Watermill: An Optimized Fingerprinting System for Databases under Constraints

Julien Lafaye, David Gross-Amblard, Camelia Constantin, and Meryem Guerrouani

Abstract—This paper presents a watermarking/fingerprinting system for relational databases. It features a built-in declarative language to specify usability constraints that watermarked data sets must comply with. For a subset of these constraints, namely, weight-independent constraints, we propose a novel watermarking strategy that consists of translating them into an integer linear program. We show two watermarking strategies: an exhaustive one based on integer linear programming constraint solving and a scalable pairing heuristic. Fingerprinting applications, for which several distinct watermarks need to be computed, benefit from the reduced computation time of our method that precomputes the watermarks only once. Moreover, we show that our method enables practical collusion-secure fingerprinting since the precomputed watermarks are based on binary alterations located at exactly the same positions. The paper includes an in-depth analysis of false-hit and false-miss occurrence probabilities for the detection algorithm. Experiments performed on our open source software WATERMILL assess the watermark robustness against common attacks and show that our method outperforms the existing ones concerning the watermark embedding speed.

Index Terms—Fingerprinting, relational databases, linear optimization, query optimization.

1 INTRODUCTION

The growth of the Internet and the World Wide Web has been followed by a drastic increase of digital data exchange. Users are searching for valuable data, whose elaboration costs time and money. In this setting, data owners are exposed to malicious users who leech and disseminate unauthorized copies of their copyrighted works. Watermarking aims at strengthening ownership proofs by hiding copyright information (called watermarks) within digital documents. Watermarks have to be invisible, that is, they must not impact the document usefulness, and robust, that is, resilient against removal attempts. Watermarking techniques are widely studied in the area of multimedia documents (see [5] and [12] for a survey), and several recent efforts have lead to solutions for watermarking relational databases [2], [17], [22]. In these works, alterations are performed by selecting some purely numerical values in the database and by altering either the least significant bits (LSBs) or the distribution of these values. Altering the data might sound like a limitation for a legitimate purchaser, but it is now widely admitted that this is mandatory to achieve watermark robustness. The challenge is then to control the alterations so that the document only undergoes a minor loss of quality.

There are various ways to measure and control data quality. For multimedia data, this quality can be expressed by signal processing characteristics like the peak signal-to-noise ratio (PSNR). For numerical databases, the accuracy of each numerical value is commonly used. To preserve this accuracy, most existing database watermarking methods are parameterized with the maximum number of LSBs that can be altered [2], [17]. Even if the control of accuracy is a prerequisite for generic database watermarking, it might not be sufficient. Indeed, data sets are used not only as simple tables but also through views, that is, through the evaluation of natural queries. For instance, it may happen that even small accuracy-preserving alterations on the data have a strong impact on the results of specific SQL queries (for example, join queries) that are important for the database purchaser. Hence, specific tools for defining database usefulness according to the user’s need are required.

The quality of a data set, that is, what is important from the user’s perspective, can be expressed through usability constraints (also known as semantic integrity constraints in the database community [7], [24]). Before selling a watermarked database, the owner specifies his usability constraints. For instance, a business application may require that important local properties are preserved (like a maximum alteration of 1 Euro on a cost), as well as global properties (like the result of a natural join between products and sales tables or the mean of the cost of all products). A convenient way to specify such constraints is to use SQL. Part of these constraints can be public so that the purchaser is convinced of the quality of the watermarked database. However, part of them might also be kept secret so that the precise location of watermarked values is not revealed, hence enforcing invisibility. An important point is that the owner may refuse to sell its database to a purchaser that specifies too strong constraints that prevent the alterations (for example, a purchaser that requires no alteration at all). However, a
purchaser usually prefers controlled alterations rather than no data at all. The main problem then relies on finding robust watermarks that respect all usability constraints simultaneously.

A natural method to achieve this is the greedy search [21]. For each watermark bit, it consists of trying some alterations and discarding them if they lead to a usability constraint violation. This method is very expensive, as a great number of alterations have to be performed (which is a prerequisite for the watermark to be robust). Hence, there is a need for an optimized method of valid watermark discovery.

This need is also strengthened by a natural extension of watermarking: fingerprinting. In fingerprinting, databases with different embedded watermarks are distributed to members of a group of purchasers [25] sharing the same usability constraints. A classic application of fingerprinting is traitor tracing, that is, proving which purchaser is the source of an illegal diffusion. This extension is challenging since now, not only one but several acceptable watermarks are to be computed. In this case, the computational effort required by the greedy method would be tremendous. This method, which is already expensive for one consumer, has to be iterated several times for fingerprinting.

One may argue that speed is not an issue, since watermarking is done once and for all in the life cycle of a document. However, for some applications, speeding up the watermarking process is a real requirement. Indeed, several onerous data sets like meteorological measurements [1] have a huge commercial or scientific value but only during a short time window. For instance, a method that requires eight days of computation for watermarking a weekly weather forecast data set for 10 purchasers is ineffective. Since the delay between the production and the sale is short, anticipated computation (caching) cannot be used. For several real applications, watermarking time is an important issue and should be lowered.

Our contribution. In this paper, we present WATERMILL [14], an optimized watermarking and fingerprinting system for relational databases. It features a built-in usability constraint definition language, as well as an efficient watermarking engine to override the limitations of the greedy method. Our system achieves the following capabilities:

- **Speeding up the watermarking/fingerprinting process.** We identify a set of constraint patterns for which it is possible to translate the watermarking problem into an integer linear program (ILP). Valid watermarks are found among the solutions of this system. These patterns capture what we call weight-independent constraints, which include aggregate and join computations, which are central in the design of usability constraints used in real-world data sets [6]. To produce valid watermarks, we propose two approaches, both implemented in WATERMILL. The first one uses existing ILP solvers. The second one, the Pairing algorithm, searches for pairs of compensating alterations. It performs a precomputation to obtain a simple description of a huge set of valid watermarks. After precomputation, finding several valid watermarks is immediate (linear time). This second approach is far much faster and best suits huge databases (millions of tuples).
- **Resisting attacks from a collusion of purchasers while preserving usability.** When distributing several fingerprinted versions of a database, the owner is exposed to collusion attacks. In this setting, several malicious purchasers may collude to compare their watermarked versions. By locating the positions where their documents differ, they can discover where the alterations were performed. By modifying documents on these positions, they might obtain a new version without a readable watermark, hence evading detection. The design of efficient collusion-secure fingerprinting codes is a long-standing effort [3], [9], [23]. To achieve collusion security, watermark messages must be carefully chosen from a precise codebook and inserted in the same positions in all the distributed versions. We show that for the aforementioned weight-independent constraints, a family of good watermarks can be quickly found so that bit alterations are always performed at the same positions. Hence, using a collusion-secure code is possible while preserving usability constraints.

Outline of the paper. Section 2 recalls watermarking basics. Our declarative language for usability constraints is defined in Section 3. Further on, we study the optimization of watermark discovery for weight-independent constraints: by using pure integer linear programming in Section 4 and by focusing on a specific family of solutions in Section 5. Collusion-secure fingerprinting while preserving constraints is addressed in Section 6. A relevance and robustness analysis of watermark detection is presented in Section 7. These results are confirmed in Section 8 by experiments performed on our open source prototype WATERMILL [13], [14]. Related work is presented in Section 9.

## 2 DATABASES WATERMARKING

### 2.1 Example and Hypothesis

**Example.** The example database **mills** shown in Fig. 1 is used throughout the paper. It consists of a single relation, **mills**, containing descriptions of power plants around towns in France. Each power plant is characterized by its type, its coordinates (x, y), the place where it is located, its height, and its production. Note that the **place** attribute is the primary key of the relation **mills**. Fig. 2 shows two examples of fingerprinted **mills**, namely, **mills2** and **mills3**. They differ from the original data on several positions, for example, the **prod** and **height** of **place** Challans.

**mills instance** – **mills** relation

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>place</th>
<th>type</th>
<th>prod</th>
<th>height</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>53</td>
<td>Dinan</td>
<td>windmill</td>
<td>125</td>
<td>60</td>
</tr>
<tr>
<td>62</td>
<td>56</td>
<td>Challans</td>
<td>windmill</td>
<td>223</td>
<td>90</td>
</tr>
<tr>
<td>55</td>
<td>22</td>
<td>Colmar</td>
<td>mill</td>
<td>443</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>51</td>
<td>Chalain</td>
<td>mill</td>
<td>53</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>18</td>
<td>Dijon</td>
<td>geothermal</td>
<td>33</td>
<td>15</td>
</tr>
</tbody>
</table>

**Fig. 1.** The original **mills** instance.
watermark. Classic types of attack are: stolen data set (up to a realistic extent) in order to erase the adversarial setting kept identical to the purchased one. In a more realistic setting, primary keys have not been altered. When a suspect purchaser with illegal redistribution. We call our example databases, the height of the windmill of Challans. When the relation name is clear from the context, we use the notation \( a_p \) for value identifiers (for example, \( \text{height}_{\text{Challans}} \)). Clearly, if all relations possess a primary key, every numerical value is an identified one. We denote by \( \mathcal{I}(d) \) the set of all value identifiers of database \( d \). For example, \( \mathcal{I}(\text{mills}) \) contains all \( x, y, \text{height}, \text{prod} \) and primary identifiers for key values in \{Dinan, Challans, Colmar, Chalain, Dijon\}. Finally, the set of all identified values of the database \( d \) is seen as a vector \( \vec{v}(d) \) indexed by value identifiers from \( \mathcal{I}(d) \). On our example databases, the height of the windmill of Challans, denoted by \( v_{\text{prod}} \), is \( \vec{v}(\text{mills})[v_{\text{prod}}] = 90 \) on database \( \text{mills} \) and \( \vec{v}(\text{mills2})[v_{\text{prod}}] = 91 \) on database \( \text{mills2} \).

Watermarking method (starting point). We recall here Agrawal et al.’s method [2] for watermarking relational databases. It is used as a reference for the subsequent algorithms. Note also that a different method, from Sion et al. [21], can also be used, but we do not consider it here due to lack of space. The method relies on an integer pseudorandom generator \( S \) whose draws cannot be predicted without the knowledge of its seed. Several parameters are used: the secret key \( K \), the ratio \( 1/\gamma \) of watermarked elements, the maximum number \( \xi \) of alterable LSBs, and the maximum probability \( \delta \) of detection errors.\(^1\) The algorithm respects the following steps: For each value identifier \( i \), the random number generator is seeded with \( K \) concatenated with \( i \) (and \( i \) is seen as binary strings). If the first integer produced by \( S \) is 0 modulo \( \gamma \), then the value \( \vec{v}(d)[i] \) is considered for watermarking. In the binary expression of \( \vec{v}(d)[i] \), a position is chosen according to the next integer from \( S \), computed modulo \( \xi \). The bit located at this position in \( \vec{v}(d)[i] \) is replaced by a mark bit, whose value is given by the parity of the third production of \( S \). The detection algorithm proceeds identically by locating bit positions in a suspect data set. The binary values found at these positions are then compared with the expected ones, and the number of correct matches is recorded. If the match ratio exceeds a given threshold, the data set is declared suspect.

This method exhibits several important properties: robustness, accuracy, incremental, public system, and blindness (see [2] for a complete discussion). Among them, blindness means that the original data set is not needed for detection:

\(^1\) The number \( v \) of relational attributes available is also used, but our use of value identifiers already captures the attribute selection.
Only the watermarking parameters \((K, \gamma, \xi, \delta)\) and the suspect data set are required. Blindness is crucial in the context of very large data sets, since their backup, copy, or transmission to a trusted authority might not be possible.

In the sequel, we show how to add usability constraint preservation to this basic building block.

3 A Declarative Language for Usability Constraints

A watermarked version of a database is obtained by changing the values of some attributes. Consequently, the results of some queries on the watermarked data set are likely to be modified. The purchaser or the owner of the watermarked data set might want to limit the impact of these modifications, both on attribute values and on query results, by specifying some usability constraints. Similar to [8] and [21], we distinguish between the set \(L\) of local usability constraints, which affect tuples individually, and the set \(G\) of global usability constraints, which apply on a subset of tuples.

3.1 Constraints Examples

We introduce here the syntax of our declarative language through a list of short examples. The owner might want to enforce the quality of the watermarked data set by defining a set of local constraints \(L = \{C_1, C_2, C_3\}\), asking for a maximum alteration of 10 units on map coordinates \(x\) and \(y\) \((C_1)\), 1 m on the height of a windmill \((C_2)\), and 20 KW on a production measure \((C_3)\). Documents \(mills2\) and \(mills3\) in Fig. 2 respect each constraint in \(L\).

\[
\begin{align*}
\text{local 10 on mills.x, mills.y} & \quad \# C1 \\
\text{local 1 on (select height from mills)} & \quad \# C2 \\
\text{local 20 on mills.prod} & \quad \# C3 \\
\end{align*}
\]

A tuple value \(e\) is a modifiable value of a data set \(d\) if \(e\) is within the scope of at least one local constraint in \(L\). In this paper, alterations are done only on values that are both identified and modifiable. We denote by \(M(d)\) the set of these elements. By definition, \(M(d) \subseteq I(d)\). We now enrich our example with a set of global constraints \(G = \{C_4, C_5, C_6, C_7, C_8, C_9\}\). We say that a watermark is valid if it simultaneously respects \(L\) and \(G\). Constraint \(C_1\) allows a maximal variation of 10 for the total production of mills and windmills.

\[
\begin{align*}
\text{global 10 on (} & \quad \# C4 \\
\text{select sum(prod) from mills} & \quad \text{where type in (mill, windmill)} \\
\end{align*}
\]

Constraint \(C_5\) expresses that for legal reasons, production under 60 KW must remain under this limit after watermarking.

\[
\begin{align*}
\text{invariant (select place from mills} & \quad \# C5 \\
\text{where prod < 60)} \\
\end{align*}
\]

Document \(mills2\) respects global constraints \(C_4\) and \(C_5\), but document \(mills3\) does not (the overall production for windmills and mills increased by more than 10 units and the production for Chalain exceeds 60 KW). Constraint \(C_6\) imposes that the distance between the Dinan power plant and a power collector located in position \((10, 10)\) on the map must not be distorted by more than 5 units.

\[
\begin{align*}
\text{global 5 on (} & \quad \# C6 \\
\text{select sqrt((x - 10)^2 + (y - 10)^2) from mills where place=Colmar)} \\
\end{align*}
\]

Constraint \(C_7\) states that power plants with equal heights should still have equal heights after watermarking (but not necessarily with the same value). Note that this kind of pattern is a convenient way to express foreign key constraints between tables.

\[
\begin{align*}
\text{invariant (} & \quad \# C7 \\
\text{select ml.place, m2.place from mills ml, m2} & \quad \text{where ml.height = m2.height)} \\
\end{align*}
\]

When constraints do not fit the previous patterns, a call to an external checking program (named, for example, qualityChecker) can also be used.

\[
\begin{align*}
\text{call "qualityChecker"} & \quad \# C8 \\
\end{align*}
\]

Finally, arithmetic expressions can be used. Suppose that heights are expressed in feet for a windmill and in meter for a mill. Constraint \(C_8\) states that for administrative reasons, the sum of the heights of the power plants of Chalans and Colmar must not exceed a given limit, which is expressed in meters.

\[
\begin{align*}
\text{invariant (} & \quad \# C9 \\
\text{select 0.304 * p1.height + p2.height < 30} & \quad \text{from mills as p1, mills as p2} \\
\text{where p1.place=Chalans and p2.place=Colmar)} \\
\end{align*}
\]

3.2 Semantics

We now give the formal semantics of these constraints (this section can be skipped in the first reading). In the sequel, \(\varphi(d)\) denotes the result obtained by applying an SQL query \(\varphi\) on document \(d\). The identified part \(I(\varphi, d)\) is a subset of the set of identified values \(I(d)\). An identified value \((r, a, p) \in I(\varphi, d)\) if and only if its value is in \(\varphi(d)\) and can be traced through \(\varphi\). For instance, for the query \(\text{select type from mills where height > 80, } \varphi(d) = \{\text{windmill}\}\). Although two tuples are of type windmill, only the one identified by place Chalans participates to the result; hence, \(I(\varphi, d) = \{\text{mills.type, Chalans}\}\). Computing \(I(\varphi, d)\) is not always possible [4]. In what follows, we use only queries \(\varphi\) for which the computation of \(I(\varphi, d)\) is straightforward. It is the case for our previous sample queries. A conditional sentence \(\varphi\) is a conjunction/disjunction combination of terms \(R.A \theta c\), where \(R\) is a relation, \(A\) an attribute, \(\theta \in \{<, =, >, \geq, \leq\}\), and \(c\) a constant. A set of identifiers \(I\) is said to be independent of \(\varphi\) if no identifier in \(I\) has a relation name and an attribute name of one of the terms of \(\varphi\). The constraints are defined as follows:

- Constraint local \(p\) on \(\psi\), where \(p \in \mathbb{N}\) and \(\psi\) is a query identifying single attributes. For \(k\) local constraints with queries \(\psi_1, \ldots, \psi_k\), the set of modifiable values \(M(d)\) is the set of all identified elements of \(\psi_1, \ldots, \psi_k\), that is, \(M(d) = \bigcup I(\psi_j, d)\).
- Constraint global \(p\) on \(\psi\), where \(p \in \mathbb{N}\) and \(\psi\) is a query identifying attributes. For \(k\) global constraints with queries \(\psi_1, \ldots, \psi_k\), the set of modifiable values \(M(d)\) is the set of all identified elements of \(\psi_1, \ldots, \psi_k\), that is, \(M(d) = \bigcup I(\psi_j, d)\).
the usual syntax is obvious. For instance, $C3$ is expanded into local 20 on (select prod from mills). A watermarked data set $d_w$ is such that for all $i \in M(d)$, $\tilde{v}(d_w)[i] = \tilde{v}(d)[i] + \tilde{w}[i]$, that is, each value $\tilde{v}(d_w)[i]$ in the watermarked data set is the sum of the original value $\tilde{v}(d)[i]$ and an alteration $\tilde{w}[i]$. A data set $d_w$ is said to respect the local constraint if and only if $\forall i \in I(\psi, d)$, $|\tilde{w}[i]| \leq p$.

- **Constraint global** $p$ on $\psi$, where $p \in \mathbb{N}$ and $\psi$ is a query returning a numerical value. A data set $d_w$ is said to respect the global constraint if and only if $|\psi(d) - \psi(d_0)| \leq p$.

- **Constraint invariant** ($\psi$). A watermarked data set $d_w$ satisfies this constraint if $\psi(d) = \psi(d_0)$.

- **Constraint call** (program). This represents a call to an external program that checks constraints (for example, a computation not easily definable in SQL). This clause is respected by data set $d_w$ if the program answers “yes” with $d$ and $d_0$ as input. This corresponds to the usability plug-ins in [21].

Finding alterations of the data set that respect such constraints may be a difficult computational task. The next section shows how we optimize the discovery of such alterations for specific patterns of constraints.

## 4 Fingerprinting as an Optimization Problem: Integer Linear Program Reduction

### 4.1 Splitting Constraints

Most of the previous constraints can be translated into linear constraints, that is, inequations on the sum of values from the data set. Our approach is then to split the set $G$ of usability constraints into two sets: the set $Lin$ of linear constraints and the set of remaining constraints $Gen$ that cannot be translated (for example, call constraints). We will resolve usability constraints from $Lin$ using an ILP solver, obtaining a partial instantiation of a good watermark vector, say, $\tilde{w}_1$. Watermark positions left undefined are denoted by $M/\tilde{w}_1$. The remaining constraints from $Gen$ are explored using the greedy method GreedyMark [21] on positions $M/\tilde{w}_1$, obtaining a complete watermark vector $\tilde{w}$. The next sections present the automatic translation of constraints and our watermarking algorithm.

### 4.2 Translation into Linear Constraints

Recalling the previous example, the translation of constraint $C_9$ is immediate:

$$0.304 \tilde{w}\text{[height\_Challene]} + \tilde{w}\text{[height\_Colmar]} \leq 30. \ #C9$$

Constraints $C_1$ to $C_8$ can also be expressed by means of linear inequalities, for example,

$$-10 \leq \tilde{w}\text{[Exp\_man]} \leq 10. \ #C1$$

Constraints $C_6$ and $C_8$ cannot be linearized: There is no reason for $C_6$ to be linear, and $C_8$ is quadratic. Observe also that $C_7$ can be linearized (as explained in the sequel), but its conditions do not hold in the original data set. Based on the previous example, we identify a set of constraint patterns that can be directly translated into a linear program $P$. These patterns express useful usability constraints on the data and can be easily recognized. We consider four patterns: local constraints and weight-independent [invariant/join/sum] constraints. For each pattern, we give its general syntactic form, specific restrictions that must be checked, and its translation into a linear inequation:

1. **Local constraints** (for example, $C_1, C_2, \text{and } C_3$)
   - Pattern: local $p$ on $\psi$.
   - Restrictions: $\psi$ respects the following pattern:
     $$\text{select ... from ... where } \varphi,$$
     where $\varphi$ is an SQL condition independent of $M(d)$.
   - ILP constraint: $\forall i \in I(\psi, d), -p \leq \tilde{w}[i] \leq p$.

2. **Weight-independent sum constraints** (wis-constraints). A constraint is said to be weight independent if the set of value identifiers involved in the query computation is the same, whatever the perturbations on identified values are. For instance, $C_1$ is weight independent: Even if $\text{prod}$ is modified, the set of identified values in windmill or mill that are involved in the computation of $C_1$ does not change. This property allows us to compute once and for all the set of identified values used in a query computation and to assign variables to these values in the linear system. An example of a weight-independent constraint is given below:

$$\text{global } 10 \text{ on (select sum(prod) from mills where prod \leq 99)}.$$

If a $\text{prod}$ is equal to 99, watermarking it to 100 will exclude it of the previous linear encoding. A sufficient condition to obtain the weight-independence property is that the where clause is followed by an SQL condition independent of $M(d)$. The formal pattern is the following:

- Pattern: global $p$ on $\psi$.
- Restrictions: $\psi$ has the following pattern:
  $$\text{select sum(attName) from relName where } \varphi,$$
  where $\varphi$ is an SQL condition independent of $M(d)$.
- ILP constraint: $-p \leq \sum_{i \in I(\psi, d)} \tilde{w}[i] \leq p$, with $\psi' = \text{select attName from relName}$ where $\varphi$.

The weight-independent sum pattern can be easily extended to handle the mean aggregate. However, quadratic constraints, for example, on the standard deviation, cannot be expressed.

3. **Weight-independent invariant constraints** (for example, $C_5$)
   - Pattern: invariant($\psi$).
   - Restriction: $\psi$ has the following pattern:
     $$\text{select ... from ... where } \varphi \text{ and } A \theta c,$$
where $\varphi$ is an SQL condition independent of $M(d)$, $A$ is an attribute, $\theta \in \{=, <, >\}$, and $c \in \mathbb{N}$.

- ILP constraint: for all $i \in I(\psi_A, d)$
  \[ \bar{w}[i] + \bar{v}(d)[i] \leq c, \]
  with $\psi_A = \text{select } A \text{ from ... where } \varphi$.

4. Weight-independent join constraints (for example, $C_7$).
- Pattern: invariant$(\psi)$.
- Restriction: $\psi$ has the following pattern:
  \[ \text{select ... from ... where } \varphi \text{ and } A = B, \]
  where $\varphi$ is an SQL condition independent of $M(d)$, $A$, $B$ are attributes.
- ILP constraint: for any pair of value identifiers $(i, j) \in I(\psi_{AB}, d)$ with $\psi_{AB}$ defined by
  \[ \text{select } A, B \text{ from ... where } \varphi \text{ and } A = B, \]
  we add constraint $\bar{w}[i] = \bar{w}[j]$.

4.3 Algorithm

Clearly, any watermark satisfying the linear system respects all $Lin$ constraints. Our aim is then to extend Agrawal et al.’s algorithm [2] so that only good watermarks are selected. The sketch of the resulting algorithm is given as follows:

1. We compute the distortions using the method of Agrawal et al., and we memorize them in a vector $\bar{\Delta}[i]$ (the distortion on identified value $i$).
2. We create a new (0-1) variable $\bar{s}[i]$.
3. For each identified value $i$, we add the constraint $\bar{w}[i] = \bar{s}[i] - \bar{\Delta}[i]$ to the linear program obtained by translating the usability constraints.
4. The watermark is the solution of the above ILP $P$ that maximizes the number of values $\bar{s}[i]$ that are equal to 1.

The overall algorithm includes Li et al.’s extension of the initial Agrawal et al. algorithm [2] to fingerprinting [17] (see Algorithm 1). In order to hide the binary message $m$, at each watermarking step, a bit of $m$ is pseudorandomly chosen. This bit is masked by an exclusive OR with a pseudorandom bit. Symbol $S_t(k)$ denotes the output number $t$ of a pseudorandom generator initialized with the seed $k$ (see [2]). For the watermark detection, we use the values at positions $i$ such that $\bar{s}[i] = 1$ (see Algorithm 2) to recover the message. The value of each recovered bit is chosen using majority voting. This procedure is called Threshold-Majority (omitted due to space limitations) and returns the word formed by the bits whose vote values exceed a predefined threshold $1/2 + \alpha$ ($0 < \alpha < 1/2$). Note that some bits of the word might remain undefined, for example, when there are 50 percent of votes for both values 0 and 1.

Algorithm 1: LinearMark (dataset $d$, message $m$, local constraints $L$, linear constraints $Lin$, arbitrary constraints $Gen$, parameters $K, \xi, \gamma$)

- $P \leftarrow \emptyset$; /* empty linear program */
- $M(d) \leftarrow \text{ExtractModifiableIdentifiers}(d, L)$;
- foreach modifiable identifier $i \in M(d)$ do
  - if $(S_t(i \circ K) \mod \gamma = 0)$ then /* try mark this element */
    - $j \leftarrow S_t(i \circ K) \mod \xi$ /* bit index */
    - $k \leftarrow S_t(i \circ K) \mod m[l]$ /* letter index */
    - mask $\leftarrow S_t(i \circ K) \mod 2$;
    - mark $\leftarrow m[k] \oplus$ mask; /* mark bit */
    - value $\leftarrow \bar{v}[i]$;
    - mvalue $\leftarrow \bar{v}[i]$;
    - mvalue$[j] \leftarrow$ mark;
    - // compute distortion $\bar{\Delta}[i] \in \{-2^j, 0, 2^j\}$
    - $\bar{\Delta}[i] \leftarrow$ (value $- mvalue$);
    - Add linear constraints to $P$
      - $0 \leq \bar{s}[i] \leq 1$ and $\bar{s}[i] \in \mathbb{N}$;
      - $\bar{w}[i] \leftarrow \bar{s}[i] - \bar{\Delta}[i]$;
  - Add translation of constraints from $Lin$ to $P$
  - $\bar{w}[i] \leftarrow $LinearSolve ($P$, $m_0$); // $\bar{s}$ is the input
  - $\bar{w}[i] \leftarrow $GreedyMark ($m_1$, $M/d_1 \mathbb{N}, L$, $Gen$);
- return $\bar{w} + \bar{s}$;

Algorithm 2: LinearDetect(suspect dataset $d$, message length $l$, key $K, L$)

- for $k$ in $\{1, \ldots, l\}$ do
  - $\text{vote}[k][0] \leftarrow 0$
  - $\text{vote}[k][1] \leftarrow 0$
- foreach identifier $i$ in $d$ appearing in $\hat{s}$ do
  - $j \leftarrow S_t(i \circ K) \mod \xi$; /* bit index */
  - $k \leftarrow S_t(i \circ K) \mod l$; /* letter index */
  - mask $\leftarrow S_t(i \circ K) \mod 2$; /* mark bit */
  - readMark $\leftarrow \bar{s}[j] \oplus$ mask; /* read the mark */
  - $\text{vote}[k][\text{readMark}] \leftarrow \text{vote}[k][\text{readMark}] + 1$;
  // voting
- return ThresholdMajority($\text{vote}$);

4.4 Properties

Blindness. Our algorithm is blind in the same sense as in [21], that is, it does not require the original data set for detection. However, as explained in [21], the positions used for constraint-preserving watermarking must be recorded for future detection (in our case, the positions $i$ where $\bar{s}[i] = 1$). Indeed, the recomputation of the vector $\bar{s}$ on the watermarked data set as the solution of the linear program does not necessarily yield the same value as the one computed before watermarking. Having to backup, this set is not a real limitation since it can be efficiently compressed (for example, by simple interval encoding).

Robustness. Note that when no usability constraints are to be preserved, this algorithm yields exactly the same watermark as the one proposed by Agrawal et al. (that is, when all $\bar{s}[i]$ are equal to 1). Therefore, its robustness against attacks is the same. The robustness of the algorithm varies when usability constraints are taken into account. The more restrictive the constraints are, the easier it is to guess the location of watermarked bits. A too complex group of
constraints may even yield a nonwatermarkable data set. Nonetheless, experimental evaluations show that we are still able to find watermarks on practical constraints (see Section 8).

Problem reduction. State-of-the-art ILP solvers (like Ilog Cplex, Dash Xpress-Mp, IBM OSL, etc.) can handle classically up to $10^4$ variables. If the number of modifiable values exceeds this limit, which is likely to occur on large data sets, several methods can be used: apply standard reduction techniques to lower the number of useful variables [19], [26], work only on active identifiers, that is, those which are used in query evaluation, choose a random subset of variables, or group them according to a secret key. In what follows, we present a heuristic that allows the efficient computation of a subset of solutions of the ILP.

In this setting, identified productions correspond to primary keys Dinan, Challans, Colmar, and Chalain. The values associated with these primary keys are 125, 223, 443, and 53. Query $\psi_1$ has $125 + 223$ as a result; hence, it depends on prod from Dinan and Challans. Query $\psi_2$ has $125 + 223 + 443 + 53$ as a result, and it depends on prod from Dinan, Challans, Colmar, and Chalain. We represent this information in the following dependency matrix $A(\psi_1, \psi_2)$:

<table>
<thead>
<tr>
<th></th>
<th>Dinan</th>
<th>Challans</th>
<th>Colmar</th>
<th>Chalain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\psi_1$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\psi_2$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Let $i_1, \ldots, i_2l$ be the set of value identifiers. The aim of the pairing algorithm is to partition these values into $l$ dependency pairs $\{(i_1^1, i_1^2), \ldots, (i_l^1, i_l^2)\}$ so that for all $j \in \{1, \ldots, l\}$

- $i_1^j$ and $i_2^j$ are involved in the same constraints and
- the watermark distortions on $i_1^j$ and $i_2^j$ will be opposite.

Going back to our example, the productions of Dinan and Challans are involved in $\psi_1, \psi_2$. The productions of Colmar and Chalain participate only to $\psi_2$. Hence, the first pair will be (Dinan, Challans), and the second will be (Colmar, Chalain). For example, hiding message “10” could be achieved by applying the following alterations:

- +1 on the prod of Dinan and -1 on the prod of Challans for the first pair and
- -1 on the prod of Colmar and +1 on the prod of Chalain for the second pair.

Observe that the overall distortion on constraints $\psi_1$ and $\psi_2$ will always be zero.

Ensuring blindness. In order to produce a blind algorithm, the alteration on a pair $(i_1^1, i_2^1)$ will only depend on the primary key of $i_1$. According to this key and to the allowed local distortion, we secretly choose a bit position index. If the numerical values $\tilde{v}[i_1^1]$ and $\tilde{v}[i_2^1]$ are equal on this bit position, we cannot use the pair for watermarking. On the contrary, if they differ on this position, we can permute these bits without altering the usability constraints. Our method is the following:

- We choose a pseudorandom binary mask mask.
- To encode a “1,” we put (1 $\oplus$ mask) on $\tilde{v}[i_1^1][\text{index}]$ and (0 $\oplus$ mask) on $\tilde{v}[i_2^1][\text{index}]$.
- To encode a “0,” we put (0 $\oplus$ mask) on $\tilde{v}[i_1^1][\text{index}]$ and (1 $\oplus$ mask) on $\tilde{v}[i_2^1][\text{index}]$.

Since these bits were different in the original data set, this operation does not change their sum, and the contribution of this pair to the global distortion is still zero. The complete algorithm includes the precomputation phase (see Algorithm 4) and the actual watermarking process (see Algorithm 3). Note that similar to the previous query-preserving algorithms [21], the set of positions used for watermarking must be recorded for further detection. Anyway, this set allows for an efficient compact representation.
Algorithm 3: PairMark(document d, message m, wis- 
constraints \(C, L, K, \xi, \gamma\))

\[ \text{pairs} = \text{ComputePairs}(d, L, K); \]
// pairs of equal classes, with a 
pseudo-random order
// This computation is done only once

\[
\text{foreach pair of identifiers } (i^1, i^2) \text{ in } \text{pairs} \text{ do}
\]
// Try to mark this pair
if \(S_4(i^1 \circ K) \mod \gamma = 0\) then
\[
\begin{align*}
  j &\leftarrow S_4(i^1 \circ K) \mod \xi; \quad */ \text{ bit index } */ \\
  k &\leftarrow S_3(i^2 \circ K) \mod m[n]; \quad */ \text{ letter index } */ \\
  \text{if } (\overline{v}[j]^i) \neq (\overline{v}[j]^i) \text{ then } */ \text{ bits } 1-0 \text{ or } 0-1 */ \\
  \text{mask} &\leftarrow S_4(i^2 \circ K) \mod 2; \\
  \text{mark} &\leftarrow m[k] \oplus \text{mask}; \quad */ \text{ mark bit */} \\
  \overline{v}[j]^i &\leftarrow \text{mark}; \\
  \overline{v}[j]^i &\leftarrow \text{not (mark)};
\end{align*}
\]
if constraints in \(L\) are violated then
\[
\begin{align*}
  \overline{v}[j]^i &\leftarrow \text{not (mark)}; \\
  \overline{v}[j]^i &\leftarrow \text{mark}; \quad */ \text{ undo modifications */}
\end{align*}
\]
else
\[
\begin{align*}
  &\text{add } (i^1, i^2) \text{ to } \text{markList};
\end{align*}
\]
return markList;

Algorithm 4: ComputePairs(document d, wis- 
constraints \(\{\psi_1, \ldots, \psi_n\}, L, K\))

\[
\text{M}(d) \leftarrow \text{ExtractModifiableIdentifiers}(d, L); \\
\text{construct a matrix } M \text{ with } |M(d)| \text{ rows and two columns};
\]

\[
\text{foreach } i \in M(d) \text{ do}
\]
\[
\begin{align*}
  \text{set the } i\text{-th row of } M \to (i, \begin{array}{c} \vdots \end{array} 0 \ldots 0); \\
  \text{foreach wis-constraint } \psi_j \text{ do}
\end{align*}
\]
\[
\begin{align*}
  &\text{foreach } i \in I(\psi_j, d) \text{ do } /* \text{compute } \psi_j */ \\
  &\text{set } p[j] \text{ to } 1 \text{ in the } i\text{-th row } (i, p) \text{ of } M; \\
  &\text{sort } M \text{ according to } p \text{ and hash }(r \circ K); \\
  &\text{// identifiers with the same dependencies } \\
  &\text{look randomly shuffled set cursor to the beginning of } M; \\
  &\text{repeat}
\end{align*}
\]
\[
\begin{align*}
  &\begin{align*}
  (i^1, p^1) &\leftarrow \text{NextRow}(M); \\
  (i^2, p^2) &\leftarrow \text{NextRow}(M);
\end{align*} \\
  &\text{if } (p^1 = p^2) \text{ then } /* \text{same dependency } */ \\
  &\text{add } (i^1, i^2) \text{ into } \text{pairs};
\end{align*}
\]
\[
\text{until (end of } M); \\
\text{return } \text{pairs};
\]

5.3 Detection

The detector considers each identified value \(\overline{v}[i]\) that was 
potentially watermarked, based on the set of recorded pairs 
and on the secret key. The pseudorandom generator is used 
afterward, similar to [2] and [17], to obtain the position \(j\) of 
the watermark bit. The bit value \(\overline{v}[i][j]\) on this position is 
masked by an exclusive OR with a pseudorandom mask bit. 
The result is stored in \text{readMark}. This bit accounts for 
the letter \(k\) of the hidden message \(m\), where \(k\) is also computed 
by the pseudorandom generator. The vote for the value 
\text{readMark} of this bit \(k\) is incremented in the \text{vote} array (see 
Algorithm 5). Then, each position \(k\) in the detected message 
is assigned one of values 0, 1, or "undef." A binary value is 
assigned if the number of votes for one value is 
significantly higher than for the other one, using majority 
voting (ThresholdMajority).

Algorithm 5: PairDetect(suspect document 
\(d, \text{markList}, \text{message length } l, \text{key } K, \xi\))

\[
\text{for } k \in \{1, \ldots, l\} \text{ do}
\]
\[
\begin{align*}
  &\text{vote}[k][0] \leftarrow 0; \\
  &\text{vote}[k][1] \leftarrow 0;
\end{align*}
\]
\[
\text{for } (i^1, i^2) \text{ in } \text{markList} \text{ do}
\]
\[
\begin{align*}
  &j \leftarrow S_2(i^1 \circ K) \mod \xi; \quad */ \text{ bit index */} \\
  &k \leftarrow S_3(i^2 \circ K) \mod m[l]; \quad */ \text{ letter index */} \\
  &\text{mask} \leftarrow S_4(i^2 \circ K) \mod 2; \quad */ \text{ mask bit */} \\
  &\text{readMark} \leftarrow \overline{v}[j][k] \oplus \text{mask}; \quad */ \text{ read the mark */} \\
  &\text{vote}[k][\text{readMark}] \leftarrow \text{vote}[k][\text{readMark}] + 1; \\
  &\text{// voting}
\end{align*}
\]
return ThresholdMajority(\text{vote});

5.4 Enhancements and Extension to Other 
Constraints

ComputePairs, efficient pairs computation. The core of the 
method is the computation of pairs of identifiers. Its 
implementation must be optimized to achieve scalability. 
In our prototype, this task is mainly devoted to the RDBMS 
by computing and storing the dependency matrix as a 
relation. Suppose that \(\psi_1, \ldots, \psi_n\) are \(n\) constraints. We 
create a relation \(\text{matrix}((\text{id}, \text{dep_pattern}))\) such that 
the \text{id} column will contain the identifiers and the column 
\text{dep_pattern} will contain their dependencies expressed 
by binary patterns. If \((i, p)\) is a tuple from \(\text{matrix}\), the 
\(k\)th bit of \(p\) has the value 1 if \(k\) depends on the tuple 
identified by \(i\) and 0 otherwise. For our examples, the tuples 
of \(\text{matrix}\) are \((\text{Dinan}; 11), (\text{Challans}; 11), (\text{Colmar}; 01),\) 
and \((\text{Chalain}; 01)). In order to compute the sets of identifiers 
that have the same dependencies, we order the tuples in 
\text{matrix} according to their binary pattern. Then, the tuples 
involved in the same constraints are shuffled.

\[
\text{SELECT id FROM matrix}
\]
\[
\text{ORDER BY dep_pattern, md5(concat(Kp, id));}
\]

Pairs of identifiers having the same dependencies are 
obtained by reading pairs of identifiers from the sorted 
\text{matrix} relation.

Matrix reduction and nonzero constraints. It is possible to 
reduce the number of lines of the dependency matrix. 
Observe first that if \(\psi_1\) and \(\psi_2\) depend on exactly 
the same values, we can use only \(\psi_1\) in the dependency 
matrix without changing the solution. Second, this technique fits 
well for zero-distortion constraints. For the sake of 
simplicity, suppose that we only encode the marks 
\(+1\) or \(-1\) and that all the constraints have the same global 
distortion \(t\). Hence, if two queries \(\psi_1\) and \(\psi_2\) depend on 
the same values except on \(t\) positions, using only \(\psi_1\) in the 
dependency matrix may introduce a maximal distortion of 
at most \(t\) on query \(\psi_2\). For the general case, we delete all 
queries that are identical up to \(t\) divided by the maximal 
allowed local distortion on each element.

Capacity. Theoretical arguments [8] show that we are 
likely to find pairs on natural data sets. We assess this 
property by our experiments in Section 8.

Handling join and invariant constraints. For a join 
condition \(\overline{w}[i] = \overline{w}[j]\), we suppress \(j\) from the set of modifiable 
identifiers \(M\). When a value is assigned to \(\overline{w}[i]\), we 
propagate it to \(\overline{w}[j]\). For invariant constraints, we simply 
set to zero the allowed distortion on each considered 
identified value.
Robustness. An attacker that performs random alterations is more likely to hit a bit embedded in a pair than a bit embedded at a single position. Therefore, a watermark bit embedded in a pair is less robust than a bit embedded at a single position. This slight loss of robustness is traded for computational speedup and collusion security, as explained in the next section. This phenomenon has to be put into balance with the large amount of pairs discovered by the pairing algorithm. This allows for a large repetition of the bit encoding, which enforces robustness. Experimental evaluations (Section 8) assess this property. If an attacker knows which usability constraints are preserved in his watermarked data set, a more sophisticated attack may be envisioned. By running ComputePairs (which is assumed to be public), an attacker might find the dependencies of identified values. However, even if he knows that paired tuples share the same dependencies, the exact pairing cannot be guessed, because pairs are chosen according to a secret order, known only by the data owner.

Complexity. The following table sums up the number of query computations needed to find $T$ distinct watermarks. Parameter $n_l$ denotes the number of linearizable constraints, and $n_g$ denotes the number of nonlinear constraints. Remember that each query must be computed on a likely huge number of tuples.

<table>
<thead>
<tr>
<th>Method</th>
<th>#(query computation)</th>
<th>#(ILP solving)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>$T(n_g + n_l)$</td>
<td>0</td>
</tr>
<tr>
<td>Linear</td>
<td>$n_l + Tn_g$</td>
<td>$T$</td>
</tr>
<tr>
<td>Pairing</td>
<td>$n_l + Tn_g$</td>
<td>0</td>
</tr>
</tbody>
</table>

6 Collusion-Secure Fingerprinting

Suppose that three fingerprints $m_1 = 010$, $m_2 = 100$, and $m_3 = 110$ are used for the three users $u_1$, $u_2$, and $u_3$. If $u_1$ and $u_2$ compare their respective databases, they will observe that they differ on the positions where the first and second bits of $m_1$ and $m_2$ have been embedded. Indeed, these bits are different in fingerprints $m_1$ and $m_2$. Therefore, $u_1$ and $u_2$ can guess that the embedding process modified the original database at these precise locations. On the other hand, the positions where the third bit of the fingerprint messages was embedded are not revealed, because this bit is the same in $m_1$, $m_3$, and $m_3$. Suppose now that $u_1$ and $u_2$ combine their databases to obtain a third database in which the values located at the modified positions are taken from both databases. Since $m_3$ has the first bit of $m_2$ and the second bit of $m_1$, it can happen that $u_1$ and $u_2$ actually build a database that carries the fingerprint of $u_3$. At detection time, $v_3$ will be considered as suspect. This kind of coalition of users that collude for framing another one, which is innocent, must clearly be avoided. To achieve this, frame-proof collusion-secure codes [3, 18, 23] have been designed. Basically, they must provide the following two features: 1) no coalition of at most $c$ users can frame another innocent user, and 2) there exists a tracing algorithm $t$ that, given a suspect fingerprint, outputs at least one of the members that participated to the framing coalition. A codebook of collusion-secure fingerprints is characterized by the length of its words, which depends on the size $c$ of the maximum coalition for which the code is frameproof, the maximum probability $\varepsilon$ of error of $t$, and the maximum number $M$ of fingerprints. For instance, this length is $O(c^4 \log(1/\varepsilon) \log(M/\varepsilon))$ for Boneh and Shaw’s scheme [3] ($O(c^4 \log c \log(M/\varepsilon)$) with the improvements of Schauhun [18]) and $100 c^2 \log((1/\varepsilon))$ for Tardos’ scheme [23]. For instance, if we want to be frameproof against any coalition of at most $c = 3$ purchasers with a probability of error of $\varepsilon = 10^{-4}$, the length of the fingerprints is 3,600. To be effective, there must be some fixed set of positions within the database in which all the codewords can be embedded without affecting the usability. A recurrent problem of collusion-secure codes is that their length tends to increase quite quickly, making them not suitable for all practical fingerprinted applications.

Collusion-secure fingerprinting techniques have already been applied to databases [17] but without taking into account the usability constraints. Query-preserving watermarking is highly challenging since usability constraints drastically reduce the embedding bandwidth. The greedy approach [21] has some limitations that prevent the use of efficient collusion-secure fingerprints. For example, it may embed the codeword for user $u_1$ in positions, say, $i_1$, $i_2$, $i_3$ while embedding the codewords for $u_2$ and $u_3$ in positions $i_4$ to $i_7$, that is, in completely different positions than those for $u_1$. Hence, the databases of $u_2$ and $u_3$ differ on all watermarked positions, and all the bits of $u_1$ are exposed. Consequently, the fixed-position requirement is not satisfied by the greedy method. Moreover, this algorithm might actually embed a significantly small number of fingerprints, since it discards the modifications performed by the watermarking algorithm when they do not satisfy the usability constraints. The use of the pairing heuristic enables our algorithm to discover a large mark embedding area, making the collusion-secure fingerprinting possible for large databases, whereas the greedy approach does not allow it. Furthermore, the modified positions do not change when different messages need to be inserted. This fits the fixed set of positions required by the collusion-secure codebooks that are used. In the implementation of WATERMILL [14], we chose to use the Tardos scheme [23] for the following three reasons: 1) the codewords have a relatively small length compared to the other schemes in our setting (large data sets and a small number of purchasers), 2) the number of users $M$ does not need to be a priori fixed, and 3) its implementation is simple and efficient. The codewords of a Tardos scheme are randomly generated according to $M$ independent identically distributed Bernoulli variables whose parameters are functions of $M$ and $k$.

7 Analysis

In this section, we provide an analysis of false-hit and false-miss occurrence probabilities. Both probabilities highly depend on the detection threshold $\alpha$. If $\alpha \rightarrow 1/2$, the detector is very selective and will lead to a small rate of false hits but at the price of a loss of robustness (many false misses). On the contrary, if $\alpha \rightarrow 0$, there will be many false hits, but the detection is going to be very robust (very few false misses).

7.1 False Hits

A false hit is the wrong detection of a watermark in a nonwatermarked database. In the case of 1-bit fingerprints, we note that $v_0 = \text{vote}_c[0]$, $v_1 = \text{vote}_c[1]$, and $m = v_0 + v_1$. 
A false hit occurs as soon as $v_0/m > 1/2 + \alpha$ (fingerprint 0 is detected) or $v_1/m > 1/2 + \alpha$ (fingerprint 1 is detected). When $v_0/m$, $v_1/m \leq 1/2 + \alpha$, no defined fingerprint is detected. For a nonwatermarked data set, the value of each readMark (see the LinearDetect and PairDetect algorithms) can be modeled as the outcome of a binomial law of parameter $1/2$. We define the random variable $S$ as the sum of the $m$ random independent variables, thus modeling the sum of $m$ readMarks. Clearly, $E[S] = m/2$, and the probability of false hits is the probability $Pr(|S - E[S]| \geq m\alpha)$. Using this model, we obtain the following theorem:

**Theorem 1.** For all $\delta > 0$, if $\alpha \geq \alpha^+(m) = \sqrt{-\log \delta/2m}$, then the false-hit occurrence probability (for 1-bit fingerprints) is less than $\delta$.

**Proof.** It has been shown by Hoeffding [11] that for $m$ independent random variables $X_1, \ldots, X_m$ that are bounded ($a_i \leq X_i \leq b_i$), their sum $S$ satisfies the following inequality for any $t > 0$:

$$Pr\left(|S - E[S]| \geq mt\right) \leq \exp\left(-\frac{2m^2t^2}{\sum_{i=1}^{m} (a_i - b_i)^2}\right).$$

Applied here with $t = \alpha$, $a_i = 0$, and $b_i = 1$, this inequality gives

$$Pr\left(|S - \frac{m}{2}| \geq \alpha m\right) \leq \exp\left(-\frac{2m^2\alpha^2}{\sum_{i=1}^{m} 1}\right) = \exp(-2m\alpha^2),$$

whence

$$Pr\left(|S - \frac{m}{2}| > \alpha\right) \leq Pr\left(|S - \frac{1}{2}| \geq \alpha\right) \leq \exp(-2m\alpha^2).$$

Hence, if

$$\alpha \leq \alpha_0 = \sqrt{-\frac{\log \delta}{2m}}, \exp(-2m\alpha^2) \leq \exp(-2m\alpha^2_0) = \delta.$$

$\square$

### 7.2 False Misses

A false miss occurs when the detected message on a modified data set (for example, an attacked one) contains undefined bits or when it is different from the embedded one. Here, we perform an analysis in the 1-bit-fingerprint case. We model attacks using the model of $p$-attacks. A $p$-attack is an attack for which each readMark has the probability $p$ to be inverted. Common attacks like random bit-flipping and translation attacks are captured by this model (for different values of $p$ though). Suppose that the embedded fingerprint is 0. Then, a false miss occurs on the attacked data set if $v_0/m \leq 1/2 + \alpha$. If $v_1/m > 1/2 + \alpha$, the invalid message 1 is detected. If $v_1/m \leq 1/2 + \alpha$, an “undef” message is detected.

**Theorem 2.** For all $0 < p < \frac{1}{2}$, if $\alpha \leq \alpha^- (m) = \frac{1}{2} - p - \sqrt{-\log \delta/2m}$ and $m \geq -\frac{8\log \delta}{(1-2p)^2}$, then the false-miss occurrence probability (for 1-bit fingerprints) over the $p$-attacked database is less than $\delta$.

**Proof.** Let $0 < p < \frac{1}{2}$, $\alpha = \frac{1}{2} (\frac{1}{2} - p)$, and $t = (\frac{1}{2} - p) - \alpha$. Hence, $t = \frac{1}{2} (\frac{1}{2} - p) = \alpha > 0$. In the context of a $p$-attack, the probability that each readMark is preserved is $1 - p$. Therefore, the intended value for $\frac{v_0}{m}$ is $1 - p$, and we can quantify the probability of diverting from this value using the Hoeffding inequality:

$$Pr\left(|v_0/m - (1-p)| \geq t\right) \leq \exp(-2mt^2).$$

Let $x \in [0, \frac{1}{2} + \alpha]$. Then, $x \leq \frac{1}{2} + \alpha$. Since

$$\frac{1}{2} + \alpha = 1 - p - t, x \leq 1 - p - t, \text{ and } x - (1 - p) \leq -t < 0.$$

Hence, $|x - (1-p)| = (1-p) - x \geq t$. In other words, $x \leq \frac{1}{2} + \alpha \Rightarrow |x - (1-p)| \geq t$. From this implication, we obtain that

$$Pr\left(|v_0/m - 1/2| \leq \alpha\right) \leq \exp(-2mt^2).$$

Suppose now that $m \geq -\frac{8\log \delta}{(1-2p)^2}$. Then,

$$\exp(-2mt^2) \leq \exp\left(2 \frac{8 \log \delta}{(1-2p)^2} \right) \leq \delta,$$

and

$$Pr\left(|v_0/m - 1/2| \leq \alpha\right) \leq \delta.$$

$\square$

**Example.** Suppose that $\xi = 3$. If an attacker randomly selects one of the $\xi$ LSBs and inverts it, the probability for each watermarked bit to be inverted is $1/3$. If we want to have both false-hit and false-miss occurrence probabilities below $\delta$, we need to have $\alpha^+(TC) \leq \alpha \leq \alpha^-(TC)$, which can be achieved using $\alpha = \frac{1-2p}{2}$ as soon as $\alpha^+(m) \leq \alpha^-(m)$, that is, when $m \geq -\frac{16\log \delta}{(1-2p)^2}$. For $\delta = 10^{-3}$, we obtain $m \geq 994.71$. If the database contains 10,000 tuples, setting $\gamma = 10\left(\frac{10,000}{994.71}\right)$ for watermarking achieves the target robustness.

### 7.3 Extension to Arbitrary Length Fingerprints

Suppose in the following that the embedded fingerprints have the length $l$. We note that $m_i = vote[i][0] + vote[i][1]$.

**Corollary 1.** For all $\delta > 0$, if $m = \min(m_i)$ and $\alpha \geq \alpha^+(l) = \sqrt{-\log \delta/2l}$, the probability that a fully defined message is detected in a third-party data set is less than $\delta$.

**Proof.** For $i = 1 \ldots n$, the probability that the bit $i$ of the detected message is defined is at most

$$\exp(-2m_ia_\alpha^2) \leq \exp(-2ma_\alpha^2) \leq \delta^{l/4}.$$

Then, for a message of $l$ bits, the probability that all $l$ bits are defined is at most $\delta$. $\square$

Notice that this corollary deals only with detecting a fully defined message. Not every message may be valid if the owner has only distributed $N$ copies. In this case, the probability that a valid fingerprint is recovered is at most $\frac{N}{2} \delta$. An interesting case is the one of the $p$-attacks on data
sets in which fingerprints of length \( l > 1 \) were embedded. In the 1-bit case, \( p \)-attacks when \( p < 1/2 \) were not discussed. Indeed, they were likely to invert the fingerprint and thus avoid detection. When fingerprints contain several bits, if \( p < 1/2 \), then all the bits of the fingerprint are very likely to be simultaneously inverted. In that case, the recovered fingerprint may seem as suspect as the original one (think of it as a negative image of a black-and-white picture). Hence, attacks are going to be effective only if \( p = 1/2 \), which is possible (for example, by inverting all bits of the data set) but, at the same time, destructive since all the values are altered.

8 Experimental Results

Context. Our methods are implemented on our open platform WATERMILL [14]. The experiments were performed on a workstation running Debian GNU/Linux (AMD64 port). The hardware includes an EMT64 Pentium 4 3.2-GHz HT Intel processor, 1 Gbit of RAM, and an 80-Gbit 7200RPM 8-Mbyte cache SATA hard drive. We use MySQL version 5.0.18 (Debian packaged), Sun’s Java J2RE 1.4.2 Standard Edition (Hotspot 64-bit Server VM), and JDBC is supported through MySQL-connector-Java version 3.1.6. No special hardware or software (except for the WATERMILL prototype) tuning was performed. MySQL databases use MyISAM, for example, a transactionless physical storage engine. Swap space is 2 Gbits.

Benchmark data sets. Experiments were performed on two different data sets: a synthetic data set and the Forest CoverType database from the UCI KDD archive [10]. The synthetic relational database corresponds to a sales database with \( n \) products, each product having an associated cost. A number of \( p \) shopping carts are filled with random subsets of \( k \) products. We denote such an instance by \( B(n, p, k) \). For different values of \( n, p, \) and \( k \), we modified the cost attribute with the following usability constraints: The distortion on the cost of each product must not exceed 1 Euro, and the distortion on the total cost for each shopping cart must not exceed 1 Euro.

Observe that for an increasing number of carts, these constraints are very restrictive and hard to respect simultaneously, even on a small data set. The second set of experiments was performed on the Forest CoverType database [10]. This database gathers information on forest parcels, for a total of 581,012 tuples. We have restricted our attention to the elevation and the aspect attributes. We created virtual primary keys for the data set that do not exist in the original data set. We watermarked the aspect attribute, with a local distortion of 1. We split the elevation values into 50 random overlapping intervals. The 50 corresponding usability constraints impose that the mean (that is, sum) of aspects of data with elevation in the same interval must not be altered by more than 1 unit (meaning a maximal global distortion of 1).

8.1 Speed and Capacity

8.1.1 Synthetic Data Set

For the instance \( B(5000,1000,3) \), checking the global constraints takes, on the average, 145 s. If \( \gamma = 10 \) (1 tuple out of 10 is altered), the global constraints are going to be checked about 500 times for the greedy method, requiring more than 20 hours. Note that \( B(5000,1000,3) \) is not a large database. Consequently, for huge data sets, using the greedy method is impossible since it requires to perform costly computations each time a watermark bit is to be embedded. For our experiments, we used only small data sets to be able to make a comparison between the two methods. The greedy method refers to the method of Agrawal et al. [2] with a check of the constraints every time a bit is modified. If the constraints are respected, the modification is accepted; otherwise, it is discarded. To compare the speed and capacity, we performed a series of experiments, using the instance \( B(1000,3) \), that is, three products per cart. For a number of carts \( p \) ranging from 10 to 50, we compared watermarking times, the number of watermarked bits, and watermarking rates (the number of valid watermarked bits per second) for both greedy and pairing algorithms. For each experiment, two values were recorded: the time to obtain a watermarked database and the number of altered bits. A higher number of bits is better because it allows for a larger embedding bandwidth. For the pairing method, two time values were recorded: the highest represents the precomputation of the pool, whereas the lowest is the time to obtain a watermarked database once the pool has been precomputed. Results are presented in Fig. 3, with the watermarking speed using a logarithmic scale in Fig. 3a, the watermarking capacity in Fig. 3b, and the watermarking rate in Fig. 3c. Clearly, the pairing algorithm outperforms the greedy one from the speed point of view. It can be argued than significantly more bits can be hidden using the greedy method. Indeed, the capacity is three to four times higher for the greedy method. There are several reasons for this. First, bits are altered by pairs (a factor 2). Second, not all pairs are altered, only the one having different binary values at the watermarked positions (statistically, another factor 2). Third, the pairing algorithm does not find all possible alterations but only a subset of them, whereas the greedy method does. Nevertheless, the watermarking rate is about 10 times better for the pairing method.

8.1.2 Forestcover Data Set

With the Forest CoverType data set, experiments show that more than 70,000 bits can be embedded using the pairing algorithm (with \( \gamma = 8 \)). To reach the same number of watermarked bits, the greedy method would have to check the constraints at least 70,000 times, which is expected to be very slow. Table 1 presents the results. The greedy method requires more than two days of computation, whereas our method needs only a few minutes. The situation becomes worse when another fingerprinted instance with the same constraints has to be obtained. Suppose that we want to distribute fingerprinted copies to four different purchasers. Here, the fingerprints can be coded as binary strings of length 2. Each one of the \( n \) valid bit embedding positions is randomly mapped to one of the 2 bits of the fingerprint. Then, it can be shown that on the average, the whole fingerprint is embedded \( m \approx \frac{2}{n} \) times (that is, both bits 1 and 2 of the fingerprint are embedded more than \( m \) times).

In our example, for four purchasers, the fingerprint is redundantly embedded 24,026 times using the pairing method and 19,906 times using the greedy one. Going back
to our introductory example, if the value time window of the Forest CoverType data set is one week, it is not possible to distribute it to four customers using the greedy method. Indeed, each fingerprinted instance requires more than two days of computation, that is, more than eight days for four customers.

8.2 Robustness

An attack is considered effective if it successfully cheats the detector while introducing a distortion on the data comparable to the distortion introduced by the watermarking process. Here, we measure the distortion by the mean of squared errors $mse$ (we do not use the mean since it is not modified by the pairing algorithm). If $d$ and $d'$ are two databases such that $I(d) = I(d')$, $mse(d, d')$ is defined as follows ($N = |I(d)|$):

$$mse(d, d') = \frac{1}{N} \sum_{i=1}^{N} (\bar{v}(d)[i] - \bar{v}(d')[i])^2.$$  

For a database $\tilde{d}$ obtained by watermarking $d$,

$$mse(d, \tilde{d}) = \|\bar{\omega}\|^2 / N.$$  

8.2.1 Subset Attacks

A subset attack consists of discarding from the relation every tuple with a probability $q$. Fig. 4 shows the detection...
ratio of watermarked bits against the value of $q$ for an instance $B(1,000,100,5)$. For each experiment, the threshold value $\alpha$ was chosen so that the false-hit occurrence probability remains below 0.1 percent. Points within a radius $\alpha$ of $1/2$ have been colored in gray. They represent the watermark removal area. Observe that the detection ratio remains 1 for all attacks. Note also that in order to keep a false-hit occurrence probability under 0.1 percent, $\alpha$ increases with $q$. The attacks always fail unless $\gamma = 3$ and $q > 80$ percent, that is, when the attack discards almost the entire data set.

8.2.2 Data Alteration Attacks

Another kind of attack consists of modifying the values within the data set. Note that such modifications are likely to break usability constraints. An $\varepsilon$-attack is an attack where a bounded random distortion is added to a randomly selected set of tuples. The maximum distortion is called the amplitude and is noted $\varepsilon$. To each tuple (watermarked or not) is added a random distortion $d \ (0 \leq d < \varepsilon)$ with a probability $i$. We followed the protocol with $\xi = 1$:

1. Create the instance $I = B(1,000,100,5)$,
2. Precompute the constraints on $I$, and
3. For each $(\varepsilon, \gamma) \in \{2,3\} \times \{1,2,3\}$
   - get a watermarked instance $I$, with insertion rate $\gamma$,
   - for each value of $\varepsilon$ and each value of $i \in 0 \ldots 100$, get an attacked instance $I_{\varepsilon,i}$,
   - compute the bit detection ratio $x$ and the mean squared error $mse$ after the attack, and
   - plot $(mse, x)$.

The results of the experiments (see Fig. 5) are represented with the $mse$ of the attack displayed horizontally and the detection ratio displayed vertically. The watermark removal area is displayed as a gray rectangle (chosen here so that the false-positive occurrence probability is at most 0.1 percent). Graphs are labeled by the $mse$ of the watermarking process. Successful (respectively, unsuccessful) attempts to remove the watermark are pictured by dots inside (respectively, outside) the gray rectangle. The majority of successful attacks

Fig. 5. Watermark detection after subset $\varepsilon$-attacks. (a) $\varepsilon = 2$, $mse = 0.23$, and $\gamma = 1$. (b) $\varepsilon = 3$, $mse = 0.242$, and $\gamma = 1$. (c) $\varepsilon = 2$, $mse = 0.11$, and $\gamma = 2$. (d) $\varepsilon = 3$, $mse = 0.132$, and $\gamma = 2$. (e) $\varepsilon = 2$, $mse = 0.08$, and $\gamma = 3$. (f) $\varepsilon = 3$, $mse = 0.008$, and $\gamma = 3$.
is observed for $\varepsilon = 2$ (only one attack is successful for $\varepsilon = 3$). This can be explained by the fact that the lower $\varepsilon$, the higher is the probability for an alteration to “hit” the watermarked bit. Second, if the attacker wants to erase the mark, he has to alter the quality of the data significantly more than the watermarking process. For instance, when $\gamma = 2$ and $\varepsilon = 2$ (Fig. 5c), the first successful attacks are observed for a mean squared error of 0.5, whereas the watermarking process introduced an error of 0.11. Once more, the price an attacker has to pay for a perfect watermark removal is significantly higher than the one paid for watermarking.

Collusion-secure fingerprinting. On the Forest Covertype benchmark, the pairing heuristics identifies 48,052 embedding positions in roughly 14 minutes. By setting $\gamma = 4, 200$, the execution of the greedy methods takes roughly the same amount of time but locates only 93 embedding positions. Assuming that each bit of the fingerprint is embedded five times, we can embed fingerprints that have length 48,052/5 \(\approx 10,000\). Using the Tardos codebook [23], the maximum size $c$ of a coalition of users against which the scheme is frameproof can be computed as $c = l/(100 \log(1/\varepsilon))^{1/2}$. For $\varepsilon = 10^{-4}$, we obtain $c \approx 3.3$. Fingerprints are then frameproof against a coalition of three users.

9 Related Work
Several recent works consider relational databases watermarking [2], [8], [15], [17], [21]. Agrawal et al.’s method [2] hides information in the LSBs of numerical attributes. The database owner can control the alteration on attributes by setting the number of LSB that can be modified. Although a small overall distortion on the mean of the watermarked attributes is observed, more general usability constraints are not considered, like the ones preserving the result of important SQL queries. Their technique was extended [17] to collusion-resilient fingerprinting by using collusion-secure codebooks [3]. However, again, usability constraints are not handled. Sion et al. [21] introduced the greedy method for watermarking with usability constraints. They handle potentially any kind of constraints by calling external checking programs (usability plug-ins). This very general method is not optimized as explained in the introduction, since the syntactical form of usability constraints is not explored (but another kind of optimization is introduced, since the syntactical form of usability constraints is not explored (but another kind of optimization is considered in [20]). It can also be applied to fingerprinting, but the computational effort is tremendous. Combining their method with collusion-secure codebooks is also possible but with some limitations: There is absolutely no guarantee that the same watermark positions will be found for all watermark messages in the codebook since their method is greedy. Besides, it is noteworthy that their embedding method, which differs from the LSB embedding introduced by Agrawal et al., can be adapted to our approach with a small effort. Weight-independent sum constraints were introduced from the theoretical point of view by one of the authors of the present paper in [8]. This specific query pattern was studied in order to obtain a lower bound on the number of distinct acceptable watermarks that one is likely to discover. The algorithmic counterpart of this previous paper is less suited for practical applications. The present paper considers algorithms that behave correctly with large data sets. Moreover, the algorithms of this previous work were not blind, whereas our new algorithms are. Finally, collusion attacks and other constraints described in the current paper were not considered in [8].

10 Conclusion and Future Work
In this paper, we present an optimization technique for the discovery of good watermarks in a data set that respects several usability constraint patterns. We have also considered the problem of collusion-secure fingerprinting under these constraints. Natural extensions of this work are the following: First, the number of our constraint patterns could certainly be increased. Second, we would like to address database fingerprinting where purchasers do not share the same usability constraints. Finally, we would like to devise tools for proving ownership on data sets after a specific rewriting.

References
Julien Lafaye received the degree from École Polytechnique, Paris, in 2003, the MSc degree in computer science from University of Paris XI, Orsay, in 2004, and the PhD degree in computer science from the Conservatoire National des Arts et Métiers (CNAM), Paris, in 2007. His PhD dissertation is centered around constraint-preserving watermarking, with applications on relational, geographical, and XML databases. His research interests span from databases to security, with a specific focus on watermarking. He also worked on computational complexity related aspects about watermarking.

David Gross-Amblard received the MSc and PhD degrees in computer science from the University of Paris XI, Orsay, France, in 1996 and 2000, respectively. He is an assistant professor at Cedric Laboratory, CNAM, Paris, and at the Laboratoire Electronique, Informatique et Image (LE2I) Laboratory, University of Bourgogne, France. His research area is the application of finite model theory to various domains, including query approximation in spatial databases and query preservation in the watermarking context.

Camelia Constantin received the MSc and PhD degrees in computer science from the University of Paris VI, France, in 2004 and 2007, respectively. Her research interests are in the area of watermarking, popularity-based ranking, and distributed query optimization.

Meryem Guerrouani received the degree in computer engineering from the Conservatoire National des Arts et Métiers (CNAM), Paris, in 2003. She is now working for an information technology company as an expert consultant.