Data-flow Analysis-based Approach of Database Watermarking

Sapana Rani, Preeti Kachhap, and Raju Halder

Indian Institute of Technology Patna, India
{sapana.pcs13, preeti.cs10, halder}@iitp.ac.in

Abstract. In this paper, we propose a persistent watermarking technique of information systems supported by relational databases at the back-end. The persistency is achieved by identifying an invariant part of the database which remains unchanged *w.r.t.* the operations in the associated applications. To achieve this, we apply static data-flow analysis technique to the applications. The watermark is then embedded into the invariant part of the database, leading to a persistent watermark. We also watermark the associated applications in the information system by using opaque predicates which are obtained from the variant part of the database.

Key words: Persistent Watermarking, Relational Databases, Data-flow Analysis, Security

1 Introduction

Database watermarking of relational databases has received much attentions to the research community over the last decade when various application scenarios, *e.g.* database-as-a-service, data-mining technologies, online B2B interactions, etc. demand an effective way to protect database information from various fraudulent activities, like illegal redistribution, ownership claims, forgery, theft, etc [15, 26]. Figure 1 depicts a pictorial view of database watermarking techniques, where a watermark W is embedded into the original database using a private key K (known only to the owner) and later the verification process is performed on any suspicious database using the same private key K by extracting and comparing the embedded watermark (if present) with the original watermark information.

1.1 Related Works

Existing watermarking techniques are categorized into two: distortion-based and distortion-free. Distortion-based techniques [1, 11, 28, 27, 10, 25] introduce distortion to the underlying database data, hence usability is a prime concern while watermarking. Distortion should always be introduced in such a way that it is tolerable and does not destroy the usability of the data at all. Watermarking in [1] is performed by flipping bits in numerical values at some predetermined

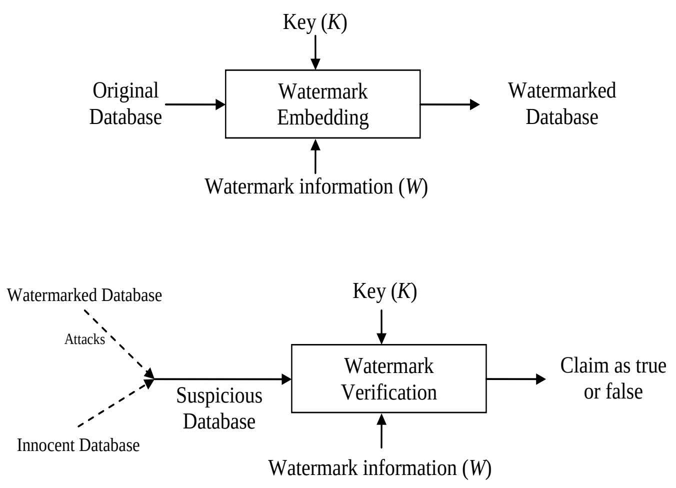

Fig. 1: Basic Watermarking Technique

positions based on the secret parameters. Image as watermark is embedded at bit-level in [28]. Approaches in [27, 10] are based on database-content: The characteristics of database-data is extracted and embedded as watermark into itself. Authors in [11] proposed a reversible-watermarking technique which allows to recover the original data from the distorted watermarked data. Khanduja et al. [19] proposed a secure embedding of blind and multi-bit watermarks using Bacterial Foraging Algorithm. Later, they used voice as biometric identifier for watermarking [18]. Unlike numerical values, categorical data type and non-numeric multi-word attributes are also considered as cover for watermarking in [25, 2]. Distortion-free watermarking techniques [21, 20, 13, 5, 6], on the other hand, do not introduce any distortion. Unlike distortion-based techniques, watermark is generated from the database rather than embedding. In [4, 21], hash value of the database is extracted as watermark information. Approaches in [20, 5, 6] are based on the conversion of database relation into a binary form to be used as watermark. In [17], watermark is generated based on digit frequency, length of data values, etc. in the database, whereas [7] generates the watermark based on the grouping of data into square matrix and the computation of determinant and diagonals' minor for each group. Although the approach [7] is not economically viable, but suitable to detect multifaceted attacks and is resilient against tuples insertion-deletion attack and value modification attack.

1.2 Motivations

This is to be observed that most of the distortion-based techniques in the literature use a part of the database content as cover [27, 10, 28], and therefore, a number of update or delete operations may distort the watermark or may make the watermark undetectable. Also re-watermarking the database is very expensive process. Authors in [13, 12] first address a key issue, called *persistency*, in the context of database watermarking where database tuples are being updated or deleted frequently by the associated legitimate applications. Their approaches aim at preserving persistency of the embedded watermark under usual database operations: watermark is embedded in an invariant part of the database (*w.r.t.* database operations), while the same is generated from the abstract variant part representing properties instead of actual values. However, they did not specify any approach to identify the variant/invariant part while watermarking a complete information system consisting of a set of applications interacting with a database at the back-end.

1.3 Contributions

In this paper, we propose a data-flow analysis-based approach which serves as a generic framework for persistent database watermarking. Unlike existing approaches, we consider watermarking of a complete information system which includes both the back-end database and the associated applications legitimately accessing or manipulating the data in the database. In particular, our proposal is unfolded into the following phases:

– Formulation of data-flow equations for the applications embedding query languages.
– Analysis of the applications based on the data-flow equations which effectively identifies an invariant part of the underlying database instances.
– Watermarking of the invariant part by distortion-based technique.
– Generation of Opaque Predicates from the variant part respecting the integrity constraints of the database systems.
– Embedding opaque predicates as watermarks into the associated applications.

The structure of the paper is as follows: Section 2 provides a motivating example. Section 3 recalls some basic notions about persistent watermarking, data-flow analysis, etc. The proposed technique is discussed in Section 4. In Section 5 and 6, we provide respectively a brief discussions on the complexity and robustness of our proposal. Experimental results are presented in Section 7. Finally, we draw our conclusions in Section 8.

2 Running Example

Consider three online trading companies, say x, y, z, who are maintaining their own databases and the associated applications. Fig 1 depicts one such database which stores the details of the customers, various products and the purchase history. Suppose, three companies have decided to collaborate, aiming at making the online purchasing system more attractive to the customers in terms of product availability.

However, according to the policy, each company can perform, in addition, its own business independently. A common interface after collaboration is developed and is allowed to access any of the three databases. This makes the database information vulnerable to various kinds of attacks, e.g. theft, illegal redistribution, ownership claiming, etc. Therefore, it is mandatory to watermark individual database in order to prevent above mentioned attacks.

Consider a code-fragment[1] P depicted in Figure 2 which accesses and manipulates database of Table 1. The code either inserts order details (statement 7-11) or offers gifts to the premium customers (statement 13-16). This is to be noted

```
0.  start;
1.  Statement stmt=DriverManager.getConnection(”jdbc:mysql://localhost:3306/demo”,”root”,”tiger”).
    createStatement();
2.  $choice = read();
3.  $Item = read();
4.  $Item_count = read();
5.  $Cust_id = read();
6.  if($choice == ”purchase”){
7.    $rs1 = SELECT ItemNo, UnitPrice FROM Store WHERE item=$Item and NoAvail>0;
8.    if($rs1.next()){
9.      $Ord_no = generate();
10.     INSERT INTO Order(OrderId, CustomerId, ItemNum, count, date, offer) VALUES ($Ord_no,
        $cust_id, $rs1.ItemNo, $Item_count, today(), NULL);
11.     UPDATE Customer SET TotalAmt = TotalAmt + $Item_count * $rs1.UnitPrice WHERE CustId
        = $Cust_id;}}
12. if($choice == ”offer”){
13.   $rs2 = SELECT CustId FROM Cust WHERE TotalAmt>5000;
14.   while($rs2.next()){
15.     $gift = read();
16.     UPDATE Order SET offer = $gift WHERE CustomerId = $rs2.CustId;}}
17. stop;
```

Fig. 2: Program *P*

that the database part corresponding to the attributes “TotalAmt” and “Offer” can possibly be updated by the application - hence it is a variant part. The rest of the database acts as invariant part. This is immediate that any watermark embedded into this variant part may get destroyed or undetectable due to the legitimate update operations on the values.

[1] Observe that we do not follow any specific language syntax.

Table 1: Online Trading Database

(a) Table "cust"

CustId	CustName	Address	Age	TotalAmt
1001CI01	rachel	London	22	2000
1001CI02	albert	New York	25	7000
1001CI03	john	Japan	27	4500

(b) Table "Store"

ItemNo	ItemName	NoAvail	UnitPrice
TN01	Notebook	23	200
TN02	calculator	25	1000

(c) Table "Order"

OrderId	CustomerId	ItemNum	Count	Date	Offer
111OI01	1001CI02	TN02	2	2-12-2012	NIL
111OI02	1001CI01	TN02	1	4-1-2013	NIL

In the subsequent sections, we propose an efficient way to identify invariant and variant part of the underlying databases *w.r.t.* the associated applications in the system. This will enhance the existing watermarking techniques *w.r.t.* the persistency issue.

3 Basic Concepts

In this Section, we recall some basic notion about persistent watermarking from [13].

Persistent watermark Given a database `dB` and a set of associate applications `A`, we denote by ⟨`dB`, `A`⟩ an information system model. Let d_0 be the initial state in which the watermark `W` is embedded. When applications from `A` are processed on d_0, the state changes and goes through a number of valid states $d_1, d_2 \dots, d_{n-1}$. The watermark `W` is persistent if we can extract and verify it blindly from any of the following $n-1$ states successfully.

Definition 1. *(Persistent Watermark)*
Let ⟨`dB`, `A`⟩ be an information system model where `A` represents the set of associated applications interacting with the database `dB`. Suppose the initial state of `dB` is d_0. The processing of applications from `A` over d_0 yields to a set of valid states $d_1, \dots, d_{n-1}$. A watermark `W` embedded in state d_0 of `dB` is called persistent if

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

where `verify`(d, `W`) is a boolean function such that the probability of "`verify`(d, `W`)=true" is negligible when `W` is not the watermark embedded in d.

Variant versus Invariant Database Part Consider an information system ⟨`dB`, `A`⟩ where `A` is the set of applications interacting with database `dB`. For any state d_i, $i \in [0..(n-1)]$, we can partition the data cells in d_i into two parts: Invariant and Variant. Invariant part contains those data cells that are not updated or deleted by the applications in $\mathtt{A}$, whereas data cells in variant part of d_i may change under the processing of applications in $\mathtt{A}$.

Let $\mathtt{CELL}_{d_i}$ be the set of cells in the state d_i. The set of invariant cells of d_i *w.r.t.* $\mathtt{A}$ is denoted by $\mathtt{Inv}^{\mathtt{A}}_{d_i} \subseteq \mathtt{CELL}_{d_i}$. For each tuple $t \in d_i$, the invariant part of t is $\mathtt{Inv}^{\mathtt{A}}_{t} \subseteq \mathtt{Inv}^{\mathtt{A}}_{d_i}$. Thus, $\mathtt{Inv}^{\mathtt{A}}_{d_i} = \bigcup_{t_j \in d_i} \mathtt{Inv}^{\mathtt{A}}_{t}$. The variant part *w.r.t.* $\mathtt{A}$, on the other hand, is defined as $\mathtt{Var}^{\mathtt{A}}_{d_i} = \mathtt{CELL}_{d_i}$ - $\mathtt{Inv}^{\mathtt{A}}_{d_i}$.

Data-flow Analysis Data-flow analysis is a technique for gathering information about the dynamic behavior of programs by only examining the static code [24]. A program's control flow graph (CFG) is used to define data-flow equations for each of the nodes in the graph. Data-flow analysis can be performed either in a forward direction or in a backward direction, depending on the equations defined. The least fix-point solution of the equations provides the required information about the program. The information gathered is often used by compilers when optimizing a program. A canonical example of a data-flow analysis is reaching definitions.

4 Proposed Technique

The intuition of our proposal is to make the embedded watermark persistent *w.r.t.* all possible operations in the information system. As database states change frequently under various legitimate operations in the associated applications, the content dependent watermarks embedded into the database are highly susceptible to benign updates. In particular, update and delete operations may remove or distort any existing watermark of the database [27, 10, 28].

In order to make the watermark persistent, our proposal aims at identifying some invariant parts of the database states which remain unchanged *w.r.t.* the applications. To this aim, we apply static data-flow analysis technique to the associated applications which identifies various parts of the database, called variant parts, targeted by update or delete operations in the applications. The complement of this variant part in the database acts as invariant part and is used for persistent watermarking. For instance, any database part retrieved by SQL select statement remains unchanged and is, of course, suitable for persistent watermarking. We also watermark the associated applications in the information system by using opaque predicates obtained from the variant part.

Summarizing, the proposed technique consists of the following phases:

- Identifying variant and invariant parts of the database, by performing data-flow analysis to the associated applications.
- Watermarking of invariant database parts.
- Watermarking of associated applications by using opaque predicates obtained from the variant part.

Assignment node n.

$$[\![n]\!] = (\texttt{JOIN}(n) \setminus \{(x, ?)\}) \cup \{(x, n)\}$$

Conditional node n.

$$[\![n]\!] = \texttt{JOIN}(n)|_b$$

UPDATE node n.

$$\begin{aligned}[\![n]\!] &= \texttt{JOIN}(n) \cup \{(\vec{v_d}|_\phi, n)\} \\ &= \texttt{JOIN}(n) \cup \{(a_1|_\phi, n), (a_2|_\phi, n), \ldots, (a_r|_\phi, n)\}\end{aligned}$$

DELETE node n.

$$\begin{aligned}[\![n]\!] &= \texttt{JOIN}(n) \cup \{(\vec{v_d}|_\phi, n)\} \\ &= \texttt{JOIN}(n) \cup \{(a_1|_\phi, n), (a_2|_\phi, n), \ldots, (a_r|_\phi, n)\}\end{aligned}$$

Other nodes.

$$[\![n]\!] = \texttt{JOIN}(n)$$

where $\texttt{JOIN}(n) = \bigcup_{w \in \texttt{pred}(n)} [\![w]\!]$.

Fig. 3: Data-flow Equations of applications embedding query languages

4.1 Data-flow Analysis

In this phase, we analyze the associated applications based on the data-flow equations in order to collect information about the part of the database information updated or deleted at each point of the applications.

The data-flow equations for various commands in the applications embedding query languages are defined in Figure 3. The abstract syntax of update and delete statements are denoted by $\langle \vec{v_d} \overset{upd}{=} \vec{e},\ \phi \rangle$ and $\langle del(\vec{v_d}),\ \phi \rangle$ respectively, where $\vec{v_d} = \langle a_1, a_2, \ldots, a_r \rangle$ denotes a sequence of database attributes, $\vec{e} = \langle e_1, e_2, \ldots, e_r \rangle$ denotes a sequence of arithmetic expressions, and ϕ denotes the `WHERE`-part of the statements following first-order formula [14]. We denote by notations $\texttt{upd}(\vec{v_d})|_\phi$ and $\texttt{del}(\vec{v_d})|_\phi$ the part of the database updated and deleted by $\langle \vec{v_d} \overset{upd}{=} \vec{e},\ \phi \rangle$ and $\langle del(\vec{v_d}),\ \phi \rangle$ respectively. Observe that any database-part is identified by a subset of attributes $\vec{v_d}$ values corresponding to a subset of tuples satisfied by ϕ. The notation (x, n) represents that x is defined at program point n, whereas $(x, ?)$ represents that x is defined by any program point. In case of conditional node with boolean expression b, we denote by notation $\texttt{JOIN}(n)|_b$ the information restricted by b.

The data-flow analysis is performed by using data-flow equations for each node of the control flow graph and solve them by repeatedly calculating the output from the input locally at each node until the whole system stabilizes, *i.e.* it reaches a fix-point. The least fix-point solution of the equations provides the information about the variant part of the database possibly updated or deleted by the program. Observe that during solving the data-flow equations, the result in any iteration may contain multiple definitions of the same attributes corresponding to different conditions (for example, say $\vec{v_d}|_{\phi_1}$ and $\vec{v_d}|_{\phi_2}$)[2]. In such case, we use `merge` function defined below:

$$\texttt{merge}\Big((a|_{\phi_1}, n_1), (a|_{\phi_2}, n_2)\Big) = \Big(a|_{\phi_1 \vee \phi_2}, \{n_1, n_2\}\Big)$$

This yields a modified data-flow equations for `UPDATE` and `DELETE` as follows:

UPDATE node n.

$$\begin{aligned} [\![n]\!] &= \texttt{merge}\Big(\texttt{JOIN}(n) \cup \{(\vec{v_d}|_{\phi}, n)\}\Big) \\ &= \texttt{merge}\Big(\texttt{JOIN}(n) \cup \{(a_1|_{\phi}, n), (a_2|_{\phi}, n), \dots, (a_r|_{\phi}, n)\}\Big) \end{aligned}$$

DELETE node n.

$$\begin{aligned} [\![n]\!] &= \texttt{merge}\Big(\texttt{JOIN}(n) \cup \{((\vec{v_d})|_{\phi}, n)\}\Big) \\ &= \texttt{merge}\Big(\texttt{JOIN}(n) \cup \{(a_1|_{\phi}, n), (a_2|_{\phi}, n), \dots, (a_r|_{\phi}, n)\}\Big) \end{aligned}$$

Lattice Structure Defining Data-flow. Let `Lab`, `Var`, ψ be the set of program points, the set of program variables and the set of well-formed formulas (in first order logic) respectively. Let $R = \texttt{Var} \times \psi \times \wp(\texttt{Lab})$. The Lattice is defined as $\Big(\wp(R), \subseteq, \emptyset, R, \cup, \cap\Big)$, where $\emptyset$ is the bottom element and R is the top element of the lattice. The lowest upper bound $\cup$ is defined as:

$$\{(x_i, \phi_i, \{l_{i,m}\})\} \cup \{(x_j, \phi_j, \{l_{j,n}\})\} = \begin{cases} \{(x_i, \phi_i \vee \phi_j, \{l_{i,m}\} \cup \{l_{j,n}\})\} & \text{if } x_i = x_j \\ \\ \{(x_i, \phi_i, \{l_{i,m}\})(x_j, \phi_j, \{l_{j,n}\})\} & \text{otherwise} \end{cases}$$

and the greatest lower bound $\cap$ is defined as:

$$\{(x_i, \phi_i, \{l_{i,m}\})\} \cap \{(x_j, \phi_j, \{l_{j,n}\})\} = \begin{cases} \{(x_i, \phi_i \wedge \phi_j, \{l_{i,m}\} \cap \{l_{j,n}\})\} & \text{if } x_i = x_j \\ \\ \emptyset \quad \text{otherwise} \end{cases}$$

Example 1. Let us illustrate the data-flow analysis on the running example `P` of section 2. The control-flow graph of `P` and the data-flow equations for each node are depicted in Figures 4 and 5 [3]. respectively. If we solve the equations assuming the initial value as empty set, we get the least fix-point solution depicted in Figure 6. The solution clearly indicates that the data corresponding to the attributes "TotalAmt" and "Offer" may possibly be defined at program points 11 and 16. Therefore, this part act as variant part of the database, while the remaining acts as an invariant part.

[2] By notation $\vec{v_d}|_{\phi}$ we denote the part of the database corresponding to the attributes $\vec{v_d}$ and tuples satisfying the condition ϕ.

[3] For the sake of simplicity, we omit set-curly-braces incase of singleton set

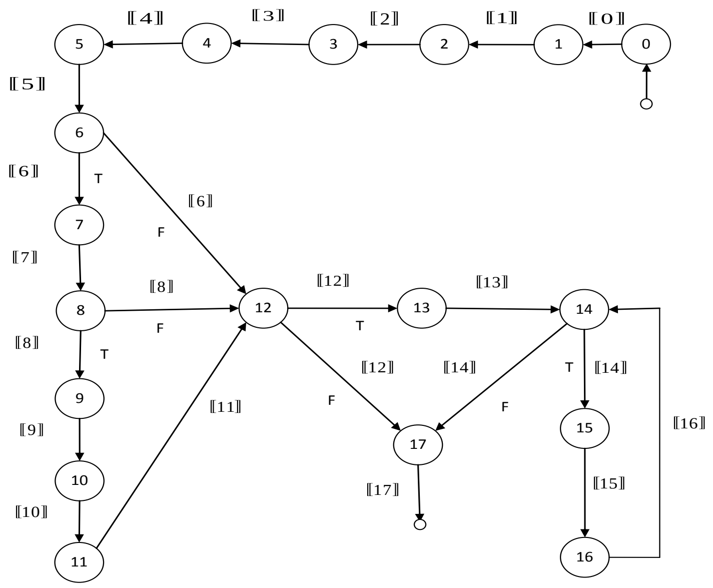

Fig. 4: Control-flow graph of P

4.2 Watermarking of invariant parts

In this phase, we may use any of the existing watermarking techniques [15] to watermark the invariant part of the database obtained in the previous phase. As invariant parts are not prone to modification, of course the embedded watermark will behave as persistent one.

However, the choice of existing watermarking technique is determined by (*i*) the use of data in a particular application context, (*ii*) the size of invariant part which is used as cover, (*iii*) the type of the cover, etc.

4.3 Watermarking of applications using opaque predicates

An opaque predicate is a predicate whose truth value is known a priori [8]. Moden et al. [22] first used opaque predicates in softwares watermarking by inserting dummy methods guarded by opaque predicates. The key challenge to design opaque predicates is that they should be resilient to various forms of attack-analysis. A variety of techniques such as using number theoretic results, pointer aliases, and concurrency have been suggested for the construction of opaque predicates [8]. In addition, Arboit also suggested a technique for constructing a family of opaque predicates through the use of quadratic residues [3]. Arboit's proposal is to encode the watermark information in the form of opaque predicates and to embed it into the software without effecting the control-flow structures.

The integrity constraints defined on a database ensure that the attributes under the constraints will have right and proper values in the database. More-

$$
\begin{aligned}
\llbracket 0 \rrbracket &= \{\} \\
\llbracket 1 \rrbracket &= (\llbracket 0 \rrbracket \backslash \{(\text{stmt}, ?)\}) \cup \{(\text{stmt}, 1)\} \\
\llbracket 2 \rrbracket &= (\llbracket 1 \rrbracket \backslash \{(\$\text{choice}, ?)\}) \cup \{(\$\text{choice}, 2)\} \\
\llbracket 3 \rrbracket &= (\llbracket 2 \rrbracket \backslash \{(\$\text{Item}, ?)\}) \cup \{(\$\text{Item}, 3)\} \\
\llbracket 4 \rrbracket &= (\llbracket 3 \rrbracket \backslash \{(\$\text{Item\_count}, ?)\}) \cup \{(\$\text{Item\_count}, 4)\} \\
\llbracket 5 \rrbracket &= (\llbracket 4 \rrbracket \backslash \{(\$\text{Cust\_id}, ?)\}) \cup \{(\$\text{Cust\_id}, 5)\} \\
\llbracket 6 \rrbracket &= \llbracket 5 \rrbracket|_{\$\text{choice}==\text{``purchase''}} \\
\llbracket 7 \rrbracket &= (\llbracket 6 \rrbracket \backslash \{(\$\text{rs1}, ?)\}) \cup \{(\$\text{rs1}, 7)\} \\
\llbracket 8 \rrbracket &= \llbracket 7 \rrbracket|_{\text{rs1.next()}} \\
\llbracket 9 \rrbracket &= (\llbracket 8 \rrbracket \backslash \{(\$\text{Ord\_no}, ?)\}) \cup \{(\$\text{Ord\_no}, 9)\} \\
\llbracket 10 \rrbracket &= \llbracket 9 \rrbracket \\
\llbracket 11 \rrbracket &= \llbracket 10 \rrbracket \cup \{\text{upd(TotalAmt)}|_{\text{WHERE CustId = \$Cust\_id}}\} \\
\llbracket 12 \rrbracket &= (\llbracket 6 \rrbracket \cup \llbracket 8 \rrbracket \cup \llbracket 11 \rrbracket)|_{\$\text{choice}==\text{offer}} \\
\llbracket 13 \rrbracket &= (\llbracket 12 \rrbracket \backslash \{(\$\text{rs2}, ?)\}) \cup \{(\$\text{rs2}, 13)\} \\
\llbracket 14 \rrbracket &= (\llbracket 13 \rrbracket \cup \llbracket 16 \rrbracket)|_{\text{rs2.next()}} \\
\llbracket 15 \rrbracket &= (\llbracket 14 \rrbracket \backslash \{(\$\text{gift}, ?)\}) \cup \{(\$\text{gift}, 15)\} \\
\llbracket 16 \rrbracket &= \llbracket 15 \rrbracket \cup \{\text{upd(offer)}|_{\text{WHERE CustomerId = \$rs2.CustId}}\} \\
\llbracket 17 \rrbracket &= \llbracket 12 \rrbracket \cup \llbracket 14 \rrbracket
\end{aligned}
$$

Fig. 5: Data-flow Equations of control-flow graph nodes of `P`

over, database designers also have opportunity to define their own assertions. These constraints which in fact define the properties of attribute-values, can be represented in terms of predicate formulas of first-order logic.

In this phase, we identify integrity constraints or we define assertions as a way to represent the properties of values in the variant part of the database obtained in the phase before. Observe that, although values in the variant part are prone to be updated or deleted, their properties represented by the constraints (integrity constarints or assertions) remain unchanged. Importantly, these constraints act as opaque predicate as their truth value *w.r.t.* the values in variant part is always true. We follow existing software watermarking techniques [16, 23] to watermark the applications in the information system by using these opaque predicates. As the applications contain SQL statements, we may use the conditional-part (`WHERE` clause) of SQL statements as cover.

Consider the running example. Consider an integrity constraint defined on the attribute "Age" which says that the age must belong to the range 15-70. This is expressed as:

$$15 <= \text{Age} <= 70$$

Since the formula is always true, it acts as an opaque predicate. Following Arboit's proposal [3], we can watermark the code by embedding this opaque predicate in the statement 13 as shown below:

```
$rs2 = SELECT CustId FROM Cust WHERE TotalAmt>5000 AND 15<=Age<=70;
```

5 Complexity Analysis

Let n be the program size. Let p be the number of variables (which include database attributes and application variables) in the program. The number of

⟦0⟧ = $\emptyset$
⟦1⟧ = {(stmt, 1)}
⟦2⟧ = {(stmt, 1), (\$choice, 2)}
⟦3⟧ = {(stmt, 1), (\$choice, 2), (\$Item, 3)}
⟦4⟧ = {(stmt, 1), (\$choice, 2), (\$Item, 3), (\$Item_count, 4)}
⟦5⟧ = {(stmt, 1), (\$choice, 2), (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5)}
⟦6⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5)}
⟦7⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)}
⟦8⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$}
⟦9⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9)}
⟦10⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9)}
⟦11⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9), (TotalAmt, 11)$\vert_{\text{CustId=\$Cust\_id}}$}
⟦12⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9), (TotalAmt, 11)$\vert_{\text{CustId=\$Cust\_id}}$}
⟦13⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9), (TotalAmt, 11)$\vert_{\text{CustId=\$Cust\_id}}$, (\$rs2, 13)}
⟦14⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9), (TotalAmt, 11)$\vert_{\text{CustId=\$Cust\_id}}$, (\$rs2, 13)$\vert_{\text{\$rs2.next()}}$, (\$gift, 15), (offer, 16)$\vert_{\text{CustomerId=\$rs2.CustId}}$}
⟦15⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9), (TotalAmt, 11)$\vert_{\text{CustId=\$Cust\_id}}$, (\$rs2, 13)$\vert_{\text{\$rs2.next()}}$, (\$gift, 15)}
⟦16⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9), (TotalAmt, 11)$\vert_{\text{CustId=\$Cust\_id}}$, (\$rs2, 13)$\vert_{\text{\$rs2.next()}}$, (\$gift, 15), (offer, 16)$\vert_{\text{CustomerId=\$rs2.CustId}}$}
⟦17⟧ = {(stmt, 1), (\$choice, 2)$\vert_{\text{\$choice==purchase}}$, (\$Item, 3), (\$Item_count, 4), (\$Cust_id, 5), (\$rs1, 7)$\vert_{\text{\$rs1.next()}}$, (\$Ord_no, 9), (TotalAmt, 11)$\vert_{\text{CustId=\$Cust\_id}}$, (\$rs2, 13)$\vert_{\text{\$rs2.next()}}$, (\$gift, 15), (offer, 16)$\vert_{\text{CustomerId=\$rs2.CustId}}$}

Fig. 6: Least fix-point solution of equations in Figure 5

data-flow equations associated with control-flow nodes of the program is n. Since each data-flow equation depends on the results of the predecessor nodes, the worst-case time complexity of each data-flow equation is $O(n)$. At each iteration the analysis provides us the information about the data defined up to each program point. Therefore, the height of the corresponding finite lattice is $O(p)$. Thus, the overall worst-case time complexity of data-flow analysis is $O(n \times n \times p) = O(n^2 p)$.

6 Security analysis

The proposed approach focuses on information systems scenario where databases are associated with a predefined set of applications. Our basic assumption is that only the database statements in the associated applications are authorized to perform computations on the database. Since attackers are not allowed to issue any other database operations, this mitigates the possibility of random value modification attacks on watermark in invariant part. This is to note that attacker can perform attacks in the variant part (see in section 7). The integrity constraints, which are treated as opaque predicates, also do not change over time. Therefore watermark detection in our approach is deterministic in practice. However, attackers may perform static analysis to detect opaque predicates [9] in order to remove watermarks from the associated applications codes.

Table 2: Descriptions of the notations

Count	no. of tuples used for particular experiment
ν	no. of attributes used for marking and detection in the relation
γ	fraction of tuples used in the experiment
ξ	no. of least significant bit available for marking in an attribute
TC	total count that is marked during embedding
α	significance level of the test for detecting watermark
τ	threshold parameter for detecting watermark

Table 3: Detection results after random update attacks in AHK algorithm [1].

Count	ν	γ	ξ	TC	embed time(msec)	ξ-updated	% tuples updated	α	match count	τ	detect time(msec)	Detect?
581012	10	10	15	58166	12058137	10	50	0.9	48432	52349	11874626	×
								0.8		46532	11632480	✓
							70	0.9	44632	52349	11761642	×
								0.75		43624	11494000	✓
			12	58166	11886111	10	50	0.9	53284	52349	11857304	✓
							70	0.9	51376	52349	11219001	×
								0.75		43624	11764361	✓
						8	50	0.9	51499	52349	11523694	×
								0.8		46532	11361032	✓
							70	0.9	49772	52349	11240122	×
								0.75		43624	12085957	✓
581012	10	20	15	28942	12040404	8	50	0.9	27013	26047	11990959	×
								0.8		23153	12058816	✓
								0.7		20259	12321103	✓
							70	0.9	26433	26047	12394587	×
								0.75		21706	11997753	✓
			12	28942	11350339	10	50	0.9	28942	26047	12926076	✓
							70	0.9	28942	26047	12198521	×
						8	50	0.9	24176	26047	12305839	×
								0.8		23153	11789865	✓
							70	0.9	22230	26047	11669565	×
								0.75		21706	11605415	✓
581012	10	50	15	11851	12358971	10	50	0.9	9867	10665	11519875	×
								0.8		9480	11358507	✓
							70	0.9	9066	10665	12006882	×
								0.75		8888	11794182	✓
			12	11851	11122883	10	50	0.9	10862	10665	11641533	✓
							70	0.9	10455	10665	11951827	×
								0.75		8888	11689275	✓
						8	50	0.9	10485	10665	11951827	×
								0.8		9480	11983700	✓
							70	0.9	10170	10665	12295857	×

Table 4: Detection after random update attacks on variant in proposed scheme

Invariant part						Variant part							
Count	ν	ξ	γ	TC	embed time(msec)	Count	ξ-updated	% tuples updated	α	match count	τ	detect time(msec)	Detect?
145253	10	12	5	29020	781581	435759	10	50	0.75	43664	65436	10276183	×
			10	14564	781281					21729	32701	9974971	×
			20	7272	781278					10736	16252	9842310	×
			50	2944	780517					4401	6680	9876861	×
290506	10	15	5	58017	2923709	290506	8	50	0.75	29247	43688	7226456	×
									0.5	29247	29125	7207290	✓
								90	0.5	29150	29125	7273410	✓
			10	29046	3128455			50	0.75	14567	21840	7153050	×
									0.5	14567	14560	7037200	✓
								90	0.5	14512	14560	7103890	×
			20	14436	3181587			50	0.75	7241	10879	7313342	×
									0.5	7241	7253	7100858	×
								90	0.5	7222	7253	7329082	×
			50	5889	2975901			25	0.9	2943	5365	7225908	×
								50	0.5	2944	2981	7100437	×
								90	0.5	2922	2981	7353554	×
435759	10	15	5	87202	6653587	145253	8	50	0.5	14572	14533	3993870	✓
								90		14553		4069217	✓
			10	43662	6556017			50	0.5	7269	7252	4017680	✓
								90		7249		3900057	×
			20	21701	6619247			50	0.5	3629	3620	4115240	✓
								90		3625		3938437	✓
			50	8866	6710785			50	0.5	1476	1492	3878820	×
								90		1471		3957758	×
522910	10	15	5	104641	9691596	58102	8	50	0.5	5901	5813	1612820	✓
								90		5893		1640135	✓
			10	52389	9862129			50	0.5	2921	2888	1713183	✓
								90		2926		1749594	✓
			20	26079	9689954			50	0.5	1453	1431	1700684	✓
								90		1451		1724654	✓
			50	10630	9511040			50	0.5	610	610	1711352	×
								90		611		1686176	✓

7 Experimental Results

We have performed experiment on the Forest Cover Type data set [4]. The data set has 581012 tuples and 61 attributes. An extra attribute *id* is added in our experiment that serves as primary key. The experiment is performed on server equipped with Intel Xeon processor, 64 GB RAM, 3.07 GHz clock speed and Linux operating system. The algorithms are implemented in java version 1.7 and MYSQL version 5.1.73.

In table 2, we describe the notations used in the tables showing experimental results. Table 3 depicts results of watermark detection after random update attacks take place in AHK algorithm [1]. Observe that detection may fail when more tuples are modified (updated) by attackers.

Experimental results obtained in our proposed scheme is depicted in Table 4. We have taken results by changing the size of invariant part as 25%, 50%, 75% and 90% that include 145253, 290506, 435759 and 522910 tuples respectively. Observed that we follow AHK algorithm to embed and detect watermark in invariant part. The experimental results depict that attackers may try to create a new watermark in variant part by performing random modification attacks. The results imply that probability of false watermark detection in variant part increases if the size of variant part decreases or the value of α (hence τ) decreases. For lower value of α, attacker may successfully prove the existence of such false-watermark. Parameters used by the attacker for detecting false-watermark are

[4] Available in the University Of California-Irvine KDD Archive kdd.ics.uci.edu/databases/covertype/covertype.html

similar as those used for marking by the owner. This situation may arise during proving the ownership in presence of all concerned people.

8 Conclusions

In this paper we proposed a persistent watermarking of information systems comprising of a set of applications supported by the database at the back-end. We provided a unified framework by combining software watermarking and database watermarking to watermark the complete system at a time. The proposal identifies both variant and invariant part of the database by applying data-flow analysis to the applications, aiming at making the embedded watermarks persistent. The proposed technique serves as generalized framework which may enhance any of the existing techniques in the literature in terms of persistency. We are now in process of building a prototype tool based on the proposal.

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. Al-Haj, A., Odeh, A.: Robust and blind watermarking of relational database systems. Journal of Computer Science 4, 1024–1029 (2008)
3. Arboit, G.: A method for watermarking java programs via opaque predicates. In: Proceedings of the 5th International Conference on Electronic Commerce Research (ICECR-5). pp. 184–196. ACM Press, San Diego, California, USA (2002)
4. Bhattacharya, S., Cortesi, A.: A distortion free watermark framework for relational databases. In: Proceedings of the 4th International Conference on Software and Data Technologies. Sofia, Bulgaria (July 26-29 2009)
5. Bhattacharya, S., Cortesi, A.: A generic distortion free watermarking technique for relational databases. In: Proceedings of the Fifth International Conference on Information Systems Security (ICISS 2009). LNCS Springer Verlag, Kolkata, India (December 2009)
6. Bhattacharya, S., Cortesi, A.: Distortion-free authentication watermarking. In: Cordeiro, J., Virvou, M., Shishkov, B. (eds.) Software and Data Technologies, pp. 205–219. Springer CCIS, Volume 170 (2013)
7. Camara, L., Li, J., Li, R., Xie, W.: Distortion-free watermarking approach for relational database integrity checking. Mathematical Problems in Engineering 2014 (2014)
8. Collberg, C., Thomborson, C., Low, D.: Manufacturing cheap, resilient, and stealthy opaque constructs. In: Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98). pp. 184–196. ACM Press, San Diego, California, USA (1998)
9. Dalla Preda, M., Madou, M., De Bosschere, K., Giacobazzi, R.: Opaque predicates detection by abstract interpretation. In: Johnson, M., Vene, V. (eds.) Algebraic Methodology and Software Technology, pp. 8–95. Springer LNCS 4019 (2006)
10. Guo, H., Li, Y., Liua, A., Jajodia, S.: A fragile watermarking scheme for detecting malicious modifications of database relations. Information Sciences 176, 1350–1378 (May 2006)

11. Gupta, G., Pieprzyk, J.: Database relation watermarking resilient against secondary watermarking attacks. In: Proceedings of the Fifth International Conference on Information Systems Security (ICISS 2009). pp. 222–236. LNCS Springer Verlag, Kolkata, India (December 2009)
12. 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, India (2010)
13. Halder, R., Cortesi, A.: A persistent public watermarking of relational databases. In: Proceedings of the 6th International Conference on Information Systems Security (ICISS '10). pp. 216–230. Springer LNCS 6503, India (2010)
14. Halder, R., Cortesi, A.: Abstract interpretation of database query languages. Computer Languages, Systems & Structures 38, 123–157 (2012)
15. Halder, R., Pal, S., Cortesi, A.: Watermarking techniques for relational databases: Survey, classification and comparison. Journal of Universal Computer Science 16(21), 3164–3190 (2010)
16. Hamilton, J., Danicic, S.: A survey of static software watermarking. In: 2011 World Congress on Internet Security (WorldCIS' 11). pp. 100–107. IEEE (2011)
17. 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)
18. Khanduja, V., Chakraverty, S., Verma, O.P., Singh, N.: A scheme for robust biometric watermarking in web databases for ownership proof with identification. In: Active Media Technology, pp. 212–225. Springer (2014)
19. Khanduja, V., Verma, O.P., Chakraverty, S.: Watermarking relational databases using bacterial foraging algorithm. Multimedia Tools and Applications pp. 1–27 (2013)
20. 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 (ASIACCS '06). pp. 78–89. ACM, Taipei, Taiwan (2006)
21. 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 Press, Washington DC, USA (2004)
22. Monden, A., Iida, H., Matsumoto, K.i., Inoue, K., Torii, K.: A practical method for watermarking java programs. In: Proceedings of the 24th Annual International Computer Software and Applications Conference, (COMPSAC 2000). pp. 191–197. IEEE (2000)
23. Myles, G., Collberg, C.: Software watermarking via opaque predicates: Implementation, analysis, and attacks 6(2), 155–171 (Apr 2006)
24. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer-Verlag New York, Inc. (1999)
25. Sion, R., Atallah, M., Prabhakar, S.: Rights protection for categorical data. IEEE transactions on knowledge and data engineering 17, 912–926 (July 2005)
26. Yingjiu, L.: Database Watermarking: A Systematic View. Springer Verlag (2007)
27. 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 2006). IEEE CS, Beijing, China (2006)
28. Zhou, X., Huang, M., Peng, Z.: An additive-attack-proof watermarking mechanism for databases' copyrights protection using image. In: SAC '07: Proceedings of the 2007 ACM symposium on Applied computing. pp. 254–258. Seoul, Korea (2007)