table B'.

Algo:	**PublicVerify**
Input:	Database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$ in state d' associated with Q, Parameter ξ, Public key $B(PK, b_0, \ldots, b_{p-1})$, Owner's Signature S.
Output:	Signature Verification Claim as True or False.
1.	Identify $A^Q_{static} = \{A^s_0, A^s_1, \ldots, A^s_{p-1}\}$
2.	FOR each tuples $t' \in d'$ Do
3.	$h = HASH(t'.PK \| \| t'.A^s_0 \| \| \ldots \| \| t'.A^s_{p-1}, p)$
4.	Construct a tuple r' in $B'(PK, a_0, \ldots, a_{p-1})$ such that $r'.PK = t'.PK$
5.	FOR $j = 0 \ldots p-1$ DO
6.	$k = HASH(t'.PK,\ t'.A^s_j)\ \%\ \xi$
7.	$r'.a_j = k^{th}$ MSB in $t'.A^s_j \otimes h[j]$
8.	END FOR
9.	END FOR
10.	S'=**ExtractSig**(B, B')
11.	Return **MatchSig**(S, S')

Fig. 4: Algorithm to Extract and verify Signature

Procedure **PublicVerify** then calls another procedure **ExtractSig**, and passes the binary table B' and the public key B (generated by the owner in watermarking phase). **ExtractSig** finds the pairs of tuples $(r,\ r')$ where $r \in B$ and $r' \in B'$ such that their primary keys are same *i.e.* $r.PK = r'.PK$. It then performs attribute-wise XOR *i.e.* $r.b_j \otimes r'.a_j$ for all $j \in [0..(p-1)]$, excluding the primary key attribute, and concatenate them to obtain a binary string *str*. If the tuple r and r' belongs to i^{th} group which is determined from the pseudo random sequence generator $PRSG$ seeded by $r.PK$ or $r'.PK$, the corresponding *str* denotes i^{th} block S'_i of a signature S'. This way we can collect all strings *str* from the tuples belonging to the i^{th} group and put them into the buffer buff[i]. If no tampering occurred, all strings in buff[i] will be same and represent S'_i. However, when data is tampered, some strings *str* in buff[i] may be different from the others. In such case, function $MajorityVote()$ returns the string with maximum match. In this way, we can determine $S'_0, \ldots, S'_{m-1}$ by extracting *str* from the tuples belonging to m different groups. By concatenating them, finally we get a signature S'. The procedure **MatchSig** returns true when S' matches with the original signature S, otherwise it returns False.

Algo:	**ExtractSig**
Input:	Public key $B(PK, b_0, \ldots, b_{p-1})$ and Binary table $B'(PK, a_0, \ldots, a_{p-1})$
Output:	Signature S'.
1.	Find binary tuple $r \in B$ and $r' \in B'$ such that $r.PK == r'.PK$
2.	FOR all pair (r, r') DO
3.	$str = NULL$
4.	For $j = 0 \ldots p-1$ DO
5.	Perform $str = str \| \| r.b_j \otimes r'.a_j$
6.	$i = PRSG(r.PK)\%m$
7.	buff[i] $\leftarrow str$
8.	END FOR
9.	FOR $i = 0 \ldots m-1$ DO
10.	S'_i=$MajorityVote$(buff[i])
11.	END FOR
12.	Return $S' = S'_0 \| \| S'_1 \| \| \ldots \| \| S'_i \| \| \ldots \| \| S'_{m-1}$

Fig. 5: Algorithm to Extract Signature

Example 1. Consider the employee database of Table 1(a) where eID is the primary key. Suppose the set of queries Q associated with the database are only able to increase the basic and gross salary of employees by at most 30%. As only the basic and gross salary can possibly be modified by the queries, we get A^Q_{static} = {*Name*, *Age*, *Dno*} and A^Q_{var}={*Basic Sal*, *Gross Sal*}.

Let the signature of the database owner be S = "RAJU" which is public and known to all end-users. By concatenating the ASCII codes of the characters in S, we get the binary representation of S as 01010010010000010100101001010101. Since $|S| = 32$ and the number of static attributes in A^Q_{static} is p=3, we divide S into $m = \lceil \frac{|S|}{p} \rceil = \lceil \frac{32}{3} \rceil = 11$ blocks each of length 3, *i.e.* 010 100 100 100 000 101 001 010 010 101 010. Since the last block contains only 2 bit, we append a 0 to make it of length 3.

Consider the tuple $t = \langle E001, Bob, 1000, 1900, 48, 2\rangle$. The primary key of t is $t.eID = E001$ and the static part of t is $STC^Q_t = \langle t.Name, t.Age, t.Dno\rangle = \langle Bob, 48, 2\rangle$. By following Step 5 of the algorithm **GenPublicKey**, let the hypothetical binary hash value of length $p = 3$ be $h = HASH(E001||Bob||48||2, 3) = 001$, obtained from its primary key $t.eID$ and its static part STC^Q_t. Based on the random value generated from $PRSG$ seeded by $t.eID$ (Step 6), suppose we determine that t belongs to the second group *i.e.* $i = 2$. Therefore, we compute $w = h \otimes S_2 = 001 \otimes 100 = 101$ in Step 7.

In Step 8, corresponding to t we create a binary tuple r in $B(PK, b_0, b_1, b_2)$ with $r.PK = t.eID = E001$. Suppose in Step 10, for each of the three static attribute values $t.Name$ = "*Bob*", $t.Age$ = "48" and $t.Dno$ = "2" we get the value of k as 2, 3 and 1 respectively (assuming ξ equal to 4). Let 0, 1 and 1 be the 2^{nd} MSB of "*Bob*", the 3^{rd} MSB of "48" and the 1^{st} MSB of "2" respectively. Therefore in Step 11, we compute $r.b_0 = 0 \otimes 1 = 1$, $r.b_1 = 1 \otimes 0 = 1$ and $r.b_2 = 1 \otimes 1 = 0$. This way we get the binary tuple $r = \langle E001, 1, 1, 0\rangle$. We do the same for other tuples in state d, and finally we obtain a binary table B which is then made publicly available.

Now we illustrate the verification phase. Consider the tuple t'=$\langle E001,\ Bob,\ 1000,\ 1900,\ 48,\ 2\rangle$. In Step 3 of the algorithm **PublicVerify**, we compute a binary hash value h of length $p = 3$ in similar way, and we obtain $h = 001$. We now construct a binary tuple r' in $B'(PK, a_0, a_1, a_2)$ as follows: (i) $r'.PK = t'.eID = E001$, (ii) for attribute values "Bob", "48" and "2", we get MSB position k as 2, 3 and 1 respectively. Thus, a_0=$0 \otimes 0 = 0$, a_1=$1 \otimes 0 = 1$ and a_2=$1 \otimes 1 = 0$, and the binary tuple r' in $B'(PK, a_0, a_1, a_2)$ is $\langle E001, 0, 1, 0\rangle$.

When we call the procedure **ExtractSig**, it finds two binary tuples $r = \langle E001,\ 1,\ 1,\ 0\rangle \in B$ and $r' = \langle E001,\ 0,\ 1,\ 0\rangle \in B'$, and it generates the string $str = 1 \otimes 0||1 \otimes 1||0 \otimes 0 = 100$. Since the tuples r and r' belong to the 2^{nd} group which is determined from the pseudorandom sequence generator seeded by $r.PK = E001$, we get that the string $str = 100$ represents the 2^{nd} block S'_2 of a signature S'. In similar way we can extract all 11 blocks $S'_0, \ldots, S'_{10}$ of S' from the tuples in d' belonging to 11 different groups, and by concatenating them we get 01010010010000010100101001010101O which is same as the original signature S = "RAJU".

3.2 Private Watermarking

The private watermarking algorithm **PrivateWatermark** is depicted in Figure 6. The inputs of the algorithm are the original database $dB(PK,\ A_0,\ A_1,\ \ldots,\ A_{\beta-1})$ in state d bounded with a set of queries Q, a secret key K, and the abstract function α. It generates a private binary watermark PW whose schema is $PW(PK,\ c_0,\ \ldots,\ c_{\beta-1},\ p_0,\ p_1,\ p_2)$.

The algorithm generates a partially abstract database state $d^\sharp$ from the original state d by abstracting the data cells belonging to the non-static part $(CELL_d - STC_d^Q)$ only. For each tuple $t^\sharp \in d^\sharp$, the algorithm generates a tuple r in $PW(PK, c_0, \ldots, c_{\beta-1}, p_0, p_1, p_2)$ whose primary key is equal to the primary key of $t^\sharp$ just to identify the tuples in PW uniquely and to perform matching in the verification phase. Note that as the primary key attribute is static in nature we never abstract its values. The algorithm, then, adds three values for the attributes p_0, p_1 and p_2 in r that correspond to the encoded values of IC, IT properties for $t^\sharp$ and encoded value of IA property for the whole database state $d^\sharp$, where g^p_{encode} represents an encoding function (*e.g. minimal perfect hash function*). G_i represents a pseudorandom sequence generator that returns i^{th} random value val when it is seeded by the attribute values of $t^\sharp$ including its primary key, and the secret key K. For all i from 0 to $\beta - 1$, val chooses an attribute randomly in $t^\sharp$ excluding the primary key and consider its MSB as the binary value for c_i in r. While computing the seed value for G_i or extracting MSBs, if there is any problem with abstract form of the values we can use its encoded form too. For instance, we can encode any interval by using the Chinese Remainder Theorem [14].

Observe that since the binary tuples in PW are constructed from semantics-based properties and partially abstract database information, the private watermark PW is invariant under processing of the queries in Q. The inputs of the verification algorithm are the database in state d' bounded with Q, the secret

Algo:	**PrivateWatermark**
Input:	Database $dB(PK, A_0, \ldots, A_{\beta-1})$ in state d bounded with a set of queries Q, Secret key K, Abstraction function α.
Output:	A private binary watermark $PW(PK, c_0, \ldots, c_{\beta-1}, p_0, p_1, p_2)$.
1.	Obtain Partially Abstract Database $dB^\sharp(PK^\sharp, A_0^\sharp, \ldots, A_{\beta-1}^\sharp)$ in state $d^\sharp$ by abstracting non-static part $(CELL_d - STC_d^Q)$ only
2.	Determine $IA_{d^\sharp}^Q$
3.	FOR each tuple $t^\sharp \in d^\sharp$ DO
4.	Construct tuple r in PW with primary key $r.PK = t^\sharp.PK^\sharp$
5.	Determine $IC_{t^\sharp}^Q$, $IT_{t^\sharp}^Q$
6.	$r.p_0 = g_{encode}^p(IC_{t^\sharp}^Q)$
7.	$r.p_1 = g_{encode}^p(IT_{t^\sharp}^Q)$
8.	$r.p_2 = g_{encode}^p(IA_{d^\sharp}^Q)$
9.	FOR (i=0; i< β; i=i+1) DO
10.	$val = G_i(K \circ t^\sharp.PK^\sharp \circ t^\sharp.A_0^\sharp \circ \ldots \circ t^\sharp.A_{\beta-1}^\sharp)$
11.	$j = val\%(no.\ of\ attributes\ in\ t^\sharp)$
12.	$r.c_i =$ (MSB of j^{th} attribute in $t^\sharp$)
13.	delete the j^{th} attribute from $t^\sharp$
14.	END FOR
15.	END FOR
16.	Return PW;

Fig. 6: Private Watermarking Algorithm

key K, the abstract function α, and the output is a binary table PW'. We use a boolean function $match(PW, PW')$ to compare PW' with the original private watermark PW which is obtained in the private watermarking phase. Note that the function $match(PW, PW')$ compares tuple by tuple taking into account the primary key of the tuples in PW and PW'. As tuples may be deleted from or added to the initial state d and yield to a different state d', only those tuples whose primary keys are common in both PW and PW' are compared. If $match(PW, PW') = True$, then the claim of the ownership is true, otherwise it is false. Observe that the verification phase is deterministic rather than probabilistic [9], as we compare and verify tuples in PW' against the tuples in PW with the same primary key only, and the binary values of the attributes in PW are invariant. Observe that there is an obvious tradeoff between the level of abstraction of the non-static part and the strength of the robustness of the private watermarking.

Example 2. Consider the database consisting of table *emp* with eID as the primary key in Table 1(a) where we determine that $A_{static}^Q = \{Name, Age, Dno\}$ and $A_{var}^Q = \{Basic\ Sal, Gross\ Sal\}$ *w.r.t.* the queries that are only able to increase the basic and gross salary of employees by at most 30%.

The partially abstract table $emp^\sharp$ is shown in Table 1(b) where data cells corresponding to the non-static attribute set A_{var}^Q are abstracted by elements from the domain of intervals. Consider an abstract tuple $t^\sharp$, say, $\langle E002, Alice,$

$[900, 1170]$, $[1685, 2190.5]$, 29, 1$\rangle$ in $emp^\sharp$. Corresponding to $t^\sharp$ we create a tuple r in watermark table $PW(PK, c_0, \ldots, c_{\beta-1}, p_0, p_1, p_2)$ with $r.PK = E002$.

In $t^\sharp$, the abstract values of the basic and gross salary are $[900, 1170]$ and $[1685, 2190.5]$ respectively. These abstract values represent IC properties for $t^\sharp$. The relation between two attributes $Basic\ Sal^\sharp$ and $Gross\ Sal^\sharp$ can be represented, for instance, by the following inequation: $Gross\ Sal^\sharp \geq \frac{(165 \times Basic\ Sal^\sharp)}{100} + 200$, assuming that $Gross\ Sal^\sharp$ includes $Basic\ Sal^\sharp$, 65% of the $Basic\ Sal^\sharp$ as PF, HRA etc and minimum of 200 euro as incentive. Thus, the IT property can be obtained by abstracting the above relation by the elements from the domain of polyhedra [3] *i.e.* by the linear equation just mentioned. The IA property may be: "The number of employees in every department is more than 2". This can also be represented by $[3, +\infty]$ in the domain of intervals. Suppose after encoding these three properties, we obtain the encoded values k_1, k_2, k_3. Therefore, the values of the attributes p_0, p_1, p_2 in r will be k_1, k_2, k_3 respectively.

Suppose the random selection of the attributes in $t^\sharp$ based on the random value generated by the pseudorandom sequence generator yields to the selection order as follows: $\langle[1685, 2190.5]$, 1, 29, $[900, 1170]$, *Alice*$\rangle$. We choose MSB from these attribute values in this order. Note that for abstract values (represented by intervals) we may extract MSB from its encoded values obtained by using Chinese Remainder Theorem. Let the extracted MSBs be 0, 1, 1, 0, 1 respectively. Thus the tuple r in PW would be $\langle E002$, 0, 1, 1, 0, 1, k_1, k_2, $k_3\rangle$.

After performing similar operations for all the tuples, the watermark PW is generated.

4 Discussions

The time complexity to generate the public key B depends only on the number of tuples in the original database linearly, whereas the time complexity to generate the private watermark PW depends on the the number of tuples in the original database as well as the complexity of the abstraction operation used in private watermarking phase. That is, the time complexity of the algorithms **GenPublicKey** and **PrivateWatermark** are $O(\eta)$ and $O(\eta \times \mu)$ respectively, where η is the number of tuples in the original database and μ is the complexity of the abstraction operation applied to tuples' values.

Given a database $dB(PK, A_0, A_1, A_2, \ldots, A_{\beta-1})$, the number of attributes in public watermark B is $p + 1$, where p is the cardinality of A^Q_{static}. Suppose η is the number of tuples in the original database state. The total number of cells in public watermark B is, thus, $(p + 1) \times \eta$. If σ is the number of bits required to represent the primary key, the total number of bits in B is $(\sigma + p) \times \eta$. Thus, the space complexity can be represented by $O(\eta)$.

Similarly we can show that the total number of bits in the private watermark PW is $(\nu + \beta) \times \eta$ where ν is the total number of bits required to represent the primary key and the three semantics-based properties p_0, p_1, p_2, and η is the number of tuples in the original database state. Thus, in this case also the space complexity can be represented by $O(\eta)$.

Before concluding, let us briefly discuss the properties of our proposal and relate them with the existing techniques in the literature.

Our proposed public and private watermarking scheme has the following properties: (*i*) It is blind, (*ii*) It does not introduce any distortions to the underlying data, and thus never degrades the usability of the data in the database, (*iii*) It preserves the persistency of both public and private watermarks, (*iv*) Public watermarking is robust as well as fragile, (*v*) There is no need of recomputation when tuples are updated by the queries associated with the database. (*vi*) The verification phase is deterministic rather than probabilistic and can, thus, reduce false positive and false negative.

Although the public watermarking algorithm of [9] is robust, it is not fragile: attackers can easily tamper the data by keeping the MSBs unchanged. Observe that our scheme uses cryptographic hash value obtained from the static part of each tuple. Any modification of the static part, thus, reflects to the hash value and makes the signature extraction from that tuple unsuccessful. In other words, any modification is narrowed down to each tuple.

The watermark embedding phase in [4, 10, 15, 16] is content-dependent. Any intentional processing of the database content may damage or distort the existing watermark, resulting the persistency of it into a risk. Our scheme is designed in such a way to preserve the persistency of the watermark by exploiting invariants of the database state.

The watermark detection algorithm of [1, 2, 9, 16] is parameterized with a threshold value. The lower the value of the threshold, the higher is the probability of a successful verification. We strictly improve on these techniques by exploiting invariants of the database state and by keeping the identity of the binary tuples in public key B and in private watermark PW. This makes the verification phase in both cases deterministic.

5 Conclusions

In this paper, we proposed a novel persistent watermarking scheme that embeds both private and public watermarks. Public watermarking is based on static data cells, whereas private watermarking is based on partially abstract database and semantics-based properties of the data. This ensures the persistency of both watermarks under processing of the queries associated with the database. We use cryptographic hash function and most significant bit positions for the location of public watermark to defeat any malicious attempt by the attackers.

Acknowledgement Work partially supported by Italian MIUR COFIN'07 project "SOFT" and by RAS project TESLA - Tecniche di enforcement per la sicurezza dei linguaggi e delle applicazioni.

References

1. Agrawal, R. and Haas, P. J. and Kiernan, J.: Watermarking relational data: framework, algorithms and analysis. The VLDB Journal 12(2), 157–169 (2003)

2. Bhattacharya, S., Cortesi, A.: A generic distortion free watermarking technique for relational databases. In: Proceedings of the 5th International Conference on Information Systems Security. pp. 252–264. Springer LNCS 5905, Kolkata, India (2009)
3. Chen, L., Miné, A., Cousot, P.: A sound floating-point polyhedra abstract domain. In: Proceedings of the 6th Asian Symposium on Programming Languages and Systems (APLAS '08). pp. 3–18. Springer-Verlag LNCS, Bangalore, India (2008)
4. Guo, H., Li, Y., Liua, A., Jajodia, S.: A fragile watermarking scheme for detecting malicious modifications of database relations. Information Sciences 176, 1350–1378 (2006)
5. Halder, R., Cortesi, A.: Abstract interpretation for sound approximation of database query languages. In: Proceedings of the IEEE 7th International Conference on INFOrmatics and Systems (INFOS2010), Advances in Data Engineering and Management Track. pp. 53–59. IEEE Catalog Number: IEEE CFP1006J-CDR, Cairo, Egypt (28–30 March 2010)
6. Halder, R., Cortesi, A.: Persistent watermarking of relational databases. In: Proceedings of the IEEE International Conference on Advances in Communication, Network, and Computing (CNC'10). IEEE CS, Calicut, India (4–5 Oct 2010)
7. Halder, R., Dasgupta, P., Naskar, S., Sarma, S.S.: An internet-based ip protection scheme for circuit designs using linear feedback shift register (lfsr)-based locking. In: Proceedings of the 22nd ACM/IEEE Annual Symposium on Integrated Circuits and System Design(SBCCI'09). ACM Press, Natal, Brazil (31st Aug–3rd Sep 2009)
8. Halder, R., Cortesi, A.: Observation-based fine grained access control for relational databases. In: Proceedings of the 5th International Conference on Software and Data Technologies (ICSOFT10). pp. 254–265. INSTICC, Athens, Greece (22–24 July 2010)
9. Li, Y., Deng, R.H.: Publicly verifiable ownership protection for relational databases. In: Proceedings of the ACM Symposium on Information, computer and communications security (ASIACCS '06). pp. 78–89. ACM, Taiwan (2006)
10. 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 (DRM '04). pp. 73–82. ACM, Washington DC (2004)
11. Lin, E., Delp, E.: A review of fragile image watermarks. In: Proceedings of the Multimedia and Security Workshop (ACM Multimedia '99). pp. 25–29. Orlando (1999)
12. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. CRC Press, Inc., Boca Raton, FL, USA (1996)
13. Miné, A.: The octagon abstract domain. Higher Order Symbol. Comput. 19(1), 31–100 (2006)
14. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation. pp. 169–180. Academic Press, New York (1978)
15. Tsai, M.H., Tseng, H.Y., Lai, C.Y.: A database watermarking technique for temper detection. In: Proceedings of the 2006 Joint Conference on Information Sciences (JCIS 2006). Atlantis Press, Kaohsiung, Taiwan, ROC (October 8-11 2006)
16. Zhang, Y., Niu, X., Zhao, D., Li, J., Liu, S.: Relational databases watermark technique based on content characteristic. In: First International Conference on Innovative Computing, Information and Control (ICICIC '06). pp. 677–680. IEEE CS, Beijing, China (16 October 2006)