9-2006

Rights Protection for Data Cubes

Jie GUO
Shanghai Jiaotong University

Yingjiu LI
Singapore Management University, yjli@smu.edu.sg

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Kefei CHEN
Shanghai Jiaotong University

DOI: https://doi.org/10.1007/11836810_26

Citation

GUO, Jie; LI, Yingjiu; DENG, Robert H.; and CHEN, Kefei. Rights Protection for Data Cubes. (2006). *Information Security: 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2: Proceedings*. 4176, 359-372. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/320

Rights Protection for Data Cubes

Jie Guo[1,2], Yingjiu Li[1], Robert H. Deng[1], and Kefei Chen[2]

[1] School of Information Systems, Singapore Management University, 178902, Singapore
{yjli, robertdeng}@smu.edu.sg
[2] School of Information Security Engineering, Shanghai Jiaotong University, 200030, China
{guojie, kfchen}@sjtu.edu.cn

Abstract. We propose a rights protection scheme for data cubes. The scheme embeds ownership information by modifying a set of selected cell values. The embedded message will not affect the usefulness of data cubes in the sense that the sum queries at any aggregation level are not affected. At the same time, the errors introduced to individual cell values are under control. The embedded message can be detected with a high probability even in the presence of typical data cube attacks. The proposed scheme can thus be used for protecting data cubes from piracy in an open, distributed environment.

1 Introduction

Data cube is a common data model that supports exploration of a large amount of electronic data from many different perspectives, in a multi-dimensional and multi-level manner. In many applications, valuable data cubes are provided to multiple users or customers. For example, business and government organizations often outsource valuable data cubes, such as sales patterns data, financial data, or customer information, to certain parties that are specialized in analyzing the data. For another example, data cubes of online interactions (e.g., airline reservation and scheduling portals) are usually provided for direct and interactive uses by many customers across the internet. In these applications, it is critical to protect the owner's rights so as to thwart any illegal use of data. Without appropriate rights protection, some dishonest users may copy and redistribute the data without its owner's permission.

Watermarking techniques have been frequently used for protecting data ownership. The ownership information is embedded into the target data in a way that it has no significant impact on the usefulness of the data and that a malicious user cannot destroy it without making the data less useful. When a pirated copy of data is discovered somewhere, the owner of the data can assert its ownership with a high probability by extracting the embedded information from the watermarked data. Watermarking has been extensively studied in the context of multimedia data (text, image, sound or video) [9] [5] [14] [15] [10] [18]. Since multimedia objects are ultimately watched or listened by human beings, it is critical that the embedded watermark has no significant impact on human perceptual systems. Recently, there have been growing interests in watermarking non-media data [3] such as relational databases [1], softwares [8], natural language texts [4], and sensor network streams [19]. The challenges posed in these new domains are quite different from those posed in the multimedia domain. As pointed out in [1][3], non-media data have characteristics and operations very different from multimedia data, thus requiring different watermark algorithms to be designed.

Agrawal et al. first proposed a watermarking scheme for relational databases [2]. A private key, known only to the owner of the data, was used to determine where to embed a watermark and how to detect it. Li at el. extended Agrawal's watermarking scheme to embed user-specific information, called fingerprint, into relational databases [17]. The fingerprint can be used to identify or track whom the traitor is from a pirated copy of data. For numerical data, Sion et al. introduced a multi-bit distribution encoding scheme that can survive the linear transformation of data among other types of attacks [20]. For categorical data, Sion proposed another watermarking technique that swaps certain categorical values so as to embed watermark information [21]. Other works include Gross-Amblard's watermark framework [12], in which a set of parametric queries can be preserved within a certain level of distortion, and Bertino's hierarchical watermarking scheme [7] for preserving the privacy and ownership of outsourced medical data against generalization attacks. All these works are targeted on watermarking relational databases.

The watermarking techniques customized for relational databases cannot be directly applied to data cubes. One reason is that the structure of data cubes is different from that of databases. While most schemes for watermarking databases depend on the existence of a primary key attribute, there is no such concept in data cubes. Another reason is that data cubes are primarily used for answering sum queries regarding the aggregations of cell values at different levels. A watermarking scheme should have as less impact as possible, better no impact, on the data cube queries. In comparison, no special treatment has been made in watermarking relational databases for reducing or eliminating the errors in sum queries.

The contribution of this paper is multi-fold. Firstly, we propose the first watermarking scheme for protecting the ownership of data cubes. A data cube owner can use his private key to control all watermarking parameters. Neither original data cube nor the watermark is required in watermark detection. Secondly, we devise the concept of mini-cube and apply it in the process of watermarking. As a result, all sum queries in a watermarked data cube can be answered without any error, yet the robustness of the embedded watermark is not affected by the use of mini-cubes. Thirdly, we extend our basic scheme to improve watermarking efficiency for very large data cubes. Detailed analysis and extensive experiments are conducted for the proposed schemes in terms of watermark detectability, robustness, imperceptibility, and efficiency. Our results show that the scheme perform well in actual applications.

The rest of this paper is organized as follows. Section 2 revisits the basic model of data cubes. Section 3 presents our watermarking algorithms. Section 4 analyzes the properties of the proposed algorithms. Section 5 evaluates our watermarking technique using real data. Section 6 concludes this paper.

2 The Basic Model of Data Cubes

We follow the data cube model proposed in paper [11]. Conceptually, a cube consists of a base cuboid, surrounded by a collection of aggregation cuboids that represent the

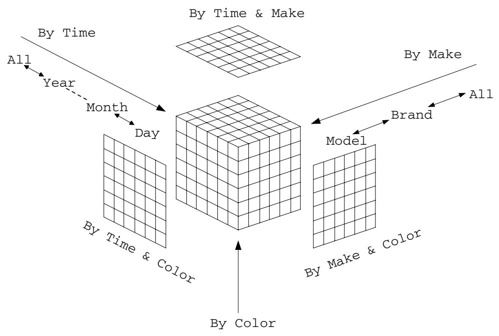

Fig. 1. An example data cube

aggregation of the base cuboid along one or more dimensions. We refer to the attribute to be aggregated as the *measure attribute*, while the other dimensions are seen as the *feature attributes*. Figure 1 gives an example of data cube from the automotive industry. The data cube is in 3 dimensions, where each cell (m, c, t) represents a combination of three feature attributes *make*, *color*, and *time*, and where a single measure attribute of the cell is *sale*. The measure attribute value of cell (m, c, t) indicates the sale of the car model m in color c on day d.

The base unit of each dimension may be aggregated to higher level units in "roll-up" operations. For example, the *time* dimension can be aggregated from the basic unit *day* to higher level units *month*, *year*, and *all*. Consequently, the cell values (i.e., car sales) are aggregated in terms of the new units in roll-up operations. If the 3-D based cuboid is rolled-up from *day* to *all* along *time* dimension, then it becomes a 2-D plane, which is labelled as "by make and color" in figure 1. The 2-D plane can be further rolled-up to a 1-D line and to a 0-D point (the 0-D point represents the grand total of the car sales for all make, all color, at all time). From higher level aggregations, one may also "drill-down" to lower level aggregations or individual cells.

Roll-up and drill-down are basic OLAP functions which can be used for knowledge discovery in decision support systems. By rolling up from the base cuboid, one can aggregate the measure attribute at various levels, thus discovering general trends of the underlying data. By drilling down to lower level aggregations or individual cells, one may discover outliers or exceptions. The interactive exploration of rolling-up and drilling down may repeat until a satisfactory understanding of the underlying data is reached. Based on the data cube model, we can design our watermarking scheme for data cubes.

3 Embedding and Detecting Watermarks

In this section, we introduce our watermarking scheme in detail. We assume that all watermarkable values in a data cube are numeric, and that small changes in a small portion of these values are acceptable. Note that the same assumption has been commonly used in watermarking relational data. It has been argued that such an assumption is supported by many applications for various types of data (e.g., parametric specifications in manufacturing industry, geological and climatic surveys, and life science data) [1].

Table 1. Notations and parameters

X_i, Y_j, Z_k	Three feature attributes of a cell
N_x, N_y, N_z	Sizes of three feature attributes in the base cuboid
η	Total number of cells in the data cube
ξ	Number of least significant bits available for watermarking in each cell
$1/\gamma$	Fraction of cells selected for embedding a watermark
ω	Number of cells selected for embedding a watermark
α	Significance level of the test for detecting a watermark
τ	Minimum number of correctly detected cells for ownership claim

Algorithm 1. Watermark embedding

```
1: for $i = 1$ to $N_x, j = 1$ to $N_y, k = 1$ to $N_z$ do
2:   $HMAC(i,j,k) = \mathcal{H}(\mathcal{K} \oplus opad, \mathcal{H}(\mathcal{K} \oplus ipad, X_i \circ Y_j \circ Z_k))$
3:   $mark(i,j,k) = HMAC(i,j,k) \bmod \gamma$
4: end for
5:
6: for $i = 1$ to $N_x, j = 1$ to $N_y, k = 1$ to $N_z$ do
7:   if ($mark(i,j,k) = 0$) then   // This cell is selected for marking
8:     $bp = HMAC(i,j,k) \bmod \xi$   // Mark position at this cell is $bp^{th}$ bit
9:     $wm = HMAC(i,j,k) \bmod 2$   // Watermark allocated to this cell is $wm$
10:    $bm = (d(i,j,k) >> bp)\&1$   // The $bp^{th}$ LSB of the cell value is $bm$
11:    if ($bm \neq wm$) then   // The cell value should be changed for marking
12:      set the $bp^{th}$ least significant bit of the cell value to $wm$
13:      $df$ = the difference between marked cell value and original cell value
14:      $minicube(i,j,k,df,D,mark)$   // Construct a minicube for the modified cell
15:    end if
16:  end if
17: end for
```

Without loss of generality, we present our scheme for a 3-dimensional data cube D, where each cell has three feature attributes (X, Y, Z) and one measure attribute M. The sizes of three feature attributes (i.e., the numbers of base units) are N_x, N_y, N_z, respectively. The total number of cells η is $N_x \times N_y \times N_z$. We use $d(X_i, Y_j, Z_k)$ to denote the cell value at the position (X_i, Y_j, Z_k). For each watermarkable cell, we assume that any change in one of its ξ least significant bits is imperceptible, where ξ is a parameter in our scheme. Another parameter γ determines the number ω of watermarkable cells that are used to embed a watermark. Approximately one out of every γ values is used in watermarking (i.e., $\omega \approx \eta/\gamma$). A significance level α is used to determine how amenable the watermarking system is to the false detection of a watermark from non-watermarked data. A parameter τ is used to denote the minimum number of correctly detected cells for ownership claim. The two parameters will be further explained in Section 3.2. For ease of reference, Table 1 gives the notations that will be used in this paper.

3.1 Watermark Embedding

The procedure of watermark embedding is shown in Algorithm 1. Let HMAC be a MAC function seeded with a private key [6][16]. For each cell in a 3-D data cube, the owner of the data computes a HMAC value from the cell's feature attributes (X_i, Y_j, Z_k) using his private key $\mathcal{K}$. The HMAC value will be used to determine whether the cell is

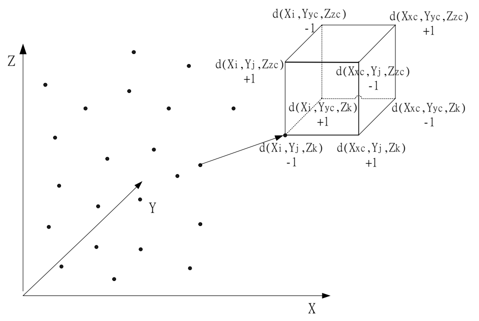

Fig. 2. An example mini-cube

selected to embed a watermark (see lines 3 and 7 in algorithm 1). On average, one out of every γ cells is selected. Because of the use of the HMAC function and the private key, only the owner of the data can determine which cells are actually selected. For each selected cell, line 8 in algorithm 1 determines a bit position among ξ least significant bits, and line 9 computes a watermark bit value which is assigned to the bit position. The watermark bit has a probability of 1/2 to be the same as the original bit in the bit position, in which case the selected cell does not change. Otherwise (see line 11), the original bit is flipped, in which case the selected cell is modified so as to embed the watermark bit.

According to our assumptions, the change made in watermarked cells is acceptable for individual values; however, it can be potentially significant to some aggregations of the cell values. In order to reduce or eliminate the accumulative errors that are introduced in watermark insertion, a mini-cube is constructed for each cell that is modified in watermarking (see line 14 in algorithm 1). Figure 2 illustrates a simple example for constructing a mini-cube. Suppose that the value of cell $d(X_i, Y_j, Z_k)$ is decremented by 1 in watermark insertion. Based on the position of $d(X_i, Y_j, Z_k)$, three other cell values $d(X_{xc}, Y_j, Z_k)$, $d(X_i, Y_{yc}, Z_k)$, and $d(X_i, Y_j, Z_{zc})$ are selected. The values of these cells are incremented by one so as to balance the deviation in any 1-D aggregation (i.e., aggregation along one feature dimension) that involves cell (X_i, Y_j, Z_k). Similarly, three more cell values $d(X_{xc}, Y_j, Z_{zc})$, $d(X_i, Y_{yc}, Z_{zc})$, and $d(X_{xc}, Y_{yc}, Z_k)$ are decremented by one, and one last cell value $d(X_{xc}, Y_{yc}, Z_{zc})$ is incremented by one. These seven cells, which we call *balance cells*, form a mini-cube together with the watermarked cell (X_i, Y_j, Z_k). With a mini-cube constructed, any data cube aggregation that involves at least two cells in the mini-cube remains unchanged after watermark insertion. Algorithm 2 gives the procedure for constructing a mini-cube.

A mini-cube is not constructed without a constraint. Firstly, the balance cells should not be selected from any cell that has been selected in watermark insertion so as to avoid interfering the watermark insertion and detection. Secondly, a mini-cube should be constructed in a way that most, if not all, aggregation queries would involve at least two cells in the mini-cube. To achieve this, (i) the balance cells should be selected as close to the watermarked cell as possible, and (ii) any two cells in the mini-cube should have the same attribute values *in terms of the smallest aggregation units* above the base unit (e.g., “brand” for attribute *model* and “month” for attribute *day* are such aggregation units in

Algorithm 2. Tow functions: mini-cube embedding and choosing cells for mini-cube

```
1: mini-cube(X-index $i$, Y-index $j$, Z-index $k$, value-difference $df$, data cube $D$, embedding position $mark$)
2: $xc = 0; yc = 0; zc = 0;$   // Parameters used for deciding cell positions for mini-cube
3: $(xc, yc, zc) = CellPosition(i, j, k, D, mark)$   // Choose cell positions for mini-cube
4: if $(xc \neq i)$ then   // Construct mini-cube
5:   $d(xc, j, k) = d(xc, j, k) - df$
6:   $d(i, yc, k) = d(i, yc, k) - df$
7:   $d(i, j, zc) = d(i, j, zc) - df$
8:   $d(xc, yc, k) = d(xc, yc, k) + df$
9:   $d(xc, j, zc) = d(xc, j, zc) + df$
10:   $d(i, yc, zc) = d(i, yc, zc) + df$
11:   $d(xc, yc, zc) = d(xc, yc, zc) - df$
12: end if
13:
14: CellPosition(X-index $i$, Y-index $j$, Z-index $k$, data cube $D$, embedding position $mark$)
    return (X-position $xc$, Y-position $yc$, Z-position $zc$)
15: $lx = round(i/\lambda_x) * \lambda_x$   // Round real value to integer
16: $my = round(j/\lambda_y) * \lambda_y$
17: $nz = round(k/\lambda_z) * \lambda_z$
18: $xc = i; yc = j; zc = k;$   // Parameters used for deciding cell positions for mini-cube
19: $SumThresh = 0$   // Value used for finding optimal mini-cube
20: for $l = lx$ to $lx + \lambda_x - 1, m = my$ to $my + \lambda_y - 1, n = nz$ to $nz + \lambda_z - 1$ do
21:   // Balance cells are not selected from the cells that are selected in watermark insertion
22:   if $((mark(l, j, k) \neq 0)$ and $(mark(i, m, k) \neq 0)$ and $(mark(i, j, n) \neq 0)$ and $(mark(l, m, k) \neq 0)$
      and $(mark(l, j, n) \neq 0)$ and $(mark(i, m, n) \neq 0)$ and $(mark(l, m, n) \neq 0))$ then
23:     $sum = |d(l, j, k)| + |d(i, m, k)| + |d(i, j, n)| + |d(l, m, k)| + |d(l, j, n)| + |d(i, m, n)| + |d(l, m, n)|$
24:     if $(sum > SumThresh)$ then
25:       $SumThresh = sum; xc = l; yc = m; zc = n;$
26:     end if
27:   end if
28: end for
29: return $(xc, yc, zc)$
```

Figure 1). We use three parameters λ_x, λ_y, and λ_z to decide how far away along each dimension to search the balance cells from a watermarked cell. These parameters can be fixed or floating for different watermarked cells. The last requirement for constructing a mini-cube is that the modification to the balance cells should be minimum. The cells with larger values are better to be selected to be balance cells, for smaller values are more sensitive to the modification that are introduced to the individuals cells during the construction of a mini-cube. In our scheme, we sum up the absolute values of candidate balance cells and select those with maximal sum value.

Note that in watermark insertion, a small portion of the cells, the bit positions of the selected cells, and the bit values assigned to the selected bit positions are all algorithmically determined under the control of a private key. The bit pattern constitutes the watermark. Without knowing the private key, an attacker is not able to know where exactly the watermark is embedded. We also note that the same HMAC function is used to determine the cells, the bit positions and the bit values in our scheme. To further increase the randomness of this process, different HMAC functions can be employed instead of single HMAC function.

3.2 Watermark Detection

The watermark detection algorithm is blind. It neither requires the knowledge of the original data cube nor the watermark in detection. Since the mini-cubes do not interfere

Algorithm 3. Watermark detection

```
1: for $i = 1$ to $N_x, j = 1$ to $N_y, k = 1$ to $N_z$ do
2:   $\mathcal{HMAC} = \mathcal{H}(\mathcal{K} \oplus opad, \mathcal{H}(\mathcal{K} \oplus ipad, X_i \circ Y_j \circ Z_k))$
3:   if ($\mathcal{HMAC}$ mod $\gamma$ equals 0) then    // This cell was marked
4:     $bp$ = $\mathcal{HMAC}$ mod $\xi$    // $bp^{th}$ bit was marked
5:     totalcount=totalcount+1
6:     matchcount=matchcount+match($\mathcal{HMAC}, b(X_i, Y_j, Z_k), bp$)
7:   end if
8: end for
9:
10: $\tau$ = threshold(totalcount, $\alpha$)    // See section 4.1
11: if (matchcount$\geq \tau$) then
12:   suspect piracy
13: end if
14:
15: match(MAC value $\mathcal{HMAC}$, cell value $v$, bit-index $j$) return $int$
16: if ($\mathcal{HMAC}$ is even) then
17:   return 1 if the $j^{th}$ least significant bit of $v$ is 0 else return 0
18: else
19:   return 1 if the $j^{th}$ least significant bit of $v$ is 1 else return 0
20: end if
```

with any cells that have been selected in watermark insertion, there is no need to consider the mini-cubes in watermark detection.

The watermark detection is shown in Algorithm 3. Line 3 determines whether a cell has been watermarked. Line 4 determines the bit position that have been watermarked. The function $match$ compares the observed bit value at the bit position with the correct watermark bit that should be allocated at the position if the data is correctly watermarked. To claim the ownership over the detected data, one must know how many cells were tested (total count) and how many of them contained the expected watermark bit values (match count). In a probabilistic framework, only if a certain minimum number τ of cells contain the expected bit values, the ownership is claimed(see Line 10). The match count is compared with the minimum number, τ, which is returned by a threshold function (see Line 11).

A significance level α is used in the threshold function to determine the minimum match count τ (see Line 10). The significance level is the upper bound of the probability that the ownership is falsely claimed for a non-watermarked data cube. If the significance level is $\alpha = 10^{-9}$, for example, the probability of falsely detecting a watermark is less than 10^{-9}. A formal analysis on the detection probability and the threshold function is given in Section 4.1.

3.3 Extensions

The computational cost of our scheme is mainly determined by the amount of HMAC computation in the watermark insertion and detection. In our original scheme, an HMAC value is computed for each cell. The computation cost is $O(N_x \times N_y \times N_z)$ in terms of HMAC operations, where N_x, N_y, and N_z are the sizes of the feature attributes ($N_x \times N_y \times N_z$ is the total number of cells). We can extend the scheme such that an HMAC value is computed for each feature attribute value, rather than each cell. The HMAC value decides whether the feature attribute value is selected for embedding a watermark. An attribute value is selected if its HMAC modular $\sqrt[3]{\gamma}$ yields zero. On average, one

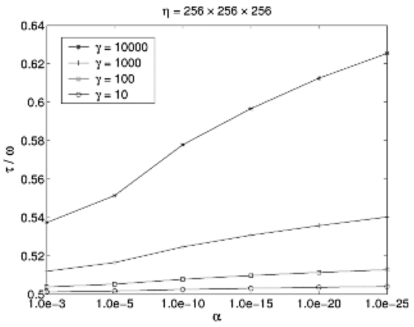

Fig. 3. Proportion of correctly detected marks required for claiming ownership in watermark detection with a significance level α

out of every $\sqrt[3]{\gamma}$ attribute values is selected in each dimension. A cell is selected for embedding a watermark if all its feature attributes are selected. Overall, one out of every $\sqrt[3]{\gamma} \times \sqrt[3]{\gamma} \times \sqrt[3]{\gamma} = \gamma$ cells is selected. This number is again the same as in our original scheme. For each selected cell, a bit position and a bit value can be determined based on the sum of the HMAC values that have been computed for the cell's feature attributes. It is easy to know that the computational cost of this extended scheme is $O(N_x + N_y + N_z)$ in terms of HMAC operations. Compared with the original scheme, the extended scheme is more efficient for watermarking large data cubes.

4 Analysis

In the above section, we have designed a watermarking scheme for numeric data cubes and the scheme satisfies some particular requirements on aggregation, localization, non-watermarkable values, and computational cost. We now analyze the properties of the proposed scheme.

4.1 Detectability

To make our watermark detection reliable, the probability of falsely detecting a watermark from non-watermarked data must be low. This probability is controlled in our scheme by adjusting the significance level α and the number ω of watermarked cells. Bernoulli trials are employed for analyzing the probability of detecting each watermark bit from non-watermarked data. Since each watermark bit is determined by a HMAC pseudo-randomly, it has a probability of 1/2 to match the corresponding bit in non-watermarked data. Let $b(i; n, p)$ be the probability that n Bernoulli trials result in i successes and $n - i$ failures, where p is the probability of success and $q = 1 - p$ the probability of failure. The probability of having at least k successes in n trials (i.e., the cumulative binomial probability) can be written as

$$B(k; n, p) = \sum_{i=k}^{n} b(i; n, p) = \sum_{i=k}^{n} \frac{n!}{i!(n-i)!} p^i q^{n-i} \tag{1}$$

We now specify the threshold function that is used in Line 10 of Algorithm 3. Supposing $totalcount = \omega$, the watermark detection would examine ω "watermark bits" from non-watermarked data. The probability of falsely detecting a watermark (i.e., at least τ out of ω "watermark bits" match their expected values by sheer chance) is $B(\tau; \omega, 1/2)$. Given the significance level α, the threshold function $threshold(\omega, \alpha)$ returns the minimum value τ such that $B(\tau;\ \omega,\ 1/2) < \alpha$.

The significance level α determines how amenable the watermarking system is to the false detection of a watermark. By choosing a lower value of α, the confidence level in the detection can be raised up if the detection algorithm discovers the owner's watermark in a suspicious data cube. Figure 3 plots the required proportion (i.e., $\tau/\omega = \tau\gamma/\eta$) of the correctly detected marks for different values of α and γ. Clearly, we need to proportionately increase the number of the correctly detected marks as the value of α decreases. The figure also shows that the required proportion of the correctly detected marks decrease as the percentage of the watermarked cells increases. This illustrates that for larger data cubes, a smaller percentage of the total number of cells can be watermarked without increasing the significance level α.

4.2 Robustness

We now analyze the robustness of our watermarking technique against several malicious attacks including value modification, value selection, additive attack, and invertibility attack.

Value Modification. In a value modification attack, an attacker tries to destroy the owner's watermark by modifying some cell values. There are various forms of value modification attack including bit-flipping, zero-out, and randomization. In all value modification attacks, the exact positions where a watermark is embedded are hidden from an attacker who does not know the private key. What an attacker can do is to apply the attacks to all cell values (or bits), or a portion of them in a random manner. The effect of these attacks is reflected in the portion V_m of the watermarked bits that have been changed/flipped. For example, if 30% of the least significant bits of all cell values are flipped in a bit-flipping attack, then the portion V_m is 30%. In a zero-out attack, we have $V_m = \min(\nu/(2\xi), 1/2)$ if ν least significant bits of each cell value are set to zero. In a randomization attack, one has $V_m = 1/(2r)$ if one out of every r cells is modified in a random manner. Given the portion V_m of the watermarked bits that have been changed, the sufficient and necessary condition that the embedded watermark can still be detected with a significance level α is $V_m \leq 1 - \tau/\omega$, or equivalently, $V_m \leq 1 - \tau\gamma/\eta$. Taking Figure 3 as an example, the portion V_m can be as large as $1 - 52\% = 48\%$ for $\gamma = 1000, \alpha = 10^{-5}$ and $\eta = 256 \times 256 \times 256$.

Value Selection. In a value selection attack, a subset of cells are selected from the original data cube with an intension that the embedded watermark cannot be detected from this subset of cells with a high probability. Typical value selection attacks in a data cube include data cube slicing, iceberg-cube, and random value selection. For all types of the value selection attack, the number ω' of the watermarked cells that are included in the value selection is proportional (i.e., $1/\gamma$) to the total number η' of the watermarkable cells that are included in this selection. By a value selection attack only, no errors are introduced to the watermarked values. Therefore, the watermark detection will claim ownership for any non-zero significance level as long as ω' is greater than zero. The probability of $\omega' = 0$ is $(1 - 1/\gamma)^{\eta'}$, which can be made extremely low for a reasonably large η. For example, let $\eta = 256 \times 256 \times 256$ and assume that only one thousandth of the cells are selected in a value selection attack (i.e., $\eta' = \eta/1000$). Then the probability that no watermarked cells are included in the selection is about 5.9×10^{-74} for $\gamma = 100$, and about 5.1×10^{-8} for $\gamma = 1000$.

Additive Attack. In an additive attack, an attacker simply inserts his watermark into a data cube that has already been watermarked by its owner. If both watermarks are detected, then the ownership of the data is in dispute. To thwart this type of attack, Agrawal et al. [1] suggested to locate the overlapping regions of the two watermarks in which the bit values conflict. However, this may not be always possible if there is a value modification attack. To reach a decision with certain significance level, a enough number of watermarked cells that collide must be detected. This may not be realistic since the probability that the bit values of two watermarks conflict in any given cell is $\frac{1}{2(\gamma\xi)^2}$, which is extremely low for reasonably large γ (e.g., $\gamma = 100$). To solve this problem, one may consider using public watermark schemes (i.e., asymmetric watermarking, see [13]) that involve public key primitives such as trusted registration authorities and verifiable certificates to resolve any ownership dispute.

Invertibility Attack. An invertibility attack has also been identified in [1] for falsely claiming ownership using counterfeit watermarks. This attack discovers a key, which may or may not be the same as the original private key, to detect a satisfactory watermark from watermarked data for certain significance level α. This attack can be thwarted by imposing two additional requirements on watermarking schemes by convention. First, the private key should be long enough to thwart brute force search. Second, the significance level α should be low enough (e.g., 10^{-10}) such that the probability of accidently finding a key that yields a satisfactory watermark is negligible.

5 Real Data Experiments

We now report experimental results that complement the analysis presented in Section 4. The experiments are performed using a real 3-D data cube, the LCDM (Lambda-dominated Cold Dark Matter) cluster simulations in astronomy, available from the Department of Astronomy and Astrophysics at the University of Chicago (http://astro.uchicago.edu/d̃aisuke/Research/simdata.html#threed).

The data cube has 16,777,216 cells, and each cell value can be converted to an integer of 32 bits. In experiments, we vary γ from 1 to 10000, and fix ξ, λ_x, λ_y, and λ_z to 8. Our watermarking scheme is implemented in Visual C++ Version 6 using HMAC-SHA1 [1] as the HMAC function. All the experiments are run on a HP Compaq computer with a Pentium(R) 4 CPU of clock rate 3.00GHz, 1.0 GB of RAM, and 40 GB hard-disk running Microsoft Windows XP.

[1] SHA-1 was recently found not as secure as it was believed to be [22]. Any one-way hash function can be used in our scheme. The reason for using SHA1 in our experiment is simply because of the availability of the HMAC-SHA1 code.

Table 2. Computational cost of watermark insertion and detection

	$\gamma = 1$	$\gamma = 10$	$\gamma = 100$	$\gamma = 1000$	$\gamma = 10000$
Number of cells selected for watermarking	16,777,216	1,676,954	167,124	16,764	1,703
Number of cells modified for watermarking	8,387,424	803,837	80,185	8,094	825
Time for embedding a watermark without mini-cubes (sec.)	299	158	144	142	141
Time for embedding a watermark with mini-cubes (sec.)	340	221	160	154	153
Percentage of watermarked cells for which mini-cubes can be constructed	0	100%	100%	100%	100%
Time for detecting a watermark (sec.)	254	158	143	141	140

5.1 Computational Cost

The first set of experiments evaluate the computational cost of the watermark insertion and detection. The experimental result is summarized in Table 2. The performance of our algorithms is measured in elapsed time. For the watermark insertion, the computational cost for constructing mini-cubes can be assessed by comparing the running time with mini-cubes and without mini-cubes.

In the watermark insertion, the computational cost can be broken down into three components: 1) The computation of a HMAC value for each cell, determining whether the cell is selected for embedding a watermark; 2) The modular computation for each cell that is selected, determining which bit position is selected, what the watermark bit value is, and whether the original cell value is to be modified for embedding the watermark bit; and 3) The computation of modifying a bit value and constructing a mini-cube for each cell that is modified.

The experimental results show that the computational cost increases as the percentage of the watermarked cells increases. In the extreme case where $\gamma = 1$, the number of the watermarked cells reaches its maximum. In this case, however, no mini-cube can be constructed, for every cell is selected for watermarking. Nonetheless, it takes longer for embedding a watermark with mini-cubes than without mini-cubes because of the time taken to attempt to construct mini-cubes. When γ is set to 10, the time required for mini-cube construction is $221 - 158 = 63$ seconds. When γ increases to 10000, the time used for mini-cube construction is down to 12 seconds because much less mini-cubes need to be constructed. In all of the experiments, the computation of HMAC values is the major component of the cost for watermark insertion.

Table 2 also gives the running time for watermark detection. The cost of watermark detection is similar to that of watermark insertion except that there is no need of mini-cube construction. Again, the major component of the cost is the computation of HMAC values. Note that the watermark insertion or detection can be done within five minutes in our experiments for a data cube with 256^3 cells. This result indicates that our algorithms have adequate performance to allow for their use in real world applications.

5.2 Imperceptibility

Since a data cube is a generalization of aggregation functions, the main service provided by a data cube (e.g., in OLAP) is to answer aggregation queries at various aggregation levels. To maintain the usefulness of data, the embedded watermark must be imperceptible; that is, it does not introduce intolerable errors to any aggregation queries that can be answered by a data cube. Consider the data cube shown in Figure 1. An example of

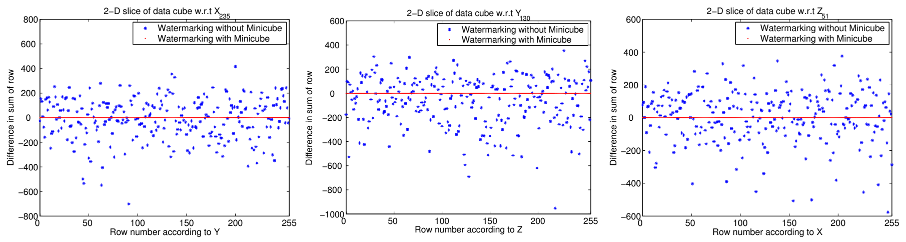

Fig. 4. Comparison of imperceptibility between two watermarking schemes

the aggregation queries asks for the total sale of red Ford GT cars. To answer this query, a row is localized in the data cube according to the *color* attribute "red" and the *make* attribute "Ford GT." Then, all the cell values in the row are summed up to get the whole sale. To measure the effect to the aggregation queries of the watermark insertion, we calculate the differences between the sums of the original cell values and those of the watermarked cell values.

In our real data experiments, we evaluate typical aggregation queries that sum the cell values in each row in a randomly chosen slice, where a slice is a 2-D plane in the data cube. Let X, Y and Z denote the three feature attributes of the data cube, each of which has 256 values (i.e., from X_0 to X_{255}). A 2-D slice w.r.t. X_{235}, for example, denotes the 2-D plane in which the cells have the same attribute value $X = X_{235}$. For watermarking the data cube, we use $\gamma = 9$ and $\xi = 8$ in our experiments.

Figure 4 illustrates the differences in the sums of rows for three randomly chosen slices w.r.t. X_{235}, Y_{130}, and Z_{51} respectively. The differences in the sums are measured between the original and watermarked cell values. The differences indicate the errors that are introduced by the watermark insertion to the sum queries. We compare two types of watermark insertion, with mini-cubes and without mini-cubes. In Figure 4, the scattered points '$*$' indicate the errors introduced by watermark insertion without mini-cubes. The points '.', which are lined-up horizontally at zero value, indicate zero error that watermark insertion with mini-cubes always introduces. In comparison, the errors that are introduced by the watermark insertion without mini-cubes are notably large.

6 Conclusion

With the wide spread applications of data cube models in on-line analytical processing, security techniques for data cube ownership protection is becoming increasingly important. However, to our knowledge, little research work has been done to assert rights over distributed or sold data cubes.

In this paper, we proposed the first robust watermarking scheme for protecting the ownership of numerical data cubes. In our watermarking scheme, a data cube owner uses his private key to control watermark embedding parameters, including cell positions, bit positions, and specific bit values. Our blind detection algorithm requires neither the original data cube nor the watermark during watermark detection. The most prevalent data cube operations are aggregation queries. To eliminate errors introduced by watermark to aggregation queries, we invented a novel concept called mini-cubes.

Based on clearly defined optimization rules, a mini-cube is constructed for each cell that is modified in watermarking such that all sum queries in the watermarked data cube can be answered error-free, while without introducing any degradation on the robustness of the embedded watermark. In addition, we presented an extension of our basic scheme to improve watermarking efficiency when dealing with very large data cubes. We conducted extensive analysis as well as empirical evaluation for the proposed schemes in terms of watermark detectability, robustness, imperceptibility, and efficiency. Our results indicate that the schemes perform extremely well in real world applications.

Our future research efforts include development of public watermark (in which ownership can be publicly proved) and fragile watermark (by which value modification can be detected and localized) schemes for data cubes.

Acknowledgement

This paper is funded by Office of Research, Singapore Management University.

References

1. R. Agrawal, P. J. Haas, and J. Kiernan. Watermarking relational data: framework, algorithms and analysis. *The VLDB Journal*, 12(2):157–169, 2003.
2. R. Agrawal and J. Kiernan. Watermarking relational databases. In *Proceedings of the 28th VLDB Conference*, pages 155–166, 2002.
3. M. Atallah. A survey of watermarking techniques for non-media digital objects (invited talk). In *ACSW Frontiers 2005*, page 73, 2005.
4. M. Atallah, V. Raskin, C. Hempelmann, M. Karahan, R. Sion, U. Topkara, and K. Triezenberg. Natural language watermarking and tamperproofing. In *Fifth International Information Hiding Workshop*, pages 196–212, 2002.
5. M. Atallah and S. Wagstaff. Watermarking with quadratic residues. In *Proceedings of IS&T/SPIE Conference on Security and Watermarking of Multimedia Contents*, volume 3657, pages 283–288, January 1999.
6. M. Bellare, R. Canetti, and H. Krawczyk. Keyed hash functions and message authentication. In *Proceedings of Crypto'96*, pages 1–15, 1996.
7. E. Bertino, B. Ooi, Y. Yang, and R. Deng. Privacy and ownership preserving of outsourced medical data. In *Proceedings of the 21st International Conference on Data Engineering*, pages 521–532, 2005.
8. C. Collberg and C. Thomborson. Software watermarking: Models and dynamic embeddings. In *Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of Programming Languages*, pages 311–324, 1999.
9. I. Cox, J. Kilian, T. Leighton, and T. Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*, 6(12):1673–1687, 1997.
10. I. Cox, M. Miller, and J. Bloom. *Digital Watermarking: Principles and Practice*. Morgan Kaufmann, 2001.
11. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
12. D. Gross-Amblard. Query-preserving watermarking of relational databases and xml documents. In *Proceedings of the 19th ACM Symposium on Principles of Database Systems (PODS)*, pages 191–201, 2003.

13. G. Hachez and J. Quisquater. Which directions for asymmetric watermarking. In *Proceedings of XI European Signal Processing Conference (EUSIPCO), Vol. I*, pages 283–286, 2002.
14. N. F. Johnson, Z. Duric, and S. Jajodia. *Information Hiding: Steganography and Watermarking - Attacks and Countermeasures*. Kluwer Academic, 2000.
15. S. Katzenbeisser and F. Petitcolas. *Information Hiding Techniques for Steganography and Digital Watermarking*. Artech House, January 2000.
16. H. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-hashing for message authentication. In *Internet RFC 2104*, February 1997.
17. Y. Li, V. Swarup, and S. Jajodia. Fingerprinting relational databases: Schemes and specialties. *IEEE Transactions on Dependable and Secure Computing*, 2(1):34–45, 2005.
18. R. Safavi-Naini. Tracing traitors: a selective survey. In *Digital Rights Management Workshop*, page 72, 2004.
19. R. Sion. Resilient rights protection for sensor streams. In *Proceedings of the Very Large Databases Conference*, pages 732–743, 2004.
20. R. Sion, M. Atallah, and S. Prabhakar. Rights protection for relational data. *IEEE Transactions on Knowledge and Data Engineering*, 16(12):1509–1525, 2004.
21. R. Sion, M. Atallah, and S. Prabhakar. Rights protection for categorical data. *IEEE Transactions on Knowledge and Data Engineering*, 17(7):912–926, 2005.
22. X. Wang, Y. Yin, and H. Yu. Finding collisions in the full sha-1. In *The 25th Annual International Cryptology Conference*, Aug. 2005. http://www.infosec.sdu.edu.cn/paper/sha1-crypto-auth-new-2-yao.pdf.