CERIAS Tech Report 2006-41

WATERMARKING RELATIONAL DATABASES USING OPTIMIZATION BASED TECHNIQUES

by Mohamed Shehab, Elisa Bertino, Arif Ghafoor

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Watermarking Relational Databases using Optimization Based Techniques

Mohamed Shehab[†§] Elisa Bertino[†‡§] Arif Ghafoor[†§]

shehab@ecn.purdue.edu bertino@cs.purdue.edu ghafoor@ecn.purdue.edu

[†] School of Electrical and Computer Engineering at Purdue University

[‡] Department of Computer Science at Purdue University

[§] CERIAS: Center for Education and Research in Information Assurance and Security at Purdue University

Abstract

Proving ownership rights on outsourced relational databases is a crucial issue in today internet-based application environments and in many content distribution applications. In this paper, we present a mechanism for proof of ownership based on the secure embedding of a robust imperceptible watermark in relational data. We formulate the watermarking of relational databases as a constrained optimization problem, and discuss efficient techniques to solve the optimization problem and to handle the constraints. Our watermarking technique is resilient to watermark synchronization errors because it uses a partitioning approach that does not require marker tuples. Our approach overcomes a major weakness in previously proposed watermarking techniques. Watermark decoding is based on a threshold-based technique characterized by an optimal threshold that minimizes the probability of decoding errors. We implemented a proof of concept implementation of our watermarking technique and showed by experimental results that our technique is resilient to tuple deletion, alteration and insertion attacks.

Keywords: Watermarking, Digital Rights, Optimization.

1 Introduction

The rapid growth of internet and related technologies has offered an unprecedented ability to access and redistribute digital contents. In such a context, enforcing data ownership is an important requirement which requires articulated solutions, encompassing technical, organizational and legal aspects [25]. Though we are still far from such comprehensive solutions, in the last years watermarking techniques have emerged as an important building block which plays a crucial role in addressing the ownership problem. Such techniques allow the owner of the data to embed an imperceptible watermark into the data. A watermark describes information that can be used to prove the ownership of data, such as the owner, origin, or recipient of the content. Secure embedding requires that the embedded watermark must not be easily tampered with, forged, or removed from the watermarked data [26]. Imperceptible embedding means that the presence of the watermark is unnoticeable in the data. Furthermore, the watermark detection is blinded ,that is, it neither requires the knowledge of the original data nor the watermark. Watermarking techniques have been developed for video, images, audio, and text data [24, 12, 15, 2], and also for software and natural language text [7, 3].

By contrast the problem of watermarking relational data has not been given appropriate attention. There are, however, many application contexts for which data represent an important asset, the ownership of which must thus be carefully enforced. This is the case, for example, of weather data, stock market data, power consumption, consumer behavior data, medical and scientific data. Watermark embedding for relational data is made possible by the fact that real data can very often tolerate a small amount of error without any significant degradation with respect to their usability. For example when dealing with weather data, changing some daily temperatures of 1 or 2 degrees is a modification that leaves the data still usable.

To date only a few approaches to the problem of watermarking relational data have been proposed [1, 23]. These techniques, however, are not very resilient to watermark attacks. In this paper, we present a watermarking technique for relational data that is highly resilient compared to these techniques. In particular, our proposed technique is resilient to tuple dele-

tion, alteration, and insertion attacks. The main contributions of the paper are summarized as follows:

- We formulate the watermarking of relational databases as a constrained optimization problem, and discuss efficient techniques to handle the constraints. We present two techniques to solve the formulated optimization problem based on genetic algorithms and pattern search techniques.
- We present a data partitioning technique that does not depend on marker tuples to locate the partitions and thus it is resilient to watermark synchronization errors.
- We develop an efficient technique for watermark detection that is based on an optimal threshold. The optimal threshold is selected by minimizing the probability of decoding error.
- With a proof of concept implementation of our watermarking technique, we have conducted experiments using both synthetic and real-world data. We have compared our watermarking technique with previous approaches [1, 23] and shown the superiority of our technique with respect to all types of attacks.

The paper is organized as follows. Section 2 discusses the related work which includes the available relational database watermarking techniques and highlights the shortcomings of these techniques. An overview of our watermarking technique is described in Section 3, where an overview of the watermark encoding and decoding stages is presented. Section 4 discusses the data partitioning algorithm. The watermark embedding algorithm is described in Section 5. Sections 6 and 7 discuss the decoding threshold evaluation and the watermark detection scheme. Section 8 presents the attacker model. The experimental results are presented in Section 9. Finally, conclusions are given in Section 10.

2 Related Work

Agrawal et al. [1] proposed a watermarking algorithm that embeds the watermark bits in the least significant bits (LSB) of selected attributes of a selected subset of tuples. This

technique does not provide a mechanism for multibit watermarks; instead only a secret key is used. For each tuple, a secure message authenticated code (MAC) is computed using the secret key and the tuple's primary key. The computed MAC is used to select candidate tuples, attributes and the LSB position in the selected attributes. Hiding bits in LSB is efficient. However, the watermark can be easily compromised by very trivial attacks. For example a simple manipulation of the data by shifting the LSB's one position easily leads to watermark loss without much damage to the data. Therefore the LSB-based data hiding technique is not resilient [21, 8]. Moreover, it assumes that the LSB bits in any tuple can be altered without checking data constraints. Simple unconstrained LSB manipulations can easily generate undesirable results such as changing the age from 20 to 21. Li et al. [18] have presented a technique for fingerprinting relational data by extending Agrawal et al.'s watermarking scheme.

Sion et al. [23] proposed a watermarking technique that embeds watermark bits in the data statistics. The data partitioning technique used is based on the use of special marker tuples which makes it vulnerable to watermark synchronization errors resulting from tuple deletion and tuple insertion; thus such technique is not resilient to deletion and insertion attacks. Furthermore, Sion et al. recommend storing the marker tuples to enable the decoder to accurately reconstruct the underlying partitions; however this violates the blinded watermark detection property. A detailed discussion of such attacks is presented in Section 8. The data manipulation technique used to change the data statistics does not systematically investigate the feasible region; instead a naive unstructured technique is used which does not make use of the feasible alterations that could be performed on the data without affecting its usability. Furthermore, Sion et al. proposed a threshold technique for bit decoding that is based on two thresholds. However, the thresholds are arbitrarily chosen without any optimality criteria. Thus the decoding algorithm exhibits errors resulting from the non-optimal threshold selection, even in the absence of an attacker.

Gross-Amblard [11] proposed a watermarking technique for XML documents and theoretically investigates links between query result preservation and acceptable watermarking alterations. Another interesting related research effort is to be found in [17] where the authors have proposed a fragile watermark technique to detect and localize alterations made

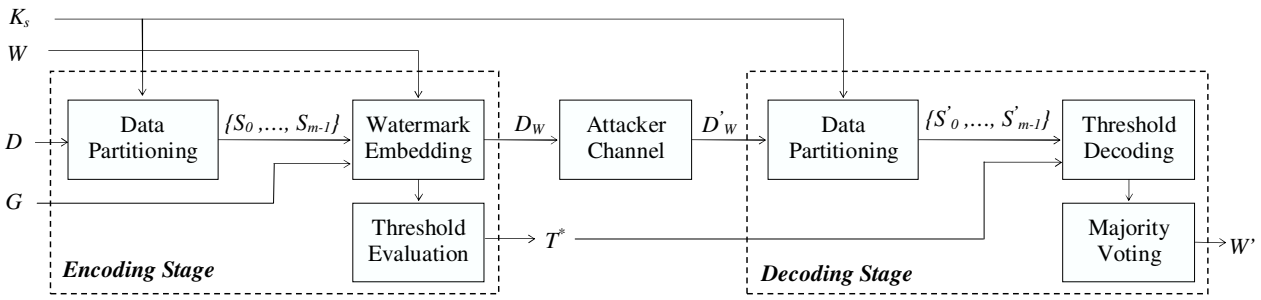


Figure 1: Stages of watermark encoding and decoding.

to a database relation with categorical attributes.

3 Approach Overview

Figure 1 shows a block diagram summarizing the main components of the watermarking system model used. A data set D is transformed into a watermarked version D_W by applying a watermark encoding function that also takes as inputs a secret key K_s only known to the copyright owner and a watermark W. Watermarking modifies the data. However these modifications are controlled by providing usability constraints referred to by the set G. These constraints limit the amount alterations that can be performed on the data, such constraints will be discussed in detail in the following sections. The watermark encoding can be summarized by the following three steps:

Step E1. Data set partitioning: by using the secret key K_s the data set D is partitioned into m non-overlapping partitions $\{S_0, \ldots, S_{m-1}\}$.

Step E2. Watermark embedding: a watermark bit is embedded in each partition by altering the partition statistics while still verifying the usability constraints in G. This alteration is performed by solving a constrained optimization problem.

Step E3. Optimal threshold evaluation: the bit embedding statistics are used to compute the optimal threshold T^* that minimizes the probability of decoding error.

The watermarked version D_W is delivered to the intended recipient. Then it can suffer from unintentional distortions or attacks aimed at destroying the watermark information. Note that even intentional attacks are performed without any knowledge of K_s or D, since these are not publicly available.

Watermark decoding is the process of extracting the embedded watermark using the watermarked data set D_W, the secret key K_s and the optimal threshold T^*. The decoding algorithm is blind as the original data set D is not required for the successful decoding of the embedded watermark. The watermark decoding is divided into three main steps:

Step D1. Data set partitioning: by using the data partitioning algorithm used in **E1**, the data partitions are generated.

Step D2. Threshold based decoding: the statistics of each partition are evaluated and the embedded bit is decoded using a threshold based scheme based on the optimal threshold T^*.

Step D3. Majority voting: The watermark bits are decoded using a majority voting technique.

In the following sections we discuss each of the encoding and decoding steps in detail.

4 Data Partitioning

In this section we present the data partitioning algorithm that partitions the data set based on a secret key K_s. The data set D is a database relation with scheme $D(P, A_0, \ldots, A_{\nu-1})$, where P is the primary key attribute, $A_0, \ldots, A_{\nu-1}$ are ν attributes which are candidates for watermarking and $|D|$ is the number of tuples in D. The data set D is to be partitioned into m non-overlapping partitions namely $\{S_0, \ldots, S_{m-1}\}$, such that each partition S_i contains on average $\frac{|D|}{m}$ tuples from the data set D. Partitions do not overlap, that is, for any two partitions S_i and S_j such that $i \neq j$ we have $S_i \cap S_j = \{\}$. For each tuple $r \in D$ the data partitioning algorithm computes a message authenticated code (MAC) which is considered to be secure [22] and is given by $H(K_s||H(r.P||K_s))$, where $r.P$ is the primary key of the tuple r, $H()$ is a secure hash function and $||$ is the concatenation operator. Using the computed MAC tuples are assigned to partitions. For a tuple r its partition assignment is given by

$$partition(r) = H(K_s||H(r.P||K_s)) \ mod \ m$$

Using the property that secure hash functions generate uniformly distributed message digests this partitioning technique on average places $\frac{|D|}{m}$ tuples in each partition. Furthermore, an attacker cannot predict the tuples-to-partition assignment without the knowledge of the

secret key K_s and the number of partitions m which are kept secret. Keeping m secret is not a requirement. However, keeping it secret makes it harder for the attacker to regenerate the partitions. The partitioning algorithm is described in Figure 2.

Though the presence of a primary key in the relation being watermarked is a common practice in relational data, our technique can be easily extended to handle cases when the relation has no primary key. Assuming a single attribute relation, the most significant χ bits (MSB) of the data could be used as a substitute for the primary key. The use of the MSB assumes that the watermark embedding data alterations will unlikely alter the MSB χ bits. However, if too many tuples share the same MSB χ bits this would enable the attacker to infer information about the partition distribution. The solution would be to select χ that minimizes the duplicates. Another technique, in case of a relation with multiple attributes is to use identifying attributes instead of the primary key; for example in medical data we could use the patient full name, patient address, patient date of birth.

Our data partitioning algorithm does not rely on special marker tuples for the selection of data partitions, which makes it resilient to watermark synchronization attacks caused by tuple deletion and tuple insertion. By contrast, Sion et al. [23] use special marker tuples, having the property that $H(K_s||H(r.P||K_s))\ mod\ m = 0$, to partition the data set. In Sion's approach a partition is defined as the set of tuples between two markers. Marker-based techniques not only use markers to define partitions but also to define boundaries between the embedded watermark bits. Such a technique is very fragile to tuple deletion and insertion due to the errors caused by the addition and deletion of marker tuples. This attack is discussed in more detail in Section 8.

```
Algorithm: get_partitions
Input: Data set D, Secret Key K_s, Number of partitions m
Output: Data partitions S_0, ..., S_{m-1}

1.  S_0, ..., S_{m-1} ← {}
2.  for each Tuple r ∈ D,
3.    partition(r) ← H(K_s||H(r.P||K_s)) mod m
4.    insert r into S_{partition(r)}
5.  return S_0, ..., S_{m-1}
```

Figure 2: Data partitioning algorithm

Symbol	Description
m	Number of data partitions
ξ	Minimum partition size
W	Watermark bit sequence $\{b_{l-1}, \ldots, b_0\}$
l	Length of watermark bit sequence
X_{max}	Maximization embedding statistics
X_{min}	Minimization embedding statistics
S_i	Data partition, numeric data vector in R^n
$\|S_i\|, n$	Length of vector S_i
K_s	Secret Key
T^*	Optimal decoding threshold
G_i	Usability constraints
Δ_i	Manipulation vector in R^n

Figure 3: Notation

5 Watermark Embedding

In this section we describe the watermark embedding algorithm by formalizing the bit encoding as a constrained optimization problem. Then we propose a genetic algorithm and a pattern search technique that can be used to efficiently solve such optimization problem. The selection of which optimization algorithm to use is decided according to the application time and processing requirements as will be discussed further. At the end of this section we give the overall watermark embedding algorithm. Our watermarking technique is able to handle tuples with multiple attributes as we will discuss in Section 7. However, to simplify the following discussion we assume the tuples in a partition S_i contain a single numeric attribute. In such a case each partition S_i can be represented as a numeric data vector $S_i = [s_{i1}, \ldots, s_{in}] \in \Re^n$.

5.1 Single Bit Encoding

Given a watermark bit b_i, and a numeric data vector $S_i = [s_{i1}, \ldots, s_{in}] \in \Re^n$ the bit encoding algorithm maps the data vector S_i to a new data vector $S_i^W = S_i + \Delta_i$, where $\Delta_i = [\Delta_{i1}, \ldots, \Delta_{in}] \in \Re^n$ is referred to as the manipulation vector. The performed manipulations are bounded by the data usability constraints referred to by the set $G_i = \{g_{i1}, \ldots, g_{ip}\}$. The encoding is based on optimizing encoding function referred to as the *hiding function*

which is defined as follows:

Definition 1 *A hiding function* $\Theta_\Upsilon : \Re^n \rightarrow \Re$*, where* Υ *is the set of secret parameters decided by the data owner.*

The set Υ can be regarded as part of the secret key. Note that when the hiding function is applied to $S_i + \Delta_i$ the only variable is the manipulation vector Δ_i, while S_i and Υ are constants. To encode bit b_i into set S_i the bit encoding algorithm optimizes the hiding function $\Theta_\Upsilon(S_i+\Delta_i)$. The objective of the optimization problem of maximizing or minimizing the hiding function is based on the bit b_i such that if the bit b_i is equal to 1 then the bit encoding algorithm solves the following maximization problem:

$$\begin{aligned} \max_{\Delta_i} \quad & \Theta_\Upsilon(S_i + \Delta_i) \\ \text{subject to} \quad & G_i \end{aligned}$$

However, if the bit b_i is equal to 0, then the problem is simply changed into a minimization problem. The solution to the optimization problem generates the manipulation vector Δ_i^* at which $\Theta_\Upsilon(S_i + \Delta_i^*)$ is optimal. The new data set S_i^W is computed as $S_i + \Delta_i^*$. Using contradicting objectives, namely maximization for $b_i = 1$ and minimization for $b_i = 0$, ensures that the values of $\Theta_\Upsilon(S_i + \Delta_i^*)$ generated in both cases are located at maximal distance and thus makes the inserted bit more resilient to attacks, in particular to alteration attacks.

Figure 4 depicts the bit encoding algorithm. The bit encoding algorithm embeds bit b_i in the partition S_i if $|S_i|$ is greater than ξ. The value of ξ represents the minimum partition size. The `maximize` and `minimize` in the bit encoding algorithm optimize the hiding function $\Theta_\Upsilon(S_i + \Delta_i^*)$ subject to the constraints in G_i. The maximization and minimization solution statistics are recorded for each encoding step in X_{max}, X_{min} respectively as indicated in lines 4 and 7 of the encoding algorithm. These statistics are used to compute optimal decoding parameters as will be discussed in Section 6.

The set of usability constraints G_i represents the bounds on the tolerated change that can be performed on the elements of S_i. These constraints describe the feasible space for the manipulation vector Δ_i for each bit encoding step. These constraints are application and data dependent. The usability constraints are similar to the constraints enforced on watermarking

```
Algorithm: encode_single_bit
Input: Data set S_i, Bit b_i, Constraints set G_i, Secret parameters set ϒ,
Statistics X_max, X_min
Output: Data set S_i + Δ_i^*

1.  if (|S_i| < ξ) then return S_i
2.  if (b_i == 1) then
3.    maximize(Θ_ϒ(S_i + Δ_i)) subject to constraints G_i
4.    insert Θ_ϒ(S_i + Δ_i^*) into X_max
5.  else
6.    minimize(Θ_ϒ(S_i + Δ_i)) subject to constraints G_i
7.    insert Θ_ϒ(S_i + Δ_i^*) into X_min
8.  return S_i + Δ_i^*
```

Figure 4: Bit encoding algorithm

algorithms for audio, images and video which mainly require that the watermark is not detectable by the human auditory and visual system [24, 12, 15, 8]. For example, interval constraints could be used to control the magnitude of the alteration for Δ_{ij}, that is,

$$\Delta_{ij}^{min} \leq \Delta_{ij} \leq \Delta_{ij}^{max}$$

Another example of usability constraints are classification-preserving constraints which constrain the encoding alterations to generate data that belong to the same classification as the original data. For example, when watermarking age data the results after the alteration should fall in the same age group e.g. "preschool" (0-6 years), "child" (7-13), "teenager" (14-18), "young male" (19-21), "adult" (22+), these constraints can be easily described using interval constraints as they are similar to defining bounds on Δ_i. Another interesting type of constraints may require that the watermarked data set maintain certain statistics. For example the mean of the generated data set be equal to mean of the original data set, in such a case the constructed constraint is of the form:

$$\sum_{j=1}^{n} \Delta_{ij} = 0$$

Several other usability constraints could be devised depending on the application requirements. These constraints are handled by the bit encoding algorithm by using constrained optimization techniques when optimizing the hiding function as will be discussed in the subsequent sections.

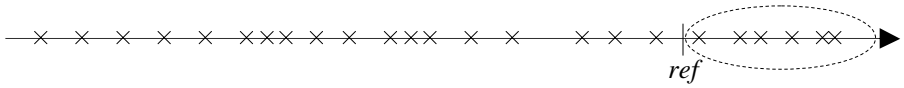


Figure 5: The distribution of the set $S_i + \Delta_i$ on the number line and the tail entries circled.

The hiding function used in this paper is dependent on the data statistics. The mean and variance estimates of the new set $S_i^W = S_i + \Delta_i$ are referred to as $\mu_{(S_i+\Delta_i)}$ and $\sigma^2_{(S_i+\Delta_i)}$ respectively; for short we will use μ and σ^2. We define the reference point as $ref = \mu + c \times \sigma$, where $c \in (0, 1)$ is a secret real number which is part of the set Υ. The data points in $S_i + \Delta_i$ that are above ref are referred to as the *"tail" entries* as illustrated by Figure 5. The hiding function Θ_c is defined as the number of tail entries normalized by the cardinality of S_i, also referred to as the normalized tail count. It is computed as follows:

$$\Theta_c(S_i + \Delta_i) = \frac{1}{n} \sum_{j=1}^{n} 1_{\{s_{ij}+\Delta_{ij} \geq ref\}}$$

where, n is the cardinality of S_i and $1_{\{\}}$ is the indicator function defined as follows:

$$1_{\{condition\}} = \begin{cases} 1 & \text{if } condition = TRUE, \\ 0 & otherwise. \end{cases}$$

Note that the reference ref is dependent on both μ and σ which means that it is not fixed and dynamically varies with the statistics of $S_i + \Delta_i$. Also note that, the normalized tail count $\Theta_c(S_i + \Delta_i)$ depends on the distribution of $S_i + \Delta_i$ and the dynamic ref.

The objective function $\Theta_c(S_i+\Delta_i)$ is nonlinear and nondifferentiable, which makes the optimization problem at hand a nonlinear constrained optimization problem. In such problems traditional gradient-based approaches turn out to be inapplicable. To solve this optimization problem we propose two techniques based on Genetic Algorithm and Pattern Search respectively. The choice of the technique to use depends on the application processing requirements. Solving the optimization problem does not necessarily require to find a global solution because finding such solution may require a large number of computations. Our main goal is to find a near optimal solution that ensures that solutions of the minimization of $\Theta_c(S_i + \Delta_i)$ and maximization of $\Theta_c(S_i + \Delta_i)$ are separated as far as possible from each other. As we will discuss further, GA could be used in order to determine global optimal solutions by trading processing time, while Pattern Search could be used to provide a local

optimal solution without trading processing time. Note that these optimization techniques will function for simpler hiding functions. However, having a simple linear hiding function makes it easier to attack. For example, if the average is used as the hiding function, in this case the optimization problem will merely be adding or subtracting a constant term to the data vector to maximize or minimize the average.

5.2 Genetic Algorithm Technique

A genetic algorithm (GA) is a search technique that is based on the principles of natural selection or survival of the fittest. Pioneering work in this field was conducted by Holland in the 1960s [13, 6]. Many researchers have refined his initial approach. Instead of using gradient information, the GA uses the objective function directly in the search. The GA searches the solution space by maintaining a population of potential solutions. Then, by using evolving operations such as crossover, mutation, and selection, the GA creates successive generations of solutions that evolve and inherit the positive characteristics of their parents and thus gradually approach optimal or near-optimal solutions. By using the objective function directly in the search, GA's can be effectively applied in nonconvex, highly nonlinear, complex problems [10, 5]. GA's have been frequently used to solve combinatorial optimization problems and nonlinear problems with complicated constraints or nondifferentiable objective functions. A GA is not guaranteed to find the global optimum; however it is less likely to get trapped at a local optimum than traditional gradient-based search methods when the objective function is not smooth and generally well behaved. A GA usually analyzes a larger portion of the solution space than conventional methods and is therefore more likely to find feasible solutions in heavily constrained problems.

The feasible set Ω_i is the set of values of Δ_i that satisfy all constraints in G_i. GA's do not work directly with points in the set Ω_i, but rather with a mapping of the points in Ω_i into a string of symbols called *chromosomes*. A simple binary representation scheme uses symbols from $\{0, 1\}$; each chromosome is L symbols long. As an example the binary chromosome representing the vector $\Delta_i = [\Delta_{i1}, \ldots, \Delta_{in}]$ is indicated in Figure 6. Note that each component of Δ_i uses L/n bits, where $n = |S_i|$. This chromosome representation automatically handles interval constraints on Δ_i. For example if Δ_{ij} can only take values in

the interval $[l_{ij}, h_{ij}]$, then by mapping the integers in the interval $[0, 2^{L/n} - 1]$ to values in the interval $[l_{ij}, h_{ij}]$ via simple translation and scaling, this ensures that, whatever operations are performed on the chromosome, the entries are guaranteed to stay within the feasible interval.

$$\underbrace{0101010101010101}_{\Delta_{i1}}\underbrace{1111111111111111}_{\Delta_{i2}}\cdots\underbrace{0011001100110011}_{\Delta_{in}}$$

Figure 6: Binary chromosome representing Δ_i

Each chromosome has a corresponding value of the objective function, referred to as the *fitness* of the chromosome. To handle other types of constraints we penalize the infeasible chromosomes by reducing their fitness value according to a penalty function $\Phi(\Delta_i)$, which represents the degree of infeasibility. Without loss of generality, if we are solving the maximization problem in Section 5.1 with constraints $G_i = \{g_{i1}, \ldots, g_{ip}\}$, then the fitness function used is $\Theta_c(S_i + \Delta_i) + \lambda\Phi(\Delta_i)$, where $\lambda \in \Re^-$ is the penalty multiplier and is chosen large enough to penalize the objective function in case of infeasible Δ_i. The penalty function $\Phi(\Delta_i)$ is given by:

$$\Phi(\Delta_i) = \sum_{j=1}^{p} g_{ij}^{+}(\Delta_i)$$

where

$$g_{ij}^{+}(\Delta_i) = \begin{cases} 0 & \text{if } \Delta_i \text{ is feasible w.r.t } g_{ij} \\ \phi(g_{ij}, \Delta_i) & otherwise \end{cases}$$

where $\phi(g_{ij}, \Delta_i) \in \Re^+$ represents the amount of infeasibility with respect to the constraint g_{ij}. For example if the constraint g_{ij} is $\sum_{j=1}^{n} \Delta_{ij} = 0$ then $\phi(g_{ij}, \Delta_i) = \| \sum_{j=1}^{n} \Delta_{ij} \|$. For a detailed discussion of penalty based techniques the interested reader is referred to [20, 5].

A GA is less likely to get stuck in local optima. However, a GA requires a large number of functional evaluations to converge to a global optimal. Thus we recommend the use of GA's only when the processing time is not a strict requirement and watermarking is performed offline. For faster performance we recommend the use of pattern search techniques discussed in the next section.

5.3 Pattern Search Technique

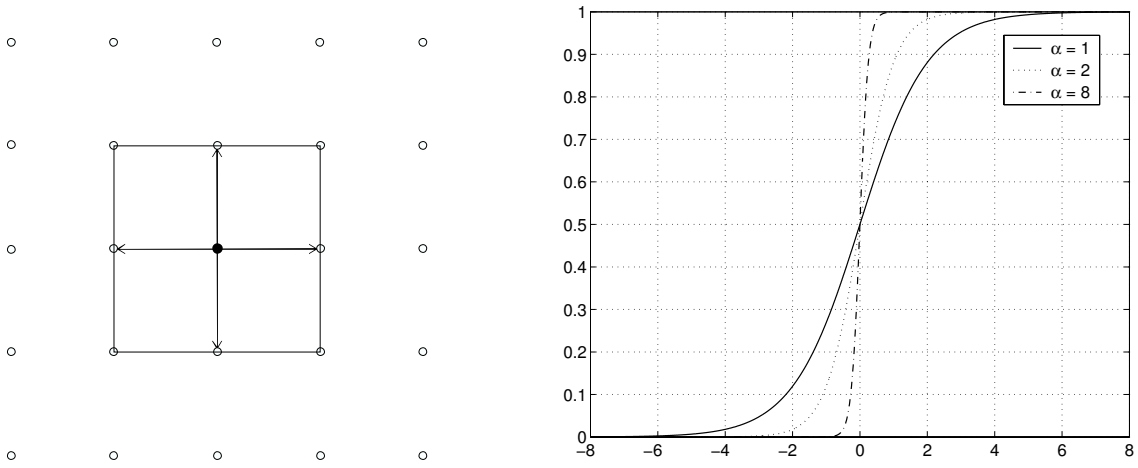
Pattern search methods are a class of direct search methods for nonlinear optimization. Pattern search methods [4, 14] have been widely used because of their simplicity and the fact that they work well in practice on a variety of problems. More recently, they are provably convergent [16, 9].

Pattern search starts at an initial point and samples the objective function at a predetermined pattern of points centered about that point with the goal of producing a new better iterate. Such moves are referred to as exploratory moves, Figure 7(a) shows an example pattern in $\Re^2$. If such sampling is successful (i.e., produces a new better iterate), the process is repeated with the pattern centered about the new best point. If not, the size of the pattern is reduced and the objective function is again sampled about the current point. For a detailed discussion on pattern search refer to [16, 9]. To improve the performance of pattern search the objective function $\Theta_c(S_i + \Delta_i)$ is approximated by smooth sigmoid functions. The objective function is approximated as follows:

$$\widehat{\Theta}_c(S_i + \Delta_i) = \frac{1}{n} \sum_{j=1}^{n} Sigmoid_{(\alpha_f, ref)}(s_{ij} + \Delta_{ij})$$

where $Sigmoid_{(\alpha,\tau)}(x)$ is a sigmoid function with parameters (α, τ), shown in Figure 7(b), is defined as $(1 - (1 + e^{\alpha(x-\tau)})^{-1})$.

Constraints could be handled using the techniques discussed earlier. However, pattern search can easily handle the constraints by limiting the exploratory moves to only the directions that end up in the feasible space; thus ensuring that the generated solution is feasible. For a more detailed discussion refer to [16]. The systematic behavior of pattern search and the adaptable pattern size leads to the fast convergence to optimal feasible solutions. However, pattern search is not guaranteed to find a global optimum. This problem can be overcome by starting the algorithm from different initial feasible points. For the sake of comparison, we conducted an experiment using normally distributed data where the tail count $\Theta_c(S_i + \Delta_i)$ was maximized and minimized using both pattern search and GA with interval constraints. Both algorithms were restricted to use an equal number of objective function evaluations. Figure 8 reports the results of this experiment, which shows that pattern search generates better optimized tail counts, and thus better separation between the maximization



(a) (b)

Figure 7: (a) Example pattern for coordinate search in $\Re^2$, as part of a larger grid. (b) Shows $Sigmoid_{(\alpha,\tau)}$ where $\tau = 0$ and $\alpha = \{1, 2, 8\}$

and minimization results. However, if GA is given more functional evaluations converges to global optimum solutions.

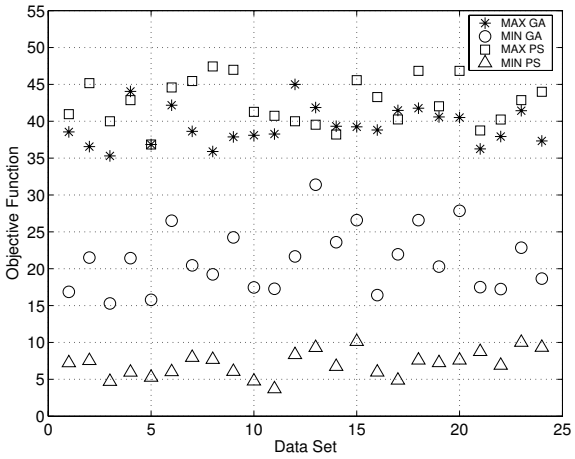


Figure 8: Comparison between GA and Pattern search (PS) for the same number of functional evaluations.

5.4 Watermark Embedding Algorithm

A watermark is a set of l bits $W = b_{l-1}, \ldots, b_0$ that are to be embedded in the data partitions $\{S_0, \ldots, S_{m-1}\}$. To enable multiple embeddings of the watermark in the data set

the watermark length l is selected such that $l \ll m$. The watermark embedding algorithm embeds a bit b_i in partition S_k such that $k \ mod \ l = i$. This technique ensures that each watermark bit is embedded $\lfloor \frac{m}{l} \rfloor$ times in the data set D. The watermark embedding algorithm is reported in Figure 9. The watermark embedding algorithm generates the partitions by calling *get_partitions*, then for each partition S_k a watermark bit b_i is encoded by using the single bit encoding algorithm (*encode_single_bit*) that was discussed in the previous sections. The generated altered partition S_k^W is inserted into watermarked data set D_W. Statistics (X_{max}, X_{min}) are collected after each bit embedding and are used by the *get_optimal_threshold* algorithm to compute the optimal decoding threshold; these details will be discussed further in the following sections.

```
Algorithm:  embed_watermark
Input:  Data set D, Secret Key K_s, Number of partitions m, Watermark W = {b_0, ..., b_{l-1}}
Output:  Watermarked data set D_W, Optimal Decoding Threshold T*

 1.  D_W, X_max, X_min ← {}
 2.  S_0, ..., S_{m-1} ← get_partitions(D, K_s, m)
 3.  for each Partition S_k
 4.     i ← k mod l
 5.     S_k^W ← encode_single_bit(b_i, S_k, c, X_max, X_min)
 6.     insert S_k^W into D_W
 7.  T* ← get_optimal_thershold(X_max, X_min)
 8.  return D_W, T*
```

Figure 9: Watermark embedding algorithm

6 Decoding Threshold Evaluation

In the previous sections we discussed the bit encoding technique which embeds a watermark bit b_i in a partition S_i to generate a watermarked partition S_i^W. In this section, we discuss the bit decoding technique which is used to extract the embedded watermark bit b_i from the partition S_i^W. The bit decoding technique is based on an optimal threshold T^* that minimizes the probability of decoding error. The evaluation of such optimal threshold is discussed in this section.

Presented with the data partition S_i^W the bit decoding technique computes the hiding

function $\Theta_\Upsilon(S_i^W)$ and compares it to the optimal decoding threshold T^* to decode the embedded bit b_i. If $\Theta_\Upsilon(S_i^W)$ is greater than T^* then the decoded bit is 1 otherwise the decoded bit is 0. For example, using the hiding function described in Section 5.1 the decoding technique computes the normalized tail count of S_i^W by computing the reference ref and by counting the number of entries in S_i^W that are greater than ref. Then the computed normalized tail count is compared to T^*, see Figure 10. The decoding technique is simple; however, the value of the threshold T^* should be carefully calculated so as to minimize the *probability of bit decoding error* as will be discussed in this section.

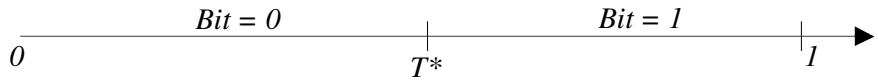


Figure 10: Threshold based decoding scheme.

The *probability of bit decoding error* is defined as the probability of an embedded bit decoded incorrectly. The decoding threshold T^* is selected such that it minimizes the probability of decoding error. The bit embedding stage discussed in Section 5.1 is based on the maximization or minimization of the tail count; these optimized hiding function values computed during the encoding stage are used to compute the optimum threshold T^*. The maximized hiding function values corresponding to $b_i's$ equal to 1 are stored in the set X_{max}. Similarly the minimized hiding function values are stored in X_{min}, (see algorithm in Figure 4).

Let P_{err}, P_0, and P_1 represent the probability of decoding error, the probability of encoding a bit = 0 and the probability of encoding a bit = 1 respectively. Furthermore, let b_e, b_d, and $f(x)$ represent the encoded bit, decoded bit, and a probability density function respectively. P_{err} is calculated as follows:

$$\begin{aligned} P_{err} &= P(b_d = 0, b_e = 1) + P(b_d = 1, b_e = 0) \\ &= P(b_d = 0|b_e = 1)P_1 + P(b_d = 1|b_e = 0)P_0 \\ &= P(x < T|b_e = 1)P_1 + P(x > T|b_e = 0)P_0 \\ &= P_1 \int_{-\infty}^{T} f(x|b_e = 1)dx + P_0 \int_{T}^{\infty} f(x|b_e = 0)dx \end{aligned}$$

To minimize the probability of decoding error (P_{err}) with respect to the threshold T we take the first order derivative of P_{err} with respect to T to locate the optimal threshold T^*, as follows:

$$\begin{aligned}\frac{\partial P_{err}}{\partial T} &= P_1 \frac{\partial}{\partial T} \int_{-\infty}^{T} f(x|b_e = 1)dx + P_0 \frac{\partial}{\partial T} \int_{T}^{\infty} f(x|b_e = 0)dx \\ &= P_1 f(T|b_e = 1) - P_0 f(T|b_e = 0)\end{aligned}$$

The distributions $f(x|b_e = 0)$ and $f(x|b_e = 1)$ are estimated from the statistics of the sets X_{min} and X_{max} respectively. From our experimental observations of X_{min} and X_{max} the distributions $f(x|b_e = 0)$ and $f(x|b_e = 1)$ pass the chi-square test of normality and thus can be estimated as Gaussian distributions $N(\mu_0, \sigma_0)$ and $N(\mu_1, \sigma_1)$ respectively. However, the following analysis can still be performed with other types of distributions. P_0 could be estimated by $\frac{|X_{min}|}{|X_{max}|+|X_{min}|}$ and $P_1 = 1 - P_0$. Substituting the Gaussian expressions for $f(x|b_e = 0)$ and $f(x|b_e = 1)$ the first order derivative of P_{err} is as follows:

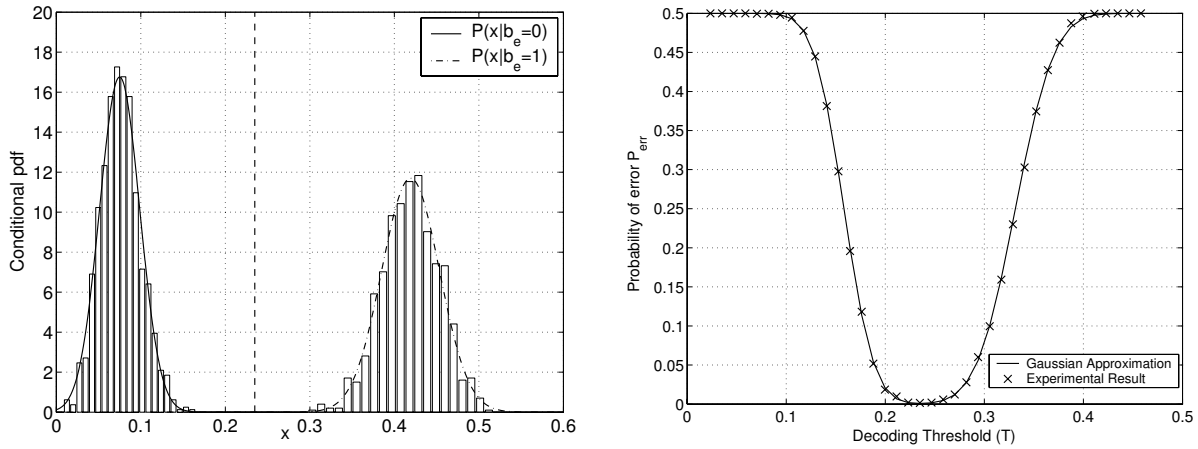
$$\frac{\partial P_{err}}{\partial T} = \frac{P_1}{\sigma_1\sqrt{2\pi}} exp(-\frac{(T-\mu_1)^2}{2\sigma_0^2}) - \frac{P_0}{\sigma_0\sqrt{2\pi}} exp(-\frac{(T-\mu_0)^2}{2\sigma_0^2})$$

By equating the first order derivative of P_{err} to zero we get the following quadratic equation the roots of which include the optimal threshold T^* that minimizes P_{err}. The second order derivative of P_{err} is evaluated at T^* to ensure that the second order necessary condition ($\frac{\partial^2 P_{err}(T^*)}{\partial T^2} > 0$) is met.

$$\frac{\sigma_0^2 - \sigma_1^2}{2\sigma_0^2\sigma_1^2} T^{*2} + \frac{\mu_0\sigma_1^2 - \mu_1\sigma_0^2}{\sigma_0^2\sigma_1^2} T^* + ln\Big(\frac{P_0\sigma_1}{P_1\sigma_0}\Big) + \frac{\mu_1^2\sigma_0^2 - \mu_0^2\sigma_1^2}{2\sigma_0^2\sigma_1^2} = 0$$

From the above analysis the selection of the optimal T^* is based on the collected output statistics of the watermark embedding algorithm. The optimal threshold T^* minimizes the probability of decoding error and thus enhances the strength of the embedded watermark by increasing the chances of successful decoding. To show the high dependency of the probability of decoding error and the choice of decoding threshold T^*, we conducted an experiment using real life[1] data with usability constraints of $\pm 10\%$ of the original data value. The histograms

[1]Description of such data is discussed in Section 9



(a) (b)

Figure 11: (a) Shows $f(x|b_e = 0)$, $f(x|b_e = 1)$ and the computed $T^* = 0.2349$. (b) Gaussian approximation and experimental values of P_{err} for different decoding threshold (T) values.

and the Gaussian estimates of X_{max} and X_{min} obtained from the experiment are reported in Figure 11(a). The optimal computed threshold T^* is indicated by the dotted vertical line. As we can see from Figure 11(a) the two distributions are far apart which is a direct result of using the competing objects for b_i equal to 1 and 0. Figure 11(b), shows the probability of decoding error for different values of the decoding threshold, which shows the presence of an optimal threshold that minimizes the probability of decoding error. Furthermore, Figure 11(b) shows both the Gaussian approximation and the experimental values of the probability of decoding error, which shows that the Gaussian approximation matches the experimental results.

The probability of decoding error is also dependent on the usability constraints. If the usability constraints are tight the amount of alterations to the data set D may not be enough for the watermark insertion. Figure 12 shows the result of an experiment where the minimum probability of decoding error was computed for data subject to different usability constraints. The overall watermark probability of decoding error is reduced by embedding the watermark multiple times in the data set, which is basically a repetition error correcting code.

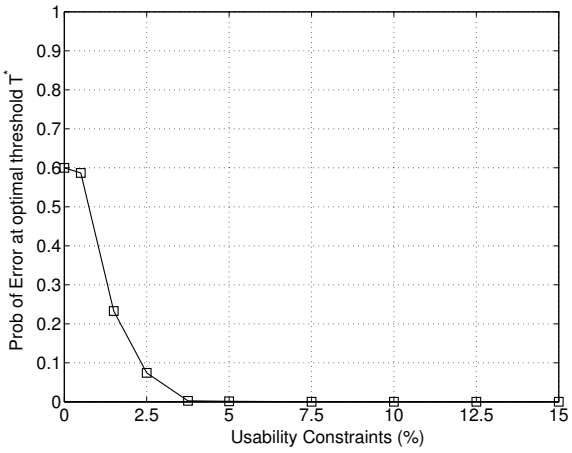


Figure 12: The minimum P_{err} at T^* for different usability constraints

7 Watermark Detection

In this section we discuss the watermark detection algorithm which extracts the embedded watermark using the secret parameters including K_s, m, ξ, c, T. The algorithm starts by generating the data partitions $\{S_0, \ldots, S_{m-1}\}$ using the watermarked data set D_W, the secret key K_s and the number of partitions m as input to the data partitioning algorithm discussed in Section 4. Each partition encodes a single watermark bit; to extract the embedded bit we use the threshold decoding scheme based on the optimal threshold T that minimizes the probability of decoding error as discussed in Section 6. If the partition size is smaller than ξ the bit is decoded as an erasure, otherwise it is decoded using the threshold scheme.

As the watermark $W = b_{l-1}, \ldots, b_0$ is embedded several times in the data set each watermark bit is extracted several times where for a bit b_i it is extracted from partition S_k where $k \ mod \ l = i$. The extracted bits are decoded using the majority voting technique which is used in the decoding of repetition error correcting codes. Each bit b_i is extracted $\frac{m}{l}$ times so it represents a $\lfloor \frac{m}{l} \rfloor$-fold repetition code [19]. The majority voting technique is illustrated by the example in Figure 13. The detailed algorithm used for watermark detection is reported in Figure 14.

In case of a relation with multiple attributes the watermark resilience can be increased by embedding the watermark in multiple attributes. This is a simple extension to the presented encoding and decoding techniques in which the watermark is embedded in each attributed

bits	5	4	3	2	1	0
w_0	0	1	1	0	1	0
w_1	x	0	1	0	1	x
w_2	0	x	0	0	1	0
w_3	0	1	1	x	0	0
w_{result}	**0**	**1**	**1**	**0**	**1**	**0**

Figure 13: An example illustrates the majority bit matching decoding algorithm for a watermark $W = 011010$, with $''\times''$ representing the erasures.

separately. For a δ attribute relation, the watermark bit is embedded in each of the δ columns separately using the bit embedding technique discussed in Section 5.1. The use of multiple attributes enables the multiple embedding of watermark bits δ times in each partition, such embedding can be considered as an inner δ-fold repetition code. For decoding purposes the statistics X_{max} and X_{min} are collected for each attribute separately. The optimal threshold is computed for each attribute using the collected statistics to minimize the probability of decoding error as discussed in Section 6. In the decoding phase, the watermark is extracted separately from each of the δ attributes using the discussed watermark detection algorithm, then majority voting is used to detect the final watermark.

8 Attacker Model

In this section we discuss the attacker model and the possible malicious attacks that can be performed. Assume, Alice is the owner of the data set D and has marked D by using a watermark W to generate a watermarked data set D_W. The attacker Mallory can perform several types of attacks in the hope of corrupting or even deleting the embedded watermark. A robust watermarking technique must be able to survive all such attacks.

We assume that Mallory has no access to the original data set D and does not know any of the secret information used in the embedding of the watermark, including the secret key

```
Algorithm: detect_watermark
Input: Watermarked data set D_W, m, c, ξ, K_s, T*, Watermark length l
Output: Detected watermark W_D

 1.   set ones[0,...,l-1] ← 0
 2.   set zeros[0,...,l-1] ← 0
 3.   S_0,...,S_{m-1} ← get_partitions(D_W, K_s, m)
 4.   for j = 0,...,m-1
 5.    if |S_j| ≥ ξ
 6.      i ← j mod l
 7.      value ← Θ(S_j, 0, c)
 8.     if value ≥ T*
 9.      ones[i] ← ones[i] + 1
 10.    else
 11.     zeros[i] ← zeros[i] + 1
 12.  for j = 0,...,l-1
 13.   if ones[j] > zeros[j]
 14.    W_D[j] ← 1
 15.   else if ones[j] < zeros[j]
 16.    W_D[j] ← 0
 17.   else
 18.    W_D[j] ← ×
 19.  return W_D
```

Figure 14: Watermark detection algorithm

K_s, the secret number of partitions m, the secret constant c, the optimization parameters and the optimal decoding threshold T^*. Given these assumptions Mallory cannot generate the data partitions $\{S_0, \ldots, S_{m-1}\}$ because this requires the knowledge of both the secret key K_s and the number of partitions m, thus Mallory cannot intentionally attack certain watermark bits. Moreover, any data manipulations executed by Mallory cannot be checked against the usability constraints because the original data set D is unknown. Under these assumptions Mallory is faced with the dilemma of trying to destroy the watermark and at the same time of not destroying the data. We classify the attacks preformed by Mallory into three types, namely *deletion, alteration and insertion* attacks.

Deletion Attack: Mallory deletes α tuples from the marked data set. If the tuples are randomly deleted, then on average each partition loses $\frac{\alpha}{m}$ tuples. The watermarking techniques available in literature rely on special tuples, referred to as marker tuples. Agrawal et al. [1] use marker tuples to locate the embedded bit and Sion et al. [23] use marker tuples to locate the start and end of data partitions. The embedded watermark is a stream of bits where

the marker tuples identify the boundaries between these bits in the stream. The successful deletion of marker tuples deletes these boundaries between the bits of the watermark stream, which makes such marker based watermarking techniques [1, 23] susceptible to watermark synchronization error. For example, using the watermarking technique presented by Sion et al. [23], Figure 15(a) shows an example partitioned data set and the corresponding majority voting map used to decode the embedded watermark. The marker tuples are represented by the shaded cells; these markers are used to identify the start and end of each partition. The embedded bit is noted in each partition; the embedded watermark is "101010". Now if Mallory successfully deletes the marker tuple controlling the first bit (b_0), this results in the deletion of the first bit, see Figure 15(b). The decoder, unaware of the deleted bit, will generate "×10101" instead, which is the result of decoding a shifted version of the embedded bits. This results in a watermark synchronization error at the decoder, see Figure 15(b). Moreover, the resynchronization of the watermark bit stream becomes more complicated in the presence of flipped bits due to other decoding errors. Thus the successful deletion of a single marker could result in a large number of errors in the decoding phase. To avoid watermark synchronization errors in marker based techniques the m marker tuples should be stored, as indicated by Sion et al. in [23]. Note that this violates the requirement that the watermark decoding is blinded.

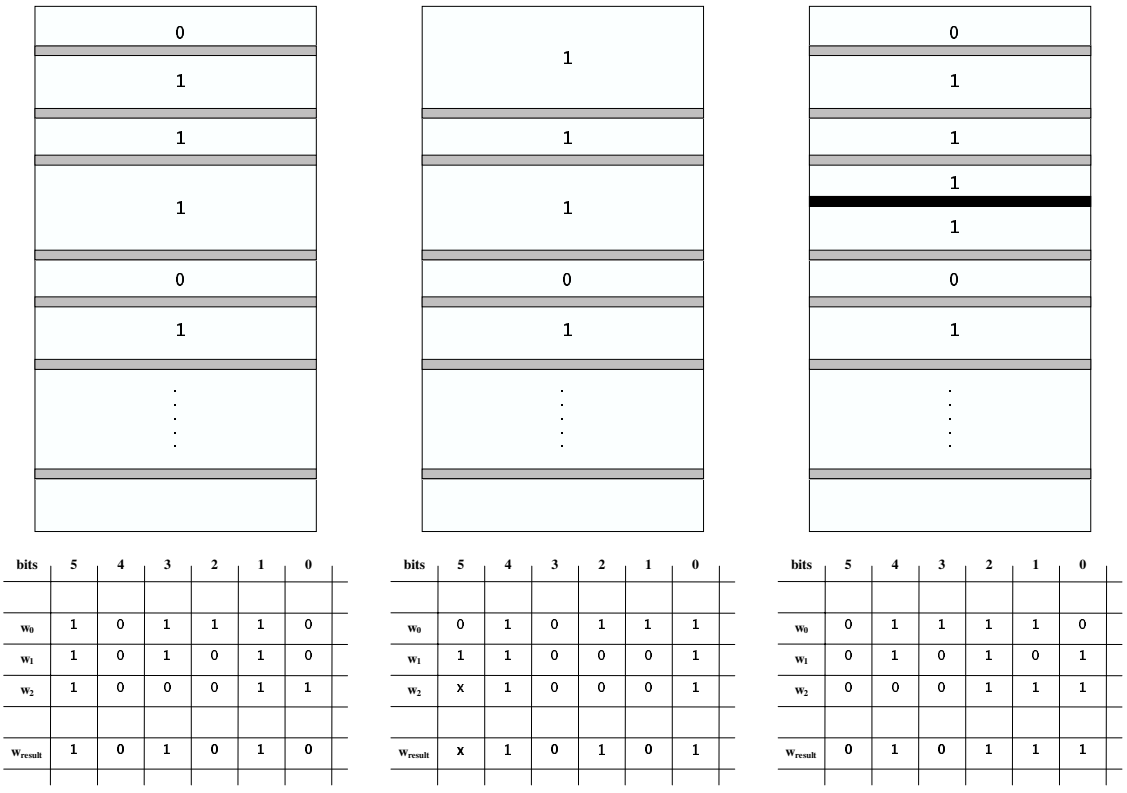
On the other hand, our partitioning technique is resilient to such synchronization errors as it does not rely on marker tuples to locate the partition limits; instead our partitioning technique assigns tuples to partitions based a different approach as discussed in Section 4. We also use erasures to indicate the loss of a bit due to insufficient partition size and thus to maintain synchronization and ensure that our technique is resilient to the watermark synchronization error.

Alteration Attack: In this attack Mallory alters the data value of α tuples. Here Mallory is faced with the challenge that altering the data may disturb the watermark; however at the same time Mallory does not have access to the original data set D, thus may easily violate the usability constraints and render the data useless. The alteration attack basically perturbs the data in hope of introducing errors in the embedded watermark bits. The attacker is trying to move the hiding function values from the left of the optimal threshold to the right and

vice versa. However, using the conflicting objectives in encoding the watermark bits, that is the maximizing the tail count for $b_i = 1$ and minimizing the tail count for $b_i = 0$, maximizes the distance between the hiding function values in both cases; thus it makes it more difficult for the attacker to alter the embedded bit. In addition, by the repeated embedding of the watermark and the use of majority voting technique discussed in Section 7 this attack can easily be mitigated.

Insertion Attack: Mallory decides to insert α tuples to the data set D_W hoping to perturb the embedded watermark. The insertion of new tuples acts as additive noise to the embedded watermark. However, the watermark embedding is not based on a single tuple and is based on a cumulative hiding function that operates on all the tuples in the partition. Thus the effect of adding tuples is a minor perturbation to the value of the hiding function and thus to the embedded watermark bit. Marker-based watermarking techniques may suffer badly from this attack because the addition of tuples may introduce new markers in the data set and thus lead to the addition of new bits in the embedded watermark sequence. Consequently, this results in watermark synchronization error. Using the example mentioned earlier, Figure 15(a) shows a partitioned data set and its corresponding majority voting map using the Sion et al. technique [23], where the embedded watermark is "101010". Now if Mallory successfully adds a marker tuple after the third marker tuple, this results in the addition of a new bit between (b_2) and (b_3); see Figure 15(c) where the black cell represents the added marker tuple. The decoder, unaware of the added bit, will generate "010111" instead, which is the result of decoding a shifted version of the embedded bits. This problem is further complicated in the presence of bit errors in the watermark stream. To ensure synchronization at the decoder the marker based watermarking techniques require the storage of the m marker tuples to ensure successful partitioning of the dataset in the presence of the insertion attack [23]. On the other hand, our partitioning algorithm is not dependent on special marker tuples which makes it resilient to such attack, and watermark synchronization is guaranteed during decoding.

The experimental results presented in Section 9 support the claims made about the resilience of our watermarking technique to all the above attacks.



(a) Watermarked dataset (b) After deletion attack (c) After insertion attack

Figure 15: Watermarked dataset subject to the deletion and insertion attacks repectively, and their corresponding majority voting maps. Grey shaded cells represent the original marker tuples and the black cells represent the added marker tuples.

9 Experimental Results

In this section we report the results of an extensive experimental study that analyzes the resilience of the proposed watermarking scheme to the attacks described in Section 8. All the experiments were performed on Intel Pentium IV CPU 3.2GHz with 512MB RAM. We use real-life data from a relatively small database that contains the daily power consumption rates of some customers over a period of one year. Such data sets are made available through the CIMEG[2] project. The database size is approximately 5 Megabytes; for testing purposes only a subset of the original data is used with 150000 tuples. We used $c = 75\%$, a 16 bit watermark, a minimum partition size $\xi = 10$, a number of partitions $m = 2048$, the data

[2]CIMEG: Consortium for the Intelligent Management of the Electric Power Grid. http://helios.ecn.purdue.edu/∼cimeg.

change was allowed within ±10%. The pattern search algorithm was used for the optimization. The optimal threshold was computed using the technique used in Section 6 to minimize the probability of decoding error. The watermarked data set was subject to different types of attacks including deletion, alteration, and addition attacks. The results were averaged over multiple runs. Similar results were obtained for both uniform and normally distributed synthetic data. We show that it is difficult for Mallory to remove or alter the watermark without destroying the data.

We assessed computation times and observed a polynomial behavior with respect to the input data size. Given the setup described above, with a local database we obtained an average of around 300 tuples/second for watermark embedding, while detection turned out to be at least approximately five times as fast. This occurs in the non-optimized, interpreted Java proof of concept implementation. We expect major orders of magnitude speedups in a real-life deployment version.

9.1 Deletion Attack

In this attack Mallory randomly drops α tuples from the watermarked data set, the watermark is then decoded and watermark loss is measured for different α values. Furthermore, in this test we compare our implementation with the marker based approach by Sion et al. [23]. Figure 16 shows the experimental results; they clearly show that our watermarking technique is resilient to the random deletion attack. Using our technique the watermark was successfully extracted with 100% accuracy even when over 85% of the tuples were deleted. On the other hand, the technique by Sion et al. badly deteriorates when only 10% of the tuples were deleted. We believe that high resilience of our watermarking technique is due to the marker-free data partitioning algorithm that is resilient to the watermark synchronization errors caused by the tuple deletion.

Because our technique is highly resilient to tuple deletion attacks the watermark can be retrieved from a small sample of the data. This important property combined with the high efficiency of our watermark detection algorithm makes it possible to develop tools able to effectively and efficiently search the web to detect illegal copies of data. We could think of an agent-based tool where the agent visits sites and selectively tests parts of the stored data

sets to check for ownership rights. Such a technique would only need inspect 15% of the data for successful watermark detection.

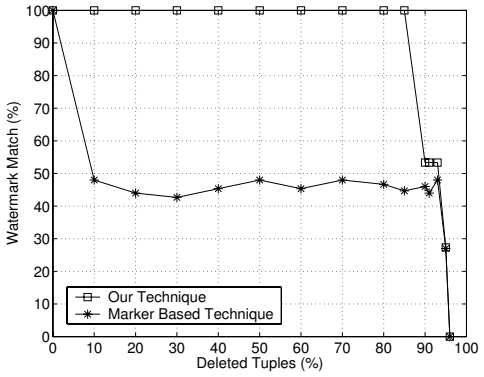
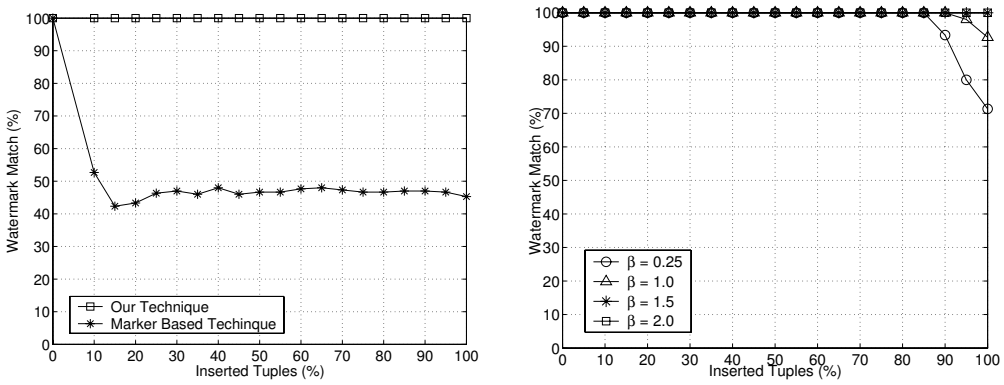


Figure 16: Resilience to deletion attack.

9.2 Insertion Attack

In this experiment Mallory attempts to add a number α of tuples hoping to weaken the embedded watermark. However, by adding tuples to the current data Mallory is adding meaningless data to the current data. Mallory could simply generate the new tuples by replicating values in randomly selected existing tuples; we refer to such attack as the α-selected insertion attack. Mallory could randomly generate the tuple values by generating random data from the range $(\mu_{D_W} - \beta\sigma_{D_W}, \mu_{D_W} + \beta\sigma_{D_W})$, where μ_{D_W} and σ_{D_W} are the mean and standard deviation of the data set D_W respectively. We refer to such attack as the (α, β)-insertion attack. Figure 17(a) shows a comparison between our approach and the marker based approach by Sion et al. The comparison shows that our technique is resilient to α-selected attack even when α is up to 100% of the data set size. Figure 17(b) shows the resilience of our watermarking technique to (α, β)-insertion attack, where the watermark was recovered with 100% accuracy even when up to 85% of the data set size tuples were inserted.



(a) α-selected insertion attack (b) (α, β)-insertion attack

Figure 17: (a) Resilience to α-selected insertion attack. (b) Resilience to (α, β)-insertion attack

9.3 Alteration Attack

We tested our watermarking technique against two types of alteration attacks namely the fixed and the random (α, β) alter attacks. In the fixed-(α, β) alter attack Mallory randomly selects α tuples and alters them by multiplying $\frac{\alpha}{2}$ tuples by exactly $(1 + \beta)$ and the other $\frac{\alpha}{2}$ tuples by $(1 - \beta)$. In this attack, the value of β is fixed. In the random-(α, β) alter attack α tuples are randomly selected; $\frac{\alpha}{2}$ tuples are then multiplied by $(1 + x)$ and the other $\frac{\alpha}{2}$ tuples by $(1 - x)$, where x is a uniform random variable in the range $[0, \beta]$.

Figures 18(a)(b)(c) show the behavior of our watermarking technique subject to the fixed-(α, β) alter attack. As we can see from Figure 18(a) the watermark is decoded with 100% accuracy even when 100% of the tuples are altered by $\beta > 10\%$. This shows the strong resilience of our watermarking technique to fixed alteration attacks. Furthermore, Figure 18(b) shows the number of corrupted tuples as the attack proceeds. Tuples that exceed the usability constraints are referred to as corrupted tuples. Figure 18(b) shows that after $\beta > 10\%$ a sudden increase in the number of corrupted tuples; such an increase is due to the usability constraints used in this experiment, which are set to $\pm 10\%$. Figure 18(c) is a clear description of the dilemma that the attacker is facing. The dotted lines show the number of corrupted tuples, while the solid lines are represent the detected watermark accuracy. By increasing β the attacker is able to corrupt the watermark to 85% accuracy, however, at the same time 50% of the tuples are corrupted. Similar results were experienced

for the random-(α, β) attack which are shown in Figures 18(d)(e)(f).

Experiments performed at lower usability constraints still showed similar resilience trends of the watermark encoding and decoding when subject to above attacks. Table 1 shows a comparison between our technique and Sion et al. technique based on the different watermark attacks and main characteristics of each technique.

Table 1: Comparison between our technique and the technique by Sion et al. [23].

	Our Technique	Sion et al. Technique
Deletion Attack	Resilient to random tuple deletion attack; 100% watermark accuracy even when more than 85% of the tuples are deleted.	Not resilient to random tuple deletion attack; watermark accuracy deteriorates to 50% when only 10% of the tuples are deleted.
Insertion Attack	Resilient to random tuple insertion attacks; 100% watermark accuracy even when more than 100% of the original number of tuples is inserted in the α-insertion attack and similar watermark accuracy when subject to the (α, β)-insertion attacks.	Not resilient to random tuple insertion attacks; watermark accuracy deteriorates to 50% when only 10% of the tuples are inserted.
Alteration Attack	Resilient to random tuple alteration; 100% watermark accuracy even when 100% of the tuples are altered by $\beta >$ 10% and $\beta >$ 20% for the fixed-(α, β) and random-(α, β) attacks respectively. The watermark embedding technique exploits the feasible alteration space by solving an optimization problem to enforce the competing objectives based on the bit to be inserted. Furthermore, decoding threshold is computed based on the embedding statistics to maximize the probability of decoding error.	It is not clear how the bit embedding is performed; no systematic alteration scheme is defined that investigates the feasible embedding space. Thresholds are not based on embedding statistics and are chosen arbitrarily by the user without any optimality criteria.
Synchronization Error	Not vulnerable to such error, because the technique does not require special marker tuples for the correct partition reconstruction.	Highly vulnerable to such error, due to the dependency on special marker tuples to locate partitions. Requires the storage of all m marker tuples for the correct partition reconstruction.
Decoding Threshold	Uses an optimal threshold T^* that minimizes the probability of decoding error.	Uses two decoding thresholds which are arbitrarily decided by the user without any optimality criteria.

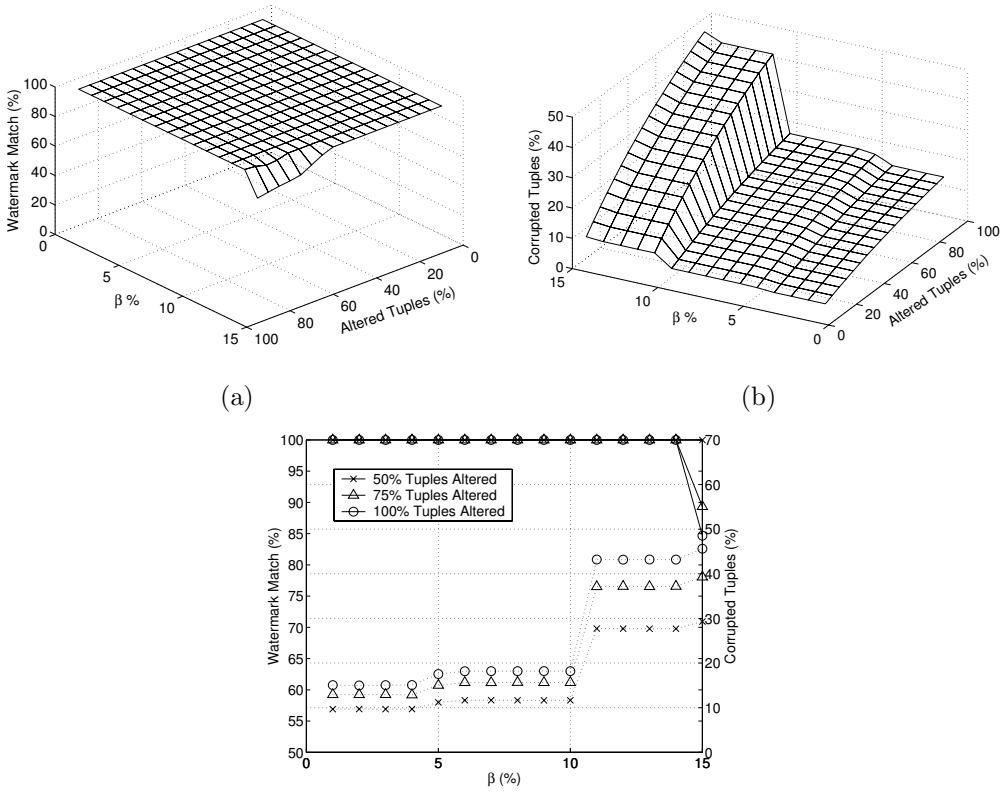


Figure 18: Resilience to fixed-(α, β) and random-(α, β) alter attacks.

10 Conclusion

In this paper, we have presented a resilient watermarking technique for relational data that embeds watermark bits in the data statistics. The watermarking problem was formulated as a constrained optimization problem, that maximizes or minimizes a hiding function based on the bit to be embedded. Genetic algorithm and pattern search techniques were employed to solve the proposed optimization problem and to handle the constraints. Furthermore, we presented a data partitioning technique that does not depend on special marker tuples to locate the partitions and proved its resilience to watermark synchronization errors. We developed an efficient threshold-based technique for watermark detection that is based on an optimal threshold that minimizes the probability of decoding error. The watermark resilience was improved by the repeated embedding of the watermark and using majority voting technique in the watermark decoding phase. Moreover, the watermark resilience was improved by using multiple attributes.

A proof of concept implementation of our watermarking technique was used to conduct experiments using both synthetic and real-world data. A comparison our watermarking technique with previously-posed techniques techniques shows the superiority of our technique to deletion, alteration and insertion attacks.

References

[1] R. Agrawal and J. Kiernan. Watermarking Relational Databases. In *Proceedings of 28th International Conference on Very Large Data Bases*, Hong Kong, China, 2002.

[2] M. Atallah and S. Lonardi. Authentication of LZ-77 Compressed Data. In *Proceedings of the ACM Symposium on Applied Computing*, Florida, USA, 2003.

[3] M. Atallah, V. Raskin, C. Hempelman, M. Karahan, R. Sion, K. Triezenberg, and U. Topkara. Natural Language Watermarking and Tamperproofing. In *Proceedings of the Fifth International Information Hiding Workshop*, Florida, USA, 2002.

[4] G. Box. Evolutionary Operation: A Method for Increasing Industrial Productivity. *Applied Statistics*, 6(2):81–101, 1957.

[5] E. Chong and S. $\dot{Z}$ak. *An Introduction to Optimization*. John Wiley & Sons, New York, 2001.

[6] D. Coley. *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific, 1999.

[7] C. Collberg and C. Thomborson. Software Watermarking: Models and Dynamic Embeddings. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, San Antonio, TX, January 1999. ACM.

[8] I. Cox, J. Bloom, and M. Miller. *Digital Watermarking*. Morgan Kaufmann, 2001.

[9] E. Dolan, R. Lewis, and V. Torczon. On the Local Convergence of Pattern Search. *SIAM Journal on Optimization*, 14(2):567–583, 2003.

[10] D. Goldberg. *Genetic Algorithm in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[11] D. Gross-Amblard. Query-Preserving Watermarking of Relational Databases and XML Documents. In *PODS '03: Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 191–201. ACM Press, 2003.

[12] F. Hartung and M.Kutter. Multimedia Watermarking Techniques. *Proceedings of the IEEE*, 87(7):1079–1107, July 1999.

[13] J. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, 1992.

[14] R. Hooke and T. Jeeves. Direct Search Solution of Numerical and Statistical Problems. *Journal of the Association for Computing Machinery*, 8(2):212–229, 1961.

[15] G. Langelaar, I. Setyawan, and R. Lagendijk. Watermarking Digital Image and Video Data: A State-of-the-Art Overview. *IEEE Signal Processing Magazine*, 17(5):20–46, September 2000.

[16] R. Lewis and V. Torczon. Pattern Search Methods for Linearly Constrained Minimization. *SIAM Journal on Optimization*, 10(3):917–941, 2000.

[17] Y. Li, H. Guo, and S. Jajodia. Tamper Detection and Localization for Categorical Data Using Fragile Watermarks. In *DRM '04: Proceedings of the 4th ACM Workshop on Digital Rights Management*, pages 73–82. ACM Press, 2004.

[18] Y. Li, V. Swarup, and S. Jajodia. Fingerprinting Relational Databases: Schemes and Specialties. *IEEE Transactions on Dependable and Secure Computing*, 02(1):34–45, Jan-Mar 2005.

[19] R. Morelos-Zaragoza. *The Art of Error Correcting Coding*. John Wiley & Sons, 2002.

[20] J. Nocedal and S. Wright. *Numerical Optimization*. Prentice Hall, 1999.

[21] F. Petitcolas, R. Anderson, and M. Kuhn. Attacks on Copyright Marking Systems. *Lecture Notes in Computer Science*, 1525:218–238, April 1998.

[22] B. Schneier. *Applied Cryptography*. John Wiley, 1996.

[23] R. Sion, M. Atallah, and S. Prabhakar. Rights Protection for Relational Data. *IEEE Transactions on Knowledge and Data Engineering*, 16(6), June 2004.

[24] M. Swanson, M. Kobayashi, and A. Tewfik. Multimedia Data-Embedding and Watermarking Technologies. *Proceedings of the IEEE*, 86:1064–1087, June 1998.

[25] L. Vaas. Putting a Stop to Database Piracy. *eWEEK, Enterprise News and Revs.*, Sept 2003.

[26] R. Wolfgang, C. Podilchuk, and E. Delp. Perceptual Watermarks for Digital Images and Video. *Proceedings of the IEEE*, 87:1108–1126, July 1999.