Watermarking Non-numerical Databases

Agusti Solanas and Josep Domingo-Ferrer

Rovira i Virgili University of Tarragona,
Dept. of Computer Engineering and Maths,
Av. Països Catalans, 26, E-43007 Tarragona, Catalonia
{agusti.solanas, josep.domingo}@urv.net

Abstract. This paper presents a new watermarking method for protecting non-numerical databases. The proposed watermarking system allows the data owner to define a similarity function in order to reduce the distortion caused by watermark embedding while, at the same time, reducing the number of element modifications needed by the embedding process. A mathematical analysis is provided to justify the robustness of the mark against different types of malicious attacks. The usefulness of this extensible and robust method is illustrated by describing some application domains and examples.

Keywords: Private watermarking, Data hiding, Database security, Non-numerical databases.

1 Introduction

Watermarking systems have been widely studied for intellectual property protection (IPR) of multimedia data [1, 2, 3, 4]. It is common for watermarking systems to make use of well-known cryptographic techniques such as digital signatures [5] or signal processing techniques such as phase modulation [6] or spread spectrum [7, 8]. Most methods designed for multimedia data rely on the perceptual limitations of humans, *e.g.* our inability to distinguish between very similar colors or sounds [9, 10]. However, over the last few years, researchers have realized that these limitations cannot be exploited when trying to protect other kinds of data, such as software [11] and databases [12].

Recent contributions on database IPR [12, 13] have clarified the main differences and singularities of typical database content (alphanumeric data) vs multimedia. Databases have very little redundancy as compared with multimedia data and this fact makes it very difficult to find enough bandwidth in which to embed the watermark. Moreover, databases can contain non-numerical or categorical data like city names, drug names, hair colors, etc. Such non-numerical data cannot be smoothly marked by increasing or reducing their value or modifying some of their bits. A non-numerical element must be completely altered in order to embed a mark and this limitation represents a great challenge that is addressed in this paper.

Some authors have tackled this problem before ([13]), but their proposals have some shortcomings regarding data distortion and watermark length. We present here a watermarking system s for non-numerical data that: i) minimizes the number of changes needed to embed the mark and; ii) reduces the distortion produced by the mark by allowing the user to customize the watermark embedding system through the definition of a similarity function related to the data.

The rest of the paper is organized as follows. In Section 2, we describe our model in detail and specify the notation and the assumptions used. Section 3 presents our watermarking system. Section 4 analyzes its properties of robustness against different types of malicious attacks. In Section 5, the usefulness of the similarity function is justified by presenting some examples and application domains. Finally, conclusions are listed in Section 6.

2 The Model

We first state a model to which all subsequent solutions refer. We consider the following elements to define our model: i) the data: what are we working with? ii) our target: what do we want to achieve? and, iii) the enemy: what kind of attacks are likely to be used by intruders to destroy our watermark? Moreover, at the end of this section, some brief comments about notation and assumptions are made.

2.1 The Data

The data we work with consist of a finite number T of non-numerical and discrete elements E stored in a database. It is assumed that all elements in E are known and can be ranked (*e.g.* alphabetic ordering would do). We easily find lots of examples of this kind of data: for example, city names, carmaker names or drug names. The main characteristic of non-numerical data is that they cannot be smoothly modified, this is, any change is a complete change of value. If the data we have are so critical that they cannot be modified at all then no watermark system can be applied because –by its very nature– a watermarking system has to change some elements in order to embed the mark. In this paper, we assume that the data can be modified to a limited extent.

We consider a database organized in relations R, where each relation can be viewed as the union of a primary key $R.Pk$ and one or more attributes A. The proposed watermarking system can be applied to any relation in which modification of the primary key is not allowed: as argued in [12], we assume that modification of the primary key results in an unacceptable loss of information.

Without loss of generality, we consider a relation R formed by the union of a primary key $R.Pk$ and a single attribute A, that is

$$R \rightarrow \{R.Pk, A\}$$

2.2 Our Goal

We want to be able to hide a mark into the data without causing unacceptable data modifications while, at the same time, making the mark as robust as possible against different types of malicious attacks. This challenging problem can be broken down into two goals:

– Minimize the number of changes caused by the watermark embedding system, while maintaining its robustness;
– Allow the user to extend the system by defining a specific similarity function for minimizing the impact of the changes.

2.3 The Intruder

The intruder wants to get hold of data D and, after a malicious modification, sell an unmarked version D' for which the owners of D are unable to show their intellectual property rights. Of course, data utility for D' and D should be similar for the attack to make sense (otherwise either D' is useful but still carries the mark or D' has become useless as a result of mark removal attacks). To destroy the mark embedded in D, the intruder can use different types of malicious attacks.

– **Horizontal sampling:** In this attack, the intruder randomly selects a set of tuples and discards the rest. Thus, if the mark depends on any kind of spatial relation, it will be lost. The mark has to be resilient to this attack, so that the intruder is forced to reduce the number of tuples selected from D to an extent such that the resulting D' is no longer very useful.
– **Vertical sampling:** Similar to the previous attack, this one is based on randomly selecting attributes in a tuple that will be erased. In order to resist this attack, the watermark should be recoverable from a single attribute.
– **Perturbation of randomly chosen elements:** Let a data element be the value taken by a specific attribute in a specific tuple. Perturbing a randomly selected subset of data elements is a very common attack for numerical data. The main difference when applying this attack to non-numerical data is that any modification is likely to be significant; it is not easy for the intruder to perturb non-numerical data without substantial utility loss. In order for the mark to be resilient against this attack, it should resist as many element perturbations as needed to render D' useless.
– **Horizontal and vertical re-ordering:** This attack consists of swapping pairs of tuples or attributes without modifying them. If the mark has to resist this attack, it cannot be based on any relative spatial position of the data elements.

2.4 Notation and Assumptions

In this section, we briefly enumerate some assumptions and notation used in the remainder of this article (see also Table 1).

– Hash function H. Secure one-way hash functions such as SHA [14] are used in our algorithm. Given a value z', we assume it is computationally unfeasible to find z such that $H(z) = z'$.

Table 1. Table of symbols

Symbols	Meaning
E	Set of all non-numerical elements
G	A pseudo-random generator
N	Number of markable elements
n	Number of actually marked elements
K_1	Secret key used for embedding the mark
K_2	Secret key used for computing the mark
P	One-dimensional table of products $x_i s_i$
R	A relation in the database
sf	A similarity function
T	Total number of elements in the data
t	A tuple in the relation
V	Binary vector of selected elements
γ	Fraction of selected elements for embedding

– Pseudo-random number generator G. This generator must be seeded with a combination of information taken from the data and a secret key only known to the owner of the data. We assume that the pseudo-random generator outputs numbers that are uniformly distributed between 0 and 1 Once the pseudo-random generator is initialized, a new number is obtained by using the *next(G)* function.
– Least significant bit $lsb(e)$. This function returns the value of the least significant bit of the elements $e \in E$ and is mainly used in the mark recovery process.

3 Our Watermarking System

The proposed watermarking system can be subdivided into two subsystems: embedding and recovery.

3.1 The Embedding Subsystem

Embedding faces three main problems. First, it is necessary to decide where the watermark has to be hidden, that is, the elements of the attribute A in the relation R which will be considered candidates for a modification. Second, a similarity function *df* can be defined by the user in order to minimize the impact of embedding a watermark W into the data D and the watermark embedding system must allow the user to do so. Third, the watermark W must be computed and embedded into D, while meeting the previous restrictions.

Selection of the embedding positions. Similarly to [12] and [13], we make the assumption that a primary key $r.P_k$ exists in the relation which cannot be modified without causing unacceptable damage to data. We want the selected elements to be picked independently of their relative position in the relation.

Algorithm 1. Selection of the elements to be modified

```
1)   function $GetElementsToModify(K_1, R)$ return $\vec{V}$
2)       for each tuple $t \in R$ do
3)           seed $G$ with $r.P_k||K_1$
4)           if $next(G) \leq \gamma$ then
5)               $V_t = 1$
6)           else
7)               $V_t = 0$
8)           end if
9)       end for
10)      return $\vec{V}$
11)  end function $GetElementsToModify$
```

To that end, we use the primary key that uniquely addresses an element. In order to make this process secure, we use a secret key K_1 and we concatenate it with the primary key $r.P_k$ to obtain a value that is used to seed a pseudo-random generator G. The data elements for embedding are selected by using the pseudo-random generator as described in Algorithm 1.

After running Algorithm 1, a one-dimensional table $\vec{V}$ is obtained. The size of this table equals the number T of tuples in the relation. Each position of $\vec{V}$ contains a 0 or a 1 representing the selection result. All positions set to 1 represent the tuples in the relation that are selected for being modified in order to embed the mark.

In order to control the impact of watermark embedding, the relation between the selected elements and the total number of elements T is controlled through the γ parameter. The expected number of selected elements is $N = \gamma T$. Thus, if $\gamma = 0.25$ then 25% of tuples can be expected to be selected for being marked.

The similarity function. When a mark has to be hidden into numerical data, numerical data elements can be smoothly modified by slightly increasing or decreasing their values. On the contrary, hiding a mark into a non-numerical data element is often not smooth, as it implies substituting a categorical value for another. Previous approaches [13] assume that the replacement of a categorical value by another introduces the same distortion into the data independently of the new categorical value that replaces the original one. Even if we agree that changing the category of a non-numerical element is an important modification, we claim that the amount of distortion caused by this replacement depends on the similarity between the original category and the replacement category. In order to minimize the impact of watermark embedding on the data, we propose to resort to a ***user-defined*** similarity function, $sf(e_1, e_2) \rightarrow [0, 1]$

Given two elements e_1 and e_2, the similarity function returns a similarity value in $[0, 1]$. A 0 similarity is interpreted as "very different" and a 1 similarity as "very similar". Using such a similarity function, the distortion produced by swapping two data elements can be quantified and minimized. In Section 5 some example similarity functions applied to different domains are described.

Hiding the mark. The last step in the embedding process consists of: i) finding the elements that will replace the original ones in order to hide the watermark; ii) carrying out the replacement. This process can be denoted as:

$$Embed(R, K_2, sf, M) \rightarrow R'$$

To hide the watermark in the relation R we need: i) a secret key K_2 different from the one used to select the embedding position [1]; ii) a similarity function sf to minimize the impact of watermark embedding (optionally defined by the user); and iii) a security parameter M.

Once the elements that will be modified are selected using Algorithm 1, we specify the constraint below to be met by the elements that will replace the original ones:

$$\sum_{i=0}^{N} s_i x_i \geq M \tag{1}$$

where: $\overrightarrow{X} = \{x_i\}$ are pseudo-random numbers uniformly distributed in $[-\lambda, \lambda]$, where λ is a robustness parameter; $S = \{s_i\}$ are the least significant bits of the replacement data elements expressed as integers in $\{-1, 1\}$; M is a user-definable security parameter that determines the robustness and the impact of the mark. In the next paragraphs, details are given on the computation of x_i and s_i.

Computation of the values $\overrightarrow{X}$. A value x_i is computed for each selected tuple. that is, for indexes i such that $V_i = 1$ (in terms of Algorithm 1). This computation is performed by using a secret key K_2 and the primary relation key $R.Pk$ [2] to seed a pseudo-random number generator G. Then a set of N pseudo-random numbers are obtained using G and they are scaled in $[-\lambda, \lambda]$. In other words, we use a hash function that receives the concatenation of the primary key and a secret key K_2 as an input parameter and returns a number in $[-\lambda, \lambda]$.

$$H(R.Pk|K_2) \rightarrow [-\lambda, \lambda]$$

We require G to be such that $H(\cdot)$ is a secure one-way hash function: inferring the value of $R.Pk|K_2$ from $H(R.Pk|K_2)$ should be infeasible.

Selection of the values $\overrightarrow{S}$. Once the values $\overrightarrow{X} = \{x_i\}$ are fixed, we must determine values s_i satisfying Constraint (1). We want to minimize the impact of mark embedding on data, which translates to reducing the number and magnitude of changes to be made.

We initialize each s_i with the least significant bit $lsb(e_i)$ of the original element e_i to be marked. Specifically,

[1] It is possible to compute the watermark by using only one secret key, but we prefer to use two keys in order to avoid the risk of correlations between the generated pseudo-random numbers[13].

[2] Since we assume that the primary key cannot be modified, the values of X are only obtainable by the data owner and cannot be modified.

Algorithm 2. Computation of $\overrightarrow{S}$

```
1)  procedure $SetSElements(\overrightarrow{S}, \overrightarrow{X}, M)$
2)      $P = ComputeProducts(\overrightarrow{S}, \overrightarrow{X})$
3)      $SortInIncreasingOrder(\overrightarrow{P})$
4)      while $\hat{M} < M$ do
5)          $i = ObtainIndexOfMostNegativeProduct(\overrightarrow{P})$
6)          $Swap(\overrightarrow{S}, i)$
7)          $RemoveIFromP(\overrightarrow{P}, i)$
8)          $\hat{M} = ComputeMark(\overrightarrow{S}, \overrightarrow{X})$
9)      endwhile
10) end procedure $SetSElements$
```

$$s_i = \begin{cases} -1 & \text{if } lsb(e_i) = 0 \\ 1 & \text{if } lsb(e_i) = 1 \end{cases}$$

After the initialization of $\overrightarrow{S}$, we compute

$$\sum_{i=0}^{N} s_i x_i = \hat{M}$$

Next, if $\hat{M} > M$ then Constraint (1) is met; so, we take $\hat{M}$ as M and no changes are introduced to the data (minimum distortion). When $\hat{M} < M$ then it is necessary to change some values of $\overrightarrow{S}$ in order to satisfy the embedding constraint. The way in which the values of $\overrightarrow{S}$ are changed is described in Algorithm 2. The algorithms called within Algorithm 2 are described in the remainder of the Section (Algorithms 3 and 4).

Initially, the products $\overrightarrow{P}$ of each x_i and s_i are computed in order to find the impact of the i-th element in the computation of $\hat{M}$. Then the one-dimensional table $\overrightarrow{P}$ is sorted in order of increasing magnitude. To satisfy Constraint (1) with the minimum number of changes, the least significant bit s_i of the most negative product is inverted; in that way, with a single bit inversion, we obtain a maximum increase of $\hat{M}$. To perform the inversion of s_i, the element e_i in the relation R must be replaced by the most similar element of E with a different least significant bit (see Algorithm 4). A similarity function sf is used to determine the most similar element to e_i. This similarity function should be defined by the owner of the database. However, it is optional and when it is not given, a simple alphabetical comparison could be made to obtain a similarity value.

Note 1 (On the role of λ). Note that the magnitude of the most negative product is related to the range $[-\lambda, \lambda]$ where the x_i are chosen. Thus, a larger λ will reduce the expected number of iterations of Algorithm 2 and therefore the expected

Algorithm 3. Computation of the watermark from a given $\overrightarrow{S}$ and $\overrightarrow{X}$

```
1)  function ComputeMark($\overrightarrow{S}, \overrightarrow{X}$) return $\hat{M}$
2)      $\hat{M} = 0$
3)      For $i = 1$ to $N$ do
4)          $\hat{M} = \hat{M} + s_i x_i$
5)      end for
6)      return $\hat{M}$
7)  end function ComputeMark
```

Algorithm 4. Replacement of an original element by its most similar substitute

```
1)  procedure Replace($\overrightarrow{S}, i$)
2)      $lsb = 0$      //Initialize the least significant bit
3)      if $s_i == 1$ then      //change the $s_i$ value
4)          $s_i = -1$
5)          $lsb = 0$
6)      else if $s_i == -1$ then
7)          $s_i = 1$
8)          $lsb = 1$
9)      endif
10)     newElement = getMostSimilarElement($e_i, lsb, sf$)
11)     $e_i$ = newElement
12) end procedure Replace
```

number $n \leq N$ of actually marked elements. The drawback of taking λ too big is that, the larger λ, the less elements will carry the mark, so that we gain imperceptibility but lose robustness. Therefore, λ should be chosen so that the resulting n is not much smaller than the number of markable elements N.

It is easy to see that, following Algorithm 2, the *number* of changes made to satisfy Constraint (1) is minimal for a fixed value of λ. Using a similarity function sf capturing the semantics of data allows each individual change (replacement) to be minimal in *magnitude*; this is done by the *getMostSimilarElement* user-definable function called in Algorithm 4. The result is minimal data alteration in watermark embedding.

3.2 The Recovery Subsystem

Watermark recovery must determine whether a watermark is embedded in a relation. To perform this task, the recovery subsystem receives as parameters: the relation $\hat{R}$ which presumably embeds the mark and may have been attacked; the security parameter M; and the secret keys K_1 and K_2 only known to the data owner. Thus, this subsystem can be denoted as:

$$Recovery(\hat{R}, K_1, K_2, M) \rightarrow (yes/no)$$

Similarly to the embedding process, it is first necessary to obtain the marked elements using Algorithm 1. Note that is not necessary to know the original R in order to apply Algorithm 1 because the primary key of R is supposed to remain unmodified in $\hat{R}$. Once the marked elements are located, the value of each x_i is computed in the same way as in the embedding process, using the secret key K_2. Finally, the value of each s'_i is obtained from the least significant bit of the elements by applying the $lsb()$ function. Note that, in general it can happen that $s'_i \neq s_i$, as a result of accidental/intentional distortion during the data lifecycle. Once all the above information is recovered, the recovery subsystem computes $\sum_{i=0}^{N} s'_i x_i = \hat{M}'$

The recovery subsystem decides that the data contain a watermark when $\hat{M}' \geq \frac{M}{2}$. Otherwise, no mark is recovered.

4 Robustness Analysis

The proposed watermarking system is robust against random alterations and vertical and horizontal sampling. The intruder can perform a broad range of different malicious attacks. We now describe how the watermark embedded by our watermarking system tolerates these attacks.

- **Vertical sampling:** Our system can be applied to any relation R with at least a primary key and an attribute A. The inserted mark does not depend on any relationship between attributes and can be embedded individually in as many attributes as desired. Thus, the attack based on selecting some attributes and erasing the rest has no effect on our watermark because, at least, one marked attribute remains.
- **Horizontal and vertical re-ordering:** The horizontal re-ordering attack consists of swapping the positions of pairs of tuples without modifying them. Our watermarking system is not vulnerable to this kind of attack because the relative position of the elements in the relation R is not used to determine whether they are marked.
  Similarly, vertical re-ordering consists in swapping the positions of pairs of attributes without modifying them. As argued in the previous sections, our method is applied to an attribute and it does not depend on its relative position in the relation.
- **Perturbation of randomly chosen elements:** The recovery system detects the existence of a mark when $\hat{M}' \geq \frac{M}{2}$. The intruder wants to destroy the mark by modifying the value of randomly chosen elements. If the intruder is able to destroy enough marked elements then the mark will not be recovered. Thus, a natural strategy that leads to arbitrary reduction of the probability of mark destruction is to increase the number n of marked elements, which can be done by decreasing λ and increasing the number N of markable elements.
- **Horizontal sampling:** This malicious attack is based on a random selection of a fraction of tuples of a relation R. This usually tricky attack is not

effective against our method because the amount of non-selected tuples has to be very big compared with the number of tuples modified by the watermark in order to destroy it. Considering that a non-selected tuple is like an altered tuple, the analysis is analogous to the one above for perturbation attack.

4.1 A Toy Example

To illustrate the robustness of our model against perturbation of randomly chosen elements, we take a toy database that consists of a relation R with 250 tuples or elements. We embed a mark that modifies $n \leq N = 25$ elements (10%) and a mark that modifies $n \leq N = 30$ elements (12%). We assume that, for data to stay useful, up to 30% of elements can be modified. This is up to 75 element modifications, about three times the number of modifications caused by watermark embedding. Also, we have chosen a value for M such that the mark is destroyed if more than half of the N markable elements are modified. If the intruder modifies P elements among the total T elements, the probability that she destroys the mark by randomly hitting more than $N/2$ markable elements is

$$P[Destruction] = \frac{\sum_{i=0}^{N/2} \binom{N}{\frac{N}{2}+i}\binom{T-N}{P-(\frac{N}{2}+i)}}{\binom{T}{P}}$$

Table 2 shows the probability of the intruder destroying the mark by modifying less than 30% of the elements, that is, up to $P = 75$ elements. It can be seen that, even if the intruder modifies three times as many elements as those modified by the mark embedding algorithm (75 vs 25) her probability of success is no more than 0.25.

Table 2. Destruction of the mark in the toy example

Modified elements	P(Destroy |25 marked elem.)	P(Destroy |30 marked elem.)
25	0.0000087	≈ 0
30	0.000077	0.0000009
35	0.000422	0.00001
40	0.0016	0.00007
45	0.0051	0.00034
50	0.0132	0.0012
55	0.029	0.0038
60	0.057	0.0099
65	0.102	0.022
70	0.16	0.045
75	0.25	0.081

5 Application Domains

The main application of the presented watermarking system is to protect non-numerical (*i.e.* categorical) databases from being copied and re-sold by an intruder. These databases are sold to companies which want to obtain information from the data, usually by applying data mining techniques.

We next illustrate how a similarity function could be defined in a couple of specific example databases.

5.1 Drugs Database

Imagine that we have a drugs database storing information about the drugs taken by a set of patients. We may have information about the composition of each drug and we can determine the similarity between them. In this case of study, the similarity function defined by the user may be based in the next considerations:

Similarity function: Coincidences in the number and proportion of components in a given drug. Following this similarity function we can replace the element "ASPIRIN 250g" by the generic element "acetylsalicylic acid 250g" without any distortion. Note that the element "acetylsalicylic acid 250g" must be in the database in order to be considered for replacing the element "ASPIRIN 250g". Similarly, if we try to replace "ASPIRIN 250g" by "CHLORHEXIDINE GLUCONATE 1g" the similarity function must return a value very close to 0.

5.2 Network Nodes Database

Let us consider a case where we have the database of an internet service provider. This database contains a set of network nodes determined by a discrete label (e.g. A2345-C, B3490-D). If we use alphabetical order and do not care about the similarity between nodes, the impact produced by watermark embedding could be important. However, if we consider a similarity function this impact could be clearly reduced.

Similarity function: In this case, the similarity function can be defined as the number of "hops" between nodes. This measurement gives a very concise idea about the location of the nodes. Thus, if two nodes are nearby, the similarity function will tend to 1. On the contrary, if the number of "hops" from one node to another is large, the similarity function returns a number close to 0.

6 Conclusions and Further Work

We have presented a new watermarking system for protecting non-numerical data. The system minimizes the number of modifications needed to embed the mark and allows the data owner to define a similarity function to guide each individual modification so that the utility loss it entails is minimal. The robustness analysis demonstrates the resiliency of our mark against different kind of malicious attacks. The similarity function is user-defined and depends on the particular database to be protected; this has been illustrated with two examples. Future work will involve a false positive rate analysis and extensive robustness tests in large databases with a broader range of attacks. Also, the definition of a similarity function that optimally (rather than reasonably) captures data utility loss in a specific database is a nontrivial issue for future research in artificial intelligence.

Acknowledgments

The authors are partly supported by the Catalan government under grant 2005 SGR 00446, and by the Spanish Ministry of Science and Education through project SEG2004-04352-C04-01 "PROPRIETAS".

References

1. Jordan, F., Vynne, T.: Motion vector watermarking. Patent (1997) Laboratoire de Traitement des Signaux École Polytechnique Fédérale de Lausanne.
2. Hartung, F., Girod, B.: Watermarking of uncompressed and compressed video. Signal Processing **66** (1998) 283–301
3. Katzenbeisser, S., Petitcolas, F.A.P.: Information Hiding: techniques for steganography and digital watermarking. Computer security series. Artech House (2000)
4. Domingo-Ferrer, J.: Anonymous fingerprinting of electronic information with automatic reidentification of redistributors. Electronics Letters **34** (1998) 1303–1304
5. Pitas, I., Kaskalis, T.H.: Applying signatures on digital images. In: IEEE Workshop on Nonlinear Signal and Image Processing, Thessaloniki, Greece (1995) 460–463
6. Ruanaidth, J.O., Downling, W., Boland, F.: Phase watermarking of digital images. In: Proceedings of the IEEE international Conference on Image Processing. Volume 3. (1996) 239–242
7. Cox, I.J., Kilian, J., Leighton, T., Shamoon, T.: Secure spread spectrum watermarking for multimedia. Technical report, NEC Research Institute, Technical Report 95 - 10 (1995)
8. Sebé, F., Domingo-Ferrer, J., Solanas, A.: Noise-robust watermarking for numerical datasets. Lecture Notes in Computer Science **3558** (2005) 134–143
9. Delaigle, J.F., Vleeschuwer, C.D., Macq, B.: Watermarking using a matching model based on the human visual system, Marly le Roi. (1997)
10. Ko, B.S., Nishimura, R., Suzuki, Y.: Time-spread echo method for digital audio watermarking. IEEE Transactions on Multimedia **7** (2005) 212–221
11. Collberg, C.S., Thomborson, C.: Watermarking, tamper-proofing and obfuscation: tools for software protection. IEEE Transactions on Software Engineering **28** (2002) 735–746 http://dx.doi.org/10.1109/TSE.2002.1027797.
12. Agrawal, R., Haas, P.J., Kiernan, J.: Watermarking relational data: framework, algorithms and analysis. VLDB journal **12** (2003) 157–169
13. Sion, R.: Proving ownership over categorical data. In: Proceedings of the 20th International Conference on Data Engineering (ICDE04), Boston (2004) 584–595
14. NIST: Proposed federal information processing standard for secure hash standard. Federal Register **57** (1992) 41727