**Semi-Structured Data Protection Scheme Based on Robust Watermarking**

Jiahuan He1, Qichao Ying1, Zhenxing Qian2, \*, Guorui Feng1, and Xinpeng Zhang2

1 School of Communication and Information Engineering, Shanghai University, Shanghai, China

2 Shanghai Institute of Intelligent Electronics and Systems, School of Computer Science, Fudan University, Shanghai, China

\* Corresponding author: zxqian@fudan.edu.cn; Tel: +86-13764870417

**Abstract**

Semi-structured data is a widely used text format for data interchange and storage. This paper proposes a robust watermarking scheme of data protection for semi-structured data, which uses JSON format as an example for illustration. We first parse JSON file into a data structure of distinct embedding pairs. Afterwards, we generate a transfer matrix to get the intermediate sequences, which are then encoded using error-correction codes and embedded into the pairs. A private key is shared by the data hider and the recipient to resist collusion attacks. On the recipient’s side, data extraction can be successfully carried out even the received stego data are tampered. The imperceptibility is realized by embedding data into the less significant digits of numeric data in the cover file. The proposed scheme can be extended on several other formats. The experimental results show that the proposed scheme is robust to various kinds of typical attacks such as contextual truncating, modification, redundancy addition, etc.

**Keywords***:* Copyright protection, collusion attack, semi-structured data, watermarking.

# 1. Introduction

Semi-structured data are widely used for information storage and data interchange for its convenience and efficiency in data transmission. It is a form of data which contains tags to separate semantic elements and allows hierarchies of records and fields within the data. Some typical applications of semi-structured data are Extensible Markup Language (XML), JavaScript Object Notation (JSON), etc. Nowadays, the dissemination of digital content and data has brought about a series of problems, for example, contextual tampering and illegal duplication. The prevalence of semi-structured data highlights the importance of the protection of data and their copyrights.

Digital watermarking has been used in copyright protection and content authentication for various kinds of data formats. Secret data are embedded into the cover data in an imperceptible way, and can be successfully extracted regardless of attacks during data transmission. Robustness and fidelity are two basic requirements for digital watermarking. The embedded watermark must be robust enough against a variety of possible attacks such as contextual truncating or tampering that can destroy the watermark existence. Meanwhile, the watermarked data need to be close enough to the original data so that users can barely distinguish the differences between them. Generally, there is a trade-off between robustness and fidelity. A stronger robustness usually causes lower fidelity, and vice versa.

In the last decades, researchers mainly focus on the advancement of watermarking technology in multimedia, including algorithms for audio [1-3], image [4-8]and video [9-11]. Researchers have also proposed many watermarking schemes for texts and databases. For text watermarking, a typical method is conducting data embedding based on document structures by shifting rows and columns [15]. In semantic schemes, He et al. [16] uses synonym of the substitution and chaotic map to embed the watermark. In syntactic schemes, Mali et al. [17] proposes a technique which used grammatical rules like conjunctions and pronouns to generate the watermark. Image-based schemes [18] convert the text into an image at first. For the certain document format, Zhang et al. [19] defines new characters using TrueType for Microsoft Word to embed the watermark into the Word document. Kuribayashi et al. [20] proposes a scheme which used the spaces between words and lines for PDF format. Database watermarking schemes can be classified into two types: distortion-based watermarking and distortion-free watermarking [21-22]. The distortion-free watermarking uses the data itself to generate watermarks and is generally a fragile watermark. Oppositely, the distortion-based watermarking schemes modify the data with the minimal impact to embed the watermark. The watermarking technology utilizing contextual redundancy [23] embed the watermark in the redundant bits or meaningless strings. The technology based on data classification [24-25] is to find the classification data in the database, i.e., the data with a limited value for embedding. Recently, some schemes based on reversible watermarking technology are proposed. Zhang et al. [26] uses expansion on data error histogram. Jawad et al. [27] proposes a scheme based on reversible and difference expansion, which estimate the distortion using the mean and standard deviation of the watermark relationship.

However, there was little effort paid on copyright protection for the semi-structured data. The above-mentioned schemes for text watermarking are generally inapplicable or not effective on semi-structured data. Also, the schemes are generally not robust enough against some typical attacks such as deletion. In this paper, we propose a robust watermarking scheme for semi-structured data protection. For simplicity, we take JSON format as a representative to present the scheme. We first conduct data pre-processing that generates the valid embedding pairs of the cover file. We generate pseudo-random sequences using the hashing result of the keys as their seeds. We further generate a transfer matrix to get the intermediate sequences, which are then encoded using error-correction coding and embedded into the pairs. A private key is shared by data hider and recipient to resist collusion attacks. On the recipient’s side, data extraction can be successfully carried out even the received watermarked data are modified. The experimental results show that the proposed scheme is robust to various kinds of typical attacks. The proposed scheme can be easily extended on other typical formats, e.g. sheet-formatted data like Microsoft Excel or comma-separated values (CSV).

Fig.1 illustrates an application of the proposed scheme. The data hider embeds different watermarks for different users before sharing the cover files. The existence of watermarks is imperceptible and does not affect normal usage of the file If the recipients illegally upload the data to the Internet, the data hider can locate the source of the information leakage by conducting watermark extraction from the leaked file.

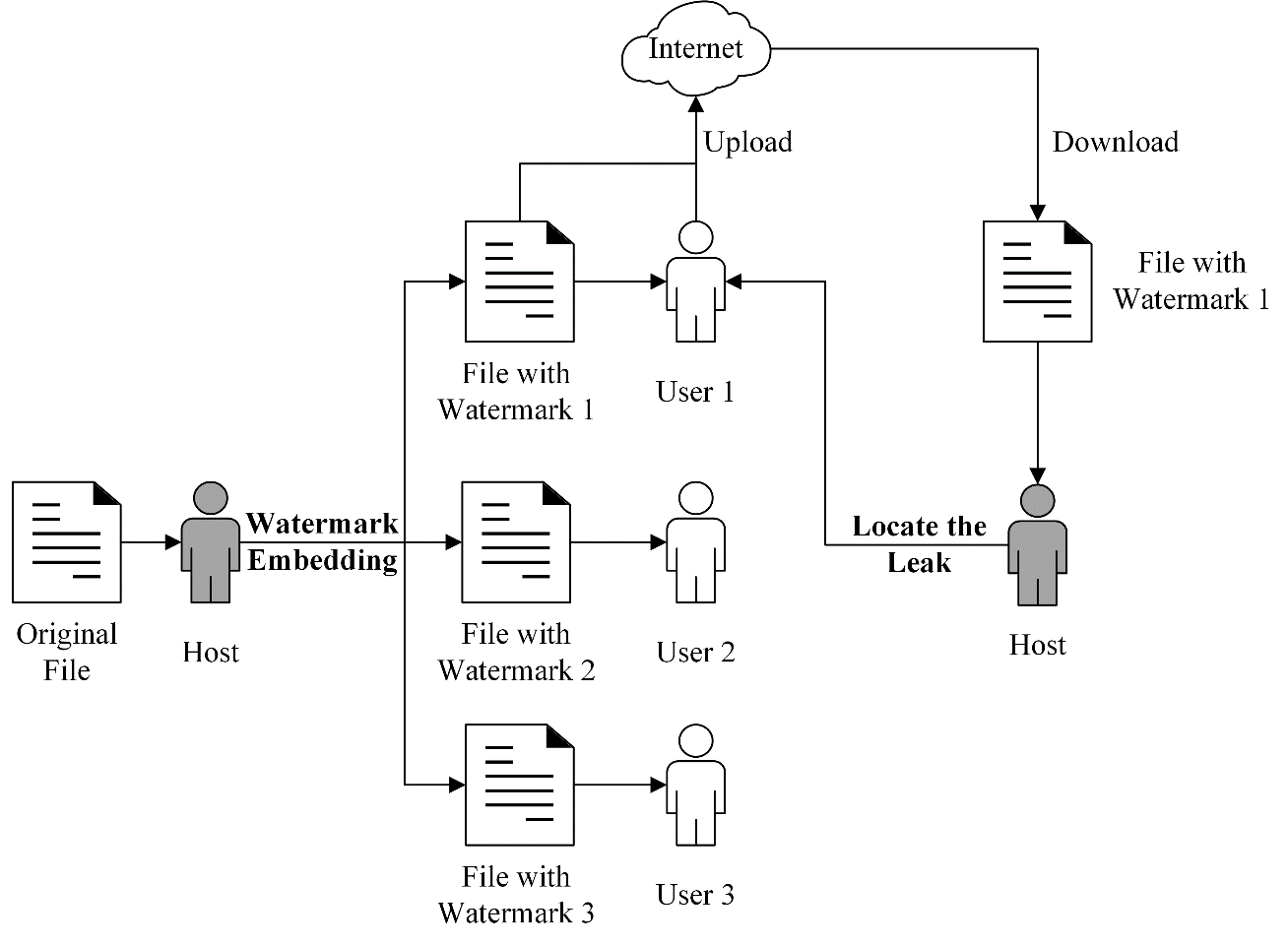


Fig. 1. An application of the proposed scheme

The rest of the paper is organized as follows. In Section 2, we present the embedding and extracting procedures of the proposed scheme. Experimental results are presented in Section 3, and Section 4 concludes the paper.

# 2. Proposed Scheme

In this section, we depict the proposed watermarking scheme for semi-structured data. The overview of embedding and extraction procedure is shown in Fig. 2. The right-going flows represent the procedures of watermark embedding, and the left-going flows represent the procedures of watermark extraction. Note that matrices and sets are written in boldface, and sequences are represented in italic.

Original Data

**T**

Pre-Processing

Embedding

Pairs **E**

Transfer Matrix Construction

Error-Correction Coding

Iterative

Embedding

Marked Data

**TM**

Pre-Processing

Embedding

Pairs **EM**

Iterative

Extraction

Error-Correction Decoding

BP

Decoding

Watermark *W*

Watermark

Private Key

Private Key

Fig. 2. Overview of the embedding and extraction procedure

2.1 Pre-processing

Denote the watermark sequence as *W* with length *lw*. We divide the watermark sequence *W* into *k* segments. . The length of each is denoted as where .

Fig. 3(a) provides an example of a cover JSON file. JSON file employs hierarchical or parallel format for data storage. The elements in a JSON file can be categorized into three types, namely, JSON objects that holds data in a hierarchical way, JSON array that holds elements in a parallel way, and JSON primitives that directly holds data. JSON objects and JSON primitives contain pairs of keys and values while JSON arrays carry parallel data without having their own keys. We use a tree structure to represent the hierarchies of data in JSON, as shown in Fig. 3(b), where the received JSON data is considered the root JSON object, and data are stores in all the leaves.

…

name: John Smith,

address:{

street: 21 2nd Street,

city: New York

},

phone:{

fixedPhone: {

home: 12345678,

office: 87654321},

mobile: 13123456789

},

...

...

name: John Smith,

address\_street: 21 2nd Street,

address\_city: New York,

phone\_fixedPhone\_home:12345678

phone\_fixedPhone\_office:87654321

phone\_mobile:13123456789

...

JsonObject

JsonPrimitive

JsonObject

JsonObject

JsonPrimitive

JsonPrimitive

JsonPrimitive

...

...

JsonArray

...

...

...

(a)

(b)

(c)

Fig. 3. Diagrams for pre-processing (a) cover JSON file, its (b) file structure and (c) embedding pairs after parsing

We parse the cover JSON file, denoted as **J**, into embedding packets **P**, which are the basic units for data embedding in the scheme. Each packet consists of a key and a value. We first initialize **P** as an empty set, and build the tree structure **T** for **J** to get the hierarchical relation. Then we start from **T** and go through all nodes in the tree. If we meet a JSON object during traversal, we append its key to the end of *prefix*, where *prefix* is originally an empty string. Then we append a “\_” in the end to represent the hierarchical relation, and continue to visit the children of this node. If we meet a JSON array, we simply visit the elements in parallel. If we meet a JSON primitive, we append its key to *prefix* . The packet of the corresponding leaf node is *={prefix, value}*, where *value* is the value of that leaf node. Then we conduct backtracking and continue traversing till all the nodes are visited. After parsing, the packets **P={}** that consists of all packets constructed from leaf nodes of the tree. Q denotes the amount of leaf nodes. Fig.3(c) illustrates the parsing result of the file in Fig.3(a).

To avoid semantic changes, we only hide data into packets whose value only contains numeric data. Other packets will be excluded during data embedding, whose values will remain unchanged. Also, the data hider can define a set of packets that are excluded from data embedding by offering their keys. For example, the information of telephone numbers, ids or timestamps should be excluded since they do not tolerate any modification. Therefore, we only embed data into the rest of the packets . We denote the valid pairs as , where *n* is the size of .

Further, to alleviate embedding distortion, data hider can define a maximal-accepted modifying tolerance *T*. Denote respectively the value of and as V() and V(). V() must be within the range of . We first exclude the leading zeros from the digit sequence. Then we further exclude *b* leading digits of to form the embeddable digit sequence

(1)

where represents taking the largest integer smaller than *x*. For example, when the tolerance is set as 0.1, *b*=2, so we exclude the leading two digits from the embedding procedure.

Denote the required embedding length as *s*. We do not embed data into the pair if *len*(V())-b<=s. Here *len*(*x*) represents the length of string *x*. Therefore, the above-mentioned embedding pairs are excluded from as well. The calculation of *s* will be discussed in Section 2.2.

We further apply hashing algorithm on the keys of each so that each pair has a unique hash . The calculation is as follows.

(2)

where takes the ASCII code of the *k*th character of . is a biased parameter, which is set as the private key *K* shared between the data hider and the recipient. is ensured to be unique in the cover data.

Afterwards, we generate a set of sequences , where is pseudo-randomly generated by using as seed, and the elements in are within the range of [0,1]. Finally, the packets in are sorted in ascending order according to the lexicographical order.

2.2. Watermark Embedding

**…**

Watermark Packets

Key of the Pair

Private Key

Hash

Pseudo-Random Sequence *Si*

**…**

Intermediate Packets

Determine *xi* following RSD

Determine *gj,i* following uniform distribution

**…**

To-be-embeded Packets

Watermarked Pairs

**…**

Identify the embedding location

**…**

**…**

**…**

**…**

**…**

Fig. 4. Sketch of watermark embedding procedure

We first construct an intermediate sequence *M* that consisting of *n* segments.. The length of each is also *p*.

Afterwards, we define a transfer matrix **G** as:

(3)

where **G** is a sparse matrix that each in **G** either equals to 0 or 1.

The construction the sparse matrix **G** is specified as follows. First, we generate the cumulative distribution function of robust soliton distribution. The probability density of RSD is defined as follows:

(4)

where is the ideal soliton distribution, and is defined as:

(5)

where denotes the expected ripple size.

For the column of **G**, we flip certain amount of different from zero to one according to the following principle.

(6)

where represents taking the element from sequence *a*, and determines the number of selected that should be flipped. . Therefore, the total numbers of ones in each column also follows the robust soliton distribution (RSD).

We further define the procedure of transferring to as:

(7)

Here, represents XOR operation between matrices, in which each is calculated respectively by:

(8)

Then, the intermediate sequence *M* can be generated.

Afterwards, we iteratively embed *M* into the pairs .For an intermediate packet , we apply cyclic coding as error correction coding. Then the to-be-embedded packet can be generated As

where the length of is denoted as *s* (*s>p)*. Next, we transfer the data format of from numbers to string, and use least significant bit (LSB) modification to conduct data embedding. Denote the embedded data of as .

We then embed *s* bits to a value of an embedding pair. Denote the value is *X* and the to-be-embedded bits are *a*. Then, according to , *a*[*j*] will be embedded in , where , and therefore

(10)

, where *Y* denotes the watermarked value. We repeat this operation until all *s* bits are embedded into the pair and then iteratively conduct the same embedding procedure to all the embedding pairs in the cover file.

Finally, we get the watermarked pairs as . At last, we restore the parsed embedding pairs to the original format. We go through all the nodes of the tree structure. If the values are different from those of , we modify them to . So that the watermarked JSON file is obtained.

*2.3. Watermark Extraction*

When the recipient gets the doubted semi-structured data , the hidden watermark can be detected and extracted. The flowchart of the extracting procedure is depicted in Fig. 5. Note that the recipient is convinced to know the private keys used in watermark embedding for the generation of random sequences, otherwise he cannot perform the watermark extraction.

Parse to the pairs

Filter the pairs and obtain

*n’* pairs

Hash generator

Embedding Location Identification

Cyclic decoding

Cyclic codes?

BP decoding

Extracting

For the embedding pair

Hidden bits

Watermarked Data

Watermark *W*

Y

N

Discard the embedding pair

?

*i*=*i*+1

Watermark Packet

Y

N

Concatenate the watermark packets

Fig. 5. Flowchart of watermark extraction

The recipient first conducts pre-processing, that parses into . He applies the same hashing algorithm on the key and generate the pseudo-random sequence , which is the same as the sequence during data embedding. Afterwards, he starts watermark extraction by initializing **G** as an empty matrix. Then, is used to get , and are used to flip , as introduced in Section 2.2. Then he can get the exact transfer matrix **G** used for data embedding.

Afterwards, the pairs in are sorted in ascending order according to the lexicographical order. For each pair, denote the modified value as Y, and the embedded bits as *a*. According to , *a*[*j*] will be embedded in , where , and therefore the recipient gets the hidden bits by (11):

(11)

Then he applies cyclic decoding on *a* to recover the intermediate sequence from . If is not a valid cyclic code, the recipient is convinced that the data included in the pair is tampered. Thus, he does not utilize the packet for watermark extraction.

For valid packets, he further uses belief propagation (BP) for decoding. He first finds the embedded packet where =1, and the packet is recovered. Then the recovered packets utilized to decrease the degree of corresponding segments. By iteration, all the source packets can be decoded. Finally, he gets the watermark by concatenating all the recovered packets.

*2.4. Extension*

The proposed scheme can be extended on sheet-formatted data as well. We take CSV format for an illustration. The embedding and extraction procedures of CSV watermarking using the proposed method can be concluded by Fig. 2 as well. We denote the data contained in the cell located in the row and column as . Also, we denote the watermark sequence as *W*, and *W* is equally divided into *k* segments. . The length of each is .

In the pre-processing stage, a cover CSV file is first parsed to the data structure of embedding pairs. Denote the sheet has *r* rows and *l* columns, and the rows are regarded as packets. We first choose a column that satisfies the following two requirements: (1) all the elements inside is not empty, and (2) the appearance of repeated elements is the least. The selected column is regarded as index column, denoted as ,. For row of the file, the key of *packet k* is .For the rest cells in the row, we define the collection that contains valid data for watermark embedding, where cells containing non-numeric data are also excluded to avoid semantic changes. Similarly, the data hider can define a collection of banned columns denoted as , and . Therefore, cells that are not included in will remain unchanged during embedding. To alleviate embedding distortion, *b* leading digits of are excluded according to (1) given a threshold *T*. Here, if the length of is less than *b*, is excluded from . We further sort the collection according to the length of the values of the cells. Denote the sorted collection as , where *r* is the remaining number of valid values in row, and *len*()≥*len*() ≥…≥ *len*().Finally, the *packet k* is generated as ={,}, and the valid packets are denoted as , where *q* denotes the amount of valid packets. Invalid packets are defined by (12).

(12)

, where *s* is the minimum required length for embedding. Therefore, the invalid rows will be excluded during watermark embedding. The sketch of pre-processing of a cover CSV file is shown in Fig. 6.

John

Tom

Mary

Lucy

Bill

86001

86002

86003

86004

86005

12

13

11

12

12

462827

802457

282891

586275

482992

3.8172744

21.373

6.7147571

8.4618759

9.1536875

16476575

17547287

14325275

26161678

11547892

86001: 3.8172744, 16476575

86002: 21.373, 17547287

86003: 6.7147571, 14325275

86004: 8.4618759, 26161678

86005: 9.1536875, 11547892

index

banned

invalid

Fig. 6. Sketch of pre-processing of a cover CSV file

We generate hash code for each using (2), where in the equation is replaced with , which is the key of packet *i*. Afterwards, pseudo-random sequences can be generated under the same principle.

The procedures of transfer matrix construction, error correction encoding is similar to that of semi-structured data. In the iterative embedding stage, for *i*th row, the to-be-embedded cyclic code is further fragmented and embedded into . For , we embed bits into the value. We iteratively embed all the bits into until data embedding is finished. Thus, cells with longer value are more likely to carry more additional bits.

On the recipient’s side, data pre-processing for sheets is also carried out ahead of watermark extraction. The recipient gets the embedding pairs and the corresponding pseudo-random sequences. Then he can successfully construct the transfer matrix. Under the same principle, he knows how many bits are hidden in a given , and he extracts the secret bits with the help of the retrieved pseudo-random sequence. He then checks whether the recovered data sequence is a valid cyclic code. Finally, back propagation is also applied for data extraction.

**3. Results and Discussion**

To verify the proposed scheme, we have conducted many experiments on an amount of JSON and CSV files. The sources are provided by some companies that are allowed for experimental uses. We use binary random sequences as digital watermark. We test the watermark robustness to several attacks.

Fig. 7 shows two short JSON files before and after embedding 8 bits watermark as an example. From Fig.7, we can easily observe that the long floating numbers are modified to carry the secret data, while the shorter ones and strings remain unchanged. Thus, we can consider the modification is imperceptible.

Embed 8 bits

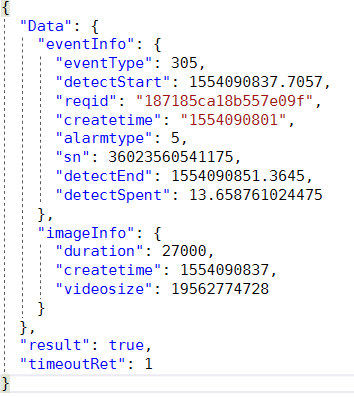


Fig. 7. Embedding Test for a JSON file (a) cover data (b) stego data

*3.1 Settings and Evaluations*

The main parameters in the proposed scheme are the length of each watermark packet *p*, the length of to-be-embedded packet *s*, and two parameters of RSD *c* andas is described in Section 2. In the experiment, we set *p* = 4, *s* = 7, *c* = 0.1, = 0.5.

The proposed scheme is low in computational complexity, which makes it easier to be embedded on hardwares. The watermark embedding and extraction can be done within several seconds by a personal laptop with 2.60 GHz CPU and 8.00 GB RAM.

For an objective performance assessment of the proposed scheme, the overall distortion after watermark embedding can be defined as:

(13)

where the semi-structured data file contains *N* key-value pairs, and equals to 0 or 1 indicates whether the key-value pair has been modified. represents the modified magnitude. =/, where and respectively denotes the original value and the modified value. Specially, if the key-value pair is deleted, is set to 1. A larger *D* indicates that the introduced distortion is larger.

*3.2 Embedding Performances*

Table 1. Capacity vs. Distortion

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Capacity (bits) | 8 | 20 | 40 | 60 | 80 |
| *D* | 0.1078 | 0.0981 | 0.1139 | 0.1069 | 0.1191 |

In Table 1, we measure the introduced distortions according to (13) under different embedding capacities. As is shown, the introduced distortion gradually remains stable even the embedding capacity grows higher. Meanwhile, the distortion of data is quite small, which indicates the watermarking scheme has little impact on the original data.

We further test the robustness of semi-structure data watermarking to three types of attack when *lw*=64 bits. Here, typical attacks including pairs deletion, value modification and redundancy insertion are applied in various degrees. We use four JSON files of different sizes for testing: file1, file2, file3 and file4, each containing 100, 200, 800 and 1500 pairs. We perform the same attack for multiple times on each watermarked file. The ratios of successful extraction under different attacks for semi-structured data is shown in Table 2, where extractions with wrongly retrieved bits are not considered successful extraction. In the table, *Pd*, *Pi*, and *Pm* respectively denotes the percentage of key-value pairs which are deleted, inserted, or tampered in the file.

Table 2. Ratios of successful extraction under different attacks for semi-structured data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of Attack | | Ratios of successful extraction | | | |
| File1.json | File2.json | File3.json | File4.json |
| No Attack | | 10/10 | 10/10 | 10/10 | 10/10 |
| Deletion | *Pd* =5% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pd* =10% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pd* =20% | 10/10 | 8/10 | 10/10 | 10/10 |
| *Pd* =30% | 10/10 | 8/10 | 9/10 | 10/10 |
| Value Modification | *Pm* =5% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pm* =10% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pm* =20% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pm* =30% | 10/10 | 9/10 | 10/10 | 10/10 |
| Insertion | *Pi* =5% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pi* =10% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pi* =15% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pi* =20% | 10/10 | 10/10 | 10/10 | 10/10 |

The proposed watermark scheme is generally robust to all the above-mentioned attacks, even the attacks are strong. Especially, the scheme shows promising embedding performance against pairs deletion and redundancy insertion. Also, files with more valid embedding pairs gradually shows stronger robustness. The main reason of high robustness is that: For pair deletion, the watermark can be extracted using the remaining valid pairs. For contextual modification, the recipient can identify the tampered location using cyclic code checks and discard the tampered pair. For insertion, data extracted from the inserted pairs can hardly pass the cyclic code checks, and thus they are also discarded.

The proposed scheme is also robust to combined attacks, as is shown in Table 3, e.g. the doubted file is a both modified and truncated version of watermarked file. The results prove that the scheme is credible and applicable in real uses.

Table 3. Ratios of successful extraction under combined attacks for semi-structured data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type of Attack | Ratios of successful extraction | | | |
| File1.json | File2.json | File3.json | File4.json | |
| Deletion & Modification (*Pd* =10%, *Pm* =10%) | 10/10 | 10/10 | 10/10 | 10/10 | |
| Deletion & Insertion (*Pd* =10%, *Pi* =10%) | 10/10 | 10/10 | 10/10 | 10/10 | |
| Modification & Insertion (*Pm* =10%, *Pi* =10%) | 10/10 | 10/10 | 10/10 | 10/10 | |

We also test the robustness of the extended part of the proposed scheme. We also use several cover CSV files for copyright watermarking. Here we use a same embedding capacity *lw*=64 bits. The attacks include sorting, row deletion, value modification, row insertion and column insertion. The test is applied on four CSV files. In Table 4, *Pdr* and *Pir* are the percentage of rows which are added or inserted to the file. *Pm* denotes the percentage of values in the sheet which are tampered and *li* denotes the number of inserted columns.

Table 4. Ratios of successful extraction under different attacks for sheets

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of Attack | | Ratios of successful extraction | | | |
| File1.csv (Size:50023) | File2 .csv (Size:10006) | File3 .csv (Size:20002) | File4 .csv (Size:50038) |
| No Attack | | 10/10 | 10/10 | 10/10 | 10/10 |
| Sorting | | 10/10 | 10/10 | 10/10 | 10/10 |
| Row Deletion | *Pdr* =20% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pdr* =40% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pdr* =60% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pdr* =80% | 10/10 | 10/10 | 10/10 | 10/10 |
| Value Modification | *Pm* =20% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pm* =40% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pm* =60% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pm* =80% | 10/10 | 10/10 | 10/10 | 10/10 |
| Row Insertion | *Pir* =10% | 10/10 | 10/10 | 10/10 | 10/10 |
| *Pir* =20% | 10/10 | 10/10 | 10/10 | 10/10 |
| Column  Insertion | *li* =1 | 10/10 | 5/10 | 4/10 | 10/10 |
| *li* =5 | 9/10 | 1/10 | 0/10 | 8/10 |

As is shown in Table 4, the extended part of the proposed scheme also shows great robustness to typical attacks. Similarly, CSV files with more rows are stronger in robustness. As for column insertion, if the inserted values are participates in the payload allocation in each row, the extracted bits are distorted and can not pass cyclic code checks. The robustness against column insertion is restrained in the proposed scheme.

Finally, for the security of the proposed scheme, we use a private key shared by data hider and recipient. Different private key results in different embedding locations, which helps to resist the collusion attack, i.e., two recipients cannot infer the embedding location by comparing two same files with different watermarks.

**4. Conclusion**

In this paper, a novel watermarking scheme for semi-structured data protection is proposed. The cover file is firstly parsed into embedding pairs. We generate a transfer matrix to get the intermediate sequences, which are then encoded using error-correction codes and embedded into the pairs. On the recipient’s side, data extraction can be successfully carried out even the received data are tampered. The proposed scheme can be extended on several other formats. The experimental results show that the proposed scheme is robust to various kinds of typical attacks such as contextual truncating, modification, redundancy addition, etc. Meanwhile, the introduced distortion is comparatively low. Finally, a private key also helps to resist collusion attacks.

**Abbreviations**

XML: Extensible markup language; JSON: JavaScript object notation; SS: Spread spectrum; DFT: Discrete Fourier transform; DCT: Discrete cosine transform; DWT: Discrete wavelet transform; CSV: Comma-separated values; RSD: Robust soliton distribution; BP: Belief propagation.

**Availability of data and materials**

The datasets used and analyzed during the current study are available from the corresponding author on reasonable request.

**Competing interests**

The authors declare that they have no competing interests.

**Funding**

This work is supported by the Natural Science Foundation of China (61572308 and U1736213), the Shanghai Dawn Scholar Plan (14SG36) and the Shanghai Excellent Academic Leader Plan (16XD1401200), National Key Research Development Program of China (2016QY01W0200).

**Authors’ contributions**

Our contributions in this paper were that the first author (JH) participated in the designing of the scheme and drafted the manuscript. The second author (QY) carried out the experiments and participated in the designing of the scheme. The third author (ZQ) conceived of the study, participated in the design and helped to draft the manuscript. The fourth author (GF) and the fifth author (XZ) helped to design and improve the scheme. All authors read and approved the final manuscript.

**Acknowledgements**

Many thanks to the anonymous reviewers for their constructive suggestions to help improving this paper.

**Authors’ information**

Jiahuan He received the B.S. degree from Shanghai University, China, in 2018, where he is currently pursuing the M.S. degree. His research interests include image processing and multimedia security.

Qichao Ying received the B.S. degree from Shanghai University, China, in 2017, where he is currently pursuing the M.S. degree. His research interests include information hiding, image processing, and multimedia security.

Zhenxing Qian received the B.S. and Ph.D. degrees from the University of Science and Technology of China (USTC), in 2003 and 2007, respectively. He is currently a Professor with the School of Computer Science, Fudan University. He has published over 100 peer-reviewed papers on international journals and conferences. His research interests include information hiding, image processing, and multimedia security.

Guorui Feng received the B.S. and M.S. degree in computational mathematic from Jilin University, China, in 1998 and 2001 respectively. He received Ph.D. degree in electronic engineering from Shanghai Jiaotong University, China, 2005. From January 2006 to December 2006, he was an assistant professor in East China Normal University, China. During 2007, he was a research fellow in Nanyang Technological University, Singapore. Now he is with the school of communication and information engineering, Shanghai University, China. His current research interests include image processing, image analysis and computational intelligence.

Xinpeng Zhang received the B.S. degree in computational mathematics from Jilin University, China, in 1995, and the M.E. and Ph.D. degrees in communication and information system from Shanghai University, China, in 2001 and 2004, respectively, where he has been with the faculty of the School of Communication and Information Engineering, since 2004, and is currently a Professor. His research interests include information hiding, image processing, and digital forensics. He has published over 200 papers in these areas.

**References**

[1] Y. Xiang, I. Natgunanathan, D. Peng, et al., Spread spectrum audio watermarking using multiple orthogonal PN sequences and variable embedding strengths and polarities. IEEE/ACM Trans. Audio, Speech, Language Process. 26(3), 529-539 (2018)

[2] Z. Liu, Y. Huang, J. Huang, Patchwork-based audio watermarking robust against de-synchronization and recapturing attacks. IEEE Trans. Inf. Forensics Security. 14(5), 1171-1180 (2019)

[3] G. Hua, J. Goh, V.L.L. Thing, Cepstral analysis for the application of echo-based audio watermark detection. IEEE Trans. Inf. Forensics Security. 10(9), 1850-1861 (2015)

[4] T. Zong, Y. Xiang, I. Natgunanathan, et al., Robust histogram shape-based method for image watermarking. IEEE Trans. Circuits Syst. Video Technol. 25(15), 717-729 (2015)

[5] M. Urvoy, D. Goudia, F. Autrusseau, Perceptual DFT watermarking with improved detection and robustness to geometrical distortions. IEEE Trans. Inf. Forensics Security. 9(7), 1108-1119 (2014)

[6] S.-W. Byun, H.-S. Son, S.-P. Lee, Fast and robust watermarking method based on DCT specific location. IEEE Access. 7, 100706-100718 (2019)

[9] D. Zheng, S. Wang, J. Zhao, RST invariant image watermarking algorithm with mathematical modeling and analysis of the watermarking processes. IEEE Trans. Image Process. 18(5), 1055-1068 (2009)

[10] P.-C. Su, Y.-C. Chang, C.-Y. Wu, Geometrically resilient digital image watermarking by using interest point extraction and extended pilot signals. IEEE Tran. Inf. Forensics Security. 8(12), 1897-1908 (2013)

[12] T. Dutta, H. Gupta, A robust watermarking framework for High Efficiency Video Coding (HEVC)—Encoded video with blind extraction process. J. Vis. Commun. Image Represent. 38, 29-44 (2016)

[13] R. Thanki, K. Borisagar, A technical review of digital image watermarking techniques. Int. J. Adv. Res. Comput. Sci. Softw. Eng. 3(5), 1290-1299 (2013)

[14] M. Amini, M. Ahmad, M. Swamy, A robust multibit multiplicative watermark decoder using vector-based hidden Markov model in wavelet domain. IEEE Trans. Circuits Syst. Video Technol. 28(2), 402-413 (2018)

[15] J. Brassil, S. Low, N. Maxemchuk, et al., Electronic marking and identification techniques to discourage document copying. IEEE J. Sel. Areas Commun. 13(8), 1495-1504 (1995)

[16] L. He, X.-l. Gui, An active attack on chaotic based text zero-watermarking. IEEE Conf. Anthol. 1-4 (2013)

[17] M. L. Mali, N. N. Patil, and J. B. Patil, “Implementation of text watermarking technique using natural language watermarks,” Proc. Int. Conf. Commun. Syst. Netw. Tchnol. 482-486 (2013)

[18] D. Huang, H. Yan, Interword distance changes represented by sine waves for watermarking text images. IEEE Trans. Circuits Syst. Video Technol. 11(12), 1237-1245 (2001)

[19] S. Zhang, Z. Yao, X. Meng, et al., New digital text watermarking algorithm based on new-defined characters, Proc. IEEE Int. Symp. Comput. Consum. Control. 713-716 (2014)

[20] M. Kuribayshi, T. Fukushima, N. Funabiki, Robust and secure data hiding for PDF text document. IEICE Trans. Inf. Syst. 102(1), 41-47 (2019)

[21] I. Kamel, A schema for protecting the integrity of databases. Comput. Secur. 28(7), 698-709 (2009)

[22] A. Khan, S. A. Husian, A fragile zero watermarking scheme to detect and characterize malicious modifications in database relations. Sci. World J., 2013(796726), 1-16 (2013)

[23] R. Agrawal, P. J. Haas, J. Kiernan, Watermarking relational data: framework, algorithms and analysis. The VLDB J. 12(2), 157-169 (2003)

[24] R. Sion, Proving ownership over categorical data, Proc. IEEE Int. Conf. Data Eng. (2004)

[25] R. Sion, M. Atallah, S. Prabhakar, Rights protection for categorical data. IEEE Trans. Knowl. Data Eng. 17(7), 912-926 (2005)

[26] Y. Zhang, B. Yang, X. Niu, Reversible watermarking for relational database authentication. J. Comput. 17(2), 59-66 (2006)

[27] K. Jawad, A. Khan, Genetic algorithm and difference expansion based reversible watermarking for relational databases. J. Syst. Softw. 86(11), 2742-2753 (2013)