Received December 25, 2019, accepted January 6, 2020, date of publication January 15, 2020, date of current version January 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966889

Steganography via E-Books With the EPUB Format by Rearrangements of the Contents of the CSS Files

DA-CHUN WU AND HSIU-YANG SU

Department of Computer and Communication Engineering, National Kaohsiung University of Science and Technology, Kaohsiung 82445, Taiwan

Corresponding author: Da-Chun Wu (dcwu@nkust.edu.tw)

This work was supported in part by the National Science Council, Taiwan, under Grant NSC 101-3113-P-009-006 and Grant NSC 102-2221-E-327-023.

ABSTRACT With the advance of information technology and mobile devices, e-book reading has become a popular way for people to gain knowledge. However, with the widespread uses of digital texts, issues about the copyright protection of e-books have arisen. The electronic publication (EPUB) format is currently one of the most widely used e-book formats. A novel data hiding method for steganographic applications is proposed in this study to hide secret messages into the e-book with the EPUB format. Specifically, the message bits are embedded into the cascading style sheets (CSS) file of the e-book using the lexicographical orders of the selectors and their declarations in the CSS rules. By making use of the fact that the selectors in a CSS file need not all be referred to, fake rules are created artificially, when necessary, to increase the data embedding capacity of the CSS file. The appearance of the e-book does not change after message embedding. Good experimental results and comparisons with other related methods show the feasibility and superiority of the proposed method.

INDEX TERMS Data hiding, steganography, e-book, EPUB format, CSS file.

I. INTRODUCTION

With the recent rapid developments of computer networks and mobile devices, the issue of protection of sensitive or secret information contents for covert communication, safe document achieving, copyright protection of personal works, etc. by steganographic techniques has become more and more important nowadays. Many information hiding techniques via various multimedia [1]–[5] and social media [6], [7] have been proposed to deal with this issue.

Recently, e-books, which are convenient to use anywhere at any time without paper assumption, have a significant influence on people's reading habits. However, owing to the ease of copying digital texts, copyright protection of e-books has become a serious issue. In addition, because of the popularity of e-books, it is a good idea to use the e-book as a type of media for message communication. Furthermore, keeping e-books securely without being stolen or copied illicitly is also an important task. Owing to these reasons, in this study we try to investigate solutions to these issues, and a new steganographic technique via the use of the e-book is proposed.

For example, for copyright protection of the e-book content using the steganographic technique, the main issue is to prevent the digital content of the e-book from being copied illegally by hackers. To solve this problem, it is desired to embed certain secret copyright-related information into the e-book to be protected to yield a steganographic effect without arousing the hacker's notice. For this aim, in the steganographic method proposed in this study, we embed such information into the cascading style sheet (CSS) file of an e-book with the electronic publication (EPUB) format without changing the e-book appearance. Specifically, we rearrange the constituent elements, called selectors and declarations, in the rules of the CSS file into specific orders to represent the message data in the form of *multiple-based numbers* [8]–[10]. Furthermore, a CSS rule with its selector *never referred to* by the extensible hypertext markup language (XHTML) file of the e-book is regarded *valid*, i.e., is not considered as an error against the syntax of the XHTML. This characteristic is utilized as well

The associate editor coordinating the review of this manuscript and approving it for publication was Mehul S. Raval.

by the proposed method, whenever necessary, to increase the embedding capacity of the e-book. The related theory of multiple-based numbers [8]–[10] used to represent the message data to be embedded is elaborated by illustrative examples, followed by detailed algorithms to implement the proposed method. Experimental results showing the correctness of the algorithms and the feasibility of the proposed method are also included.

In the remainder of this paper, at first a review of related studies, including a literature review and an introduction to the structure of the e-book with the EPUB format, is given in Section II. Then, the ideas behind the proposed data hiding method via the CSS file are presented in Section III by a number of illustrative examples. The data embedding and extraction processes which implement the proposed method are presented in Section IV, followed by some experimental results in Section V. Some concluding remarks and discussions are given in Section VI.

II. RELATED STUDIES

In this section, a survey of related works is given at first, followed by a review of the structure of the e-books with the EPUB format, which are used for data hiding in this study.

A. RELATED WORKS

There are very few studies on information hiding in the CSS files of e-books, although a few data hiding techniques via the hypertext markup language (HTML) files of web pages have been proposed in the past two decades. It is noted that both the CSS and HTML are markup languages. Some special space codes in the HTML appear as white spaces in the web page just like the commonly-seen hexadecimal space code '20' of the American Standard Code for Information Interchange (ASCII). Lee and Tsai [11] use various combinations of such special space codes to encode message bits in triplets, creating a steganographic effect. In addition, the names of tags and attributes in the HTML can be written both in capital letters or in small ones, all with identical meanings. Dey, Al-Qaheri, and Sanyal [12] use this characteristic to hide message data in the HTML document with the capital and small letters representing the bits of '1' and '0', respectively. However, this technique cannot be used for the XHTML file used by the e-book because the capital and small letters are of different meanings in the XHTML specification.

In more details about the use of the tag in the HTML and other markup languages, a logical document component, called *element*, either begins with a *start-tag* and ends with a matching *end-tag*, or consists of only an *empty-element tag*. The characters between the start-tag and end-tag, if any, are the element's *content*. An *empty-element tag* may be regarded as a combination of the start-tag and the end-tag; for example, the pair of
 and </br> may be combined to be just
 . Inoue *et al.* [13] use alternatively a pair of a start-tag and its matching end-tag as well as its corresponding empty-element tag to embed binary data. Some elements in a markup language have attributes as sub-elements, which are put inside the start-tag to control the element's behavior. The order of attributes in the tag does not affect the effect of the tag. This characteristic is utilized by several existing methods, like Artz [14] who shows examples of steganography using data ordering of attributes; Jie [14] who turns a secret message into a decimal integer, and then creates the corresponding *equal elements* through the sub-elements' permutation and combination in the XML document to embed the integer; Wu and Lai [15] who use the lexicographical order of the attributes to encode secret information; and Huang *et al.* [16] who permute attributes in the tag to embed messages.

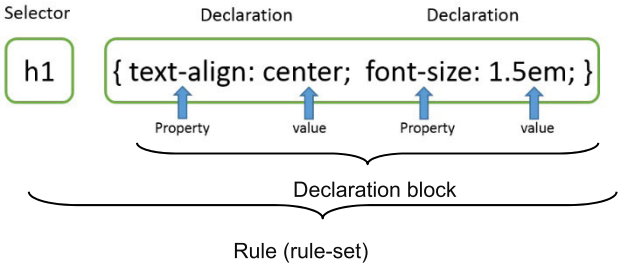


FIGURE 1. A description of the structure of a CSS rule (or rule-set) [22].

Among the few studies on using the CSS file for data hiding, one example is the method proposed by Kabetta *et al.* [17] which inserts spaces or tabs after semicolon characters at the ends of the lines of the CSS style properties to embed messages. Another is Lai *et al.* [18] which uses different settings of CSS attributes of margin and padding to encode secret message bits.

B. STRUCTURE OF THE E-BOOK WITH THE EPUB FORMAT

In this section, a brief review of the structure of the e-book with the EPUB format is given, with the content of the CSS file of the e-book described in detail.

EPUB is an e-book format that uses the ".epub" file extension. The term is short for electronic publication as mentioned before and is sometimes styled ePub [19]. EPUB is a free, open standard format with the reflowable capability. Reading devices can automatically wrap or change the font size so that the content can be displayed in a suitable manner for easy visual inspection.

A decompressed version of an e-book with the EPUB format contains a variety of different files [20]. Each of these files has its own purpose, for example, the XHTML file for describing the main elements of the e-book, and the CSS file for defining the various display properties in the XHTML file.

A CSS file [21], [22] consists of a list of *rules* (or *rule-sets)*, and each rule consists of a *selector* and a *declaration block*. The declaration block contains one or more *declarations* separated by semicolons (;), with each declaration including a *property name*, a colon (:), and a *value* in sequence. Figure 1 illustrates the description of a rule in the CSS file.

When an XHTML file makes reference to its CSS files, the user agent (UA) will obtain the rules and their declaration blocks from the CSS files. According to the syntax of

```
.formalpara-title {
 font-weight: bold;
}
div.blockquote-title {
 font-weight: bold;
 margin-top: 1em;
 margin-bottom: 1em;
}
span.msgmain-title {
 font-weight: bold;
}
nav ol li a {
text-decoration: none;
color: black;
font-family: sans-serif;
}
#guide {
 display: none;
}
```

FIGURE 2. Part of the rules in a css file of the e-book "gulliver's travels" written by jonathan swift.

the CSS, the *appearance orders* of the rules as well their selectors and declarations do *not* affect the effect of the CSS file. The data embedding method proposed in this study is based mainly on this characteristic. In addition, if an *undefined* selector is referenced by the XHTML file, the operation will be considered invalid. However, if a certain selector defined in the CSS are *never* referenced by the XHTML file, it will *not* be considered invalid. The data hiding method proposed in this paper takes advantage of this second characteristic to add to the CSS file extra rules that are *never* referenced to increase the data hiding capacity. The added rules are called *fake rules* subsequently in this paper. Figure 2 shows part of the rules in a CSS file of the e-book ''Gulliver's Travels'' written by Jonathan Swift.

III. IDEAS OF PROPOSED DATA HIDING METHOD via CSS FILES

In this section, the ideas of the proposed data hiding method is described, with a series of examples given to illustrate the presented ideas.

A. ENCODING SELECTORS AND DECLARATIONS TO REPRESENT MESSAGES

Simply speaking, the proposed method *encodes* selectors and declarations in the CSS file to represent message data to achieve data hiding. That is, if the selectors (or declarations) are *lexicographically arranged exhaustively to form a list of k items*, then the $(i+1)$st item in the list is regarded to correspond to a message with a base$-k$ value of i, or equivalently, a message of base$-k$ value i is said to be embedded in the form of the $(i+1)$st item in the list, as illustrated by the following examples.

Example 1: Suppose that A, B, and C denote three selectors in a CSS file. Here single letters are used to denote selectors for simplicity of concept, but actually each selector in a commonly-seen CSS file is usually a sequence of letters, like **'div.msgexplan'**, **'span.msgexplan-title'**, etc. A complete lexicographical arrangement of the three selectors results in an *ordered* list of six items, $L = \{ABC, ACB, BAC, BCA, CAB, CBA\}$. These items can be used to represent *six base-6 numbers*, $0_6, 1_6, \ldots, 5_6$. That is, the three selectors A, B, and C may be *encoded* to represent the six base-6 numbers 0_6 through 5_6. Note that 6 is the number of permutations of 3 objects, i.e., $6 = 3!$. On the other hand, with 6 choices of the lexicographically rearranged items in L, a message of at most 2 bits (not 3 bits) can be embedded into an item of the six because $2 = \lfloor \log_2 6 \rfloor = \lfloor \log_2(3!) \rfloor$ where $\lfloor \cdot \rfloor$ denotes the integer floor function.

The above example can be generalized to be the fact that n selectors may be encoded to represent $n!$ base-$(n!)$ numbers, $0_{(n!)}, 1_{(n!)}, \ldots, (n!-1)_{(n!)}$, and a message of at most $\lfloor \log_2(n!) \rfloor$ bits can be embedded into the $(n!)$-item lexicographical rearrangement list of the n selectors.

*Example 2:*As a continuation of Example 1, suppose that the selector A has four declarations, a, b, c, and d; and each of B and C has only one declaration. Then, by the equality $4! = 24$, an ordered list of lexicographical arrangement of 24 items, $L\prime = \{abcd, abdc, \ldots, dcba\}$, can be obtained, which can be used to represent 24 *base-24 numbers*, $0_{24}, 1_{24}, \ldots, 23_{24}$.

In addition, if the three selectors A, B, C, and the four declarations a, b, c, d of A are used *integrally*, then $(4!)\times(3!) = 24 \times 6 = 144$ items can be set up to represent 144 *two-based numbers*, namely, $0_{24}0_6, 0_{24}1_6, \ldots, 23_{24}5_6$ (or $0_{4!}0_{3!}, 0_{4!}1_{3!}, \ldots, 23_{4!}5_{3!}$), as can be figured out. For this case, a message of at most 7 bits can be embedded because $\lfloor \log_2((4!)\times(3!)) \rfloor = \lfloor \log_2 144 \rfloor = 7$. ■

*Example 3:*As a simple example but with real CSS rules to illustrate the above-mentioned basic idea of data embedding proposed in this study, suppose that two rules as follows are listed in a CSS file:

$$\text{div.msgexplan}\{\text{margin}-\text{bottom}:1\text{em};\ \text{margin}-\text{left}:1\text{em};\ \text{margin}-\text{top}:1\ \text{em};\}$$

$$\text{span.msgexplan}-\text{title}\{\text{font}-\text{weight}:\text{bold};\}$$

where the first rule has three declarations, and the second has one. According to the discussion of Example 2 above, these rules and their declarations can be rearranged to generate a list of 12 items to represent 12 two-based numbers, $0_30_2, 0_31_2, \ldots, 5_31_2$ because $(3!)\times(2!) = 12$, as shown in Table 1. ■

B. TRANSFORMING THE MESSAGE INTO A MULTIPLE-BASED NUMBER

According to the above discussions, if a *decimal-valued* message is to be embedded, at first it should be transformed into a *multiple-based number* [8]–[10] as done in this study. And this can be carried out by successive divisions, as illustrated by the following example.

Example 4: As a continuation of Example 2, let the message to be embedded be the decimal value 141_{10}, which, after being transformed into a number with the two bases of 24 (= 4!) and 6 (= 3!), becomes $23_{24}3_6$ because we can apply successive divisions to 141_{10} to get $\lfloor 141/6 \rfloor = 23$

TABLE 1. A list of 12 items generated from rearrangements of two selectors, one with three declarations and the other with one, which can represent 12 two-based numbers $0_{3!}0_{2!}, 0_{3!}1_{2!}, \cdots, 5_{3!}1_{2!}$.

Item No.	Represented two-based number	Items of rearrangements of selectors & declarations
1	$0_{(3!)}0_{(2!)}$	div.msgexplan { margin-bottom: 1em; margin-left: 1em; margin-top: 1em; } span.msgexplan-title { font-weight: bold; }
2	$0_{(3!)}1_{(2!)}$	span.msgexplan-title { font-weight: bold; } div.msgexplan { margin-bottom: 1em; margin-left: 1em; margin-top: 1em; }
3	$1_{(3!)}0_{(2!)}$	div.msgexplan { margin-bottom: 1em; margin-top: 1em; margin-left: 1em; } span.msgexplan-title { font-weight: bold; }
4	$1_{(3!)}1_{(2!)}$	span.msgexplan-title { font-weight: bold; } div.msgexplan { margin-bottom: 1em; margin-top: 1em; margin-left: 1em; }
. . .	. . .	. . .
12	$5_{(3!)}1_{(2!)}$	span.msgexplan-title { font-weight: bold; } div.msgexplan { margin-top: 1em; margin-left: 1em; margin-bottom: 1em; }

with a remainder of 3 and $\lfloor 23/24 \rfloor = 0$ with a remainder of 23. Reverse to the divisions, the following equalities can be figured out to be true:

$$141_{10} = 23_{24}3_6 = 23 \times 6 + 3 = 23 \times 3! + 3.$$

Subsequently, the base-24 number 23_{24} can be embedded into the CSS file by *encoding* the four declarations to be in the form of the *24th* item (not the 23th one), namely, *dcba*, in the 24-item lexicographical rearrangement list of the declarations; and the base-6 number 3_6 can be embedded by *encoding* the three selectors to be in the form of the *4th* item (not the 3rd), namely, *BAC*, in the 6-item lexicographical rearrangement list of the selectors.

C. A GENERAL CASE OF MESSAGE EMBEDDING

From the above examples, it can be seen that in general, n selectors $S_1, S_2, \ldots, S_n$ with each S_i having q_i declarations $p_{i1}, p_{i2}, \ldots, p_{iqi}$ in a CSS file C can be encoded to represent N *multiple-based numbers* where

$$N = q_n! \times q_{n-1}! \times \ldots \times q_1! \times n!. \tag{1}$$

And a message to be embedded in the form of a decimal value I_{10} may be transformed in advance by successive divisions into an equivalent $(n+1)$-digit *multiple-based number* I_{mb} as follows:

$$I_{\text{mb}} = D_n D_{n-1} \ldots D_1 D_0 \tag{2}$$

where

(a) D_i with $i = 1, 2, \ldots, n$ is the base-$(q_i!)$ component of I_{mb} like the base-(4!) component 23_{24} in the two-based number $23_{24}3_6$ mentioned in the above example; and

(b)D_0 is the base-$(n!)$ component of I_{mb} like the base-(3!) component 3_6 in $23_{24}3_6$ above.

Then, embedding of I_{mb} into the CSS file may be carried out by the following two steps:

(a) embedding each base-$(q_i!)$ number D_i, $i = 1, 2, \ldots, n$, into the q_i declarations of the $i-$th selector S_i as the (D_i+1)st item in the $(q_i!)$-item lexicographical rearrangement list of the q_i declarations; and

(b) embedding the base-$(n!)$ number D_0 into the n selectors as the $(D_0 + 1)$st item in the $(n!)$-item lexicographical rearrangement list of the n selectors.

Furthermore, the successive divisions used to transform the decimal message I_{10} into the $(n + 1)$-digit multiple-based number $I_{\text{mb}} = D_n D_{n-1} \ldots D_1 D_0$ as described by Eq. (2) can be expressed as

$$\begin{aligned} I_{10} &= D_n \times (q_{n-1}! \times \ldots \times q_1! \times n!) + \ldots + D_2 \times (q_1! \times n!) \\ &\quad + D_1 \times n! + D_0 = D_n \times (q_{n-1}! \times \ldots \times q_1! \times q_0!) \\ &\quad + \ldots + D_2 \times (q_1! \times q_0!) + D_1 \times q_0! + D_0 \\ &= \sum_{i=1}^{n} [D_i \times (\prod_{j=0}^{i-1} q_j!)] + D_0 \end{aligned} \tag{3}$$

where $q_0!$ is defined as $n!$; and D_0 through D_n actually are the remainders generated by the intermediate steps of the successive division process as can be seen in the next illustrative example (Example 5). It is noted by the way that the remainder of 'a number a divided by another b' is in fact just the value yielded by the modulo operation 'a mod b'.

Finally, if the message to be embedded into the selectors and declarations is binary instead of decimal, then it can be figured out from the reasoning conducted in Example 3 that at most the following number of bits can be embedded into the CSS file:

$$m = \lfloor \log_2 N \rfloor \tag{4}$$

where N is computed by Eq. (1) above. The two equality of (1) and (4) may be combined into a single one as follows, which will be used in the proposed data hiding algorithm described later:

$$m = \lfloor \log_2 (q_n! \times q_{n-1}! \times \ldots \times q_1! \times n!) \rfloor \tag{5}$$

A more general case than Example 4 is given in the following to illustrate the proposed message embedding idea more completely.

Example 5: Suppose that there are three selectors, S_1, S_2, and S_3, which have three, four, and two declarations, respectively. Then, by Eq. (1), totally the following number of lexicographical rearrangement items can be generated to embed various message values:

$$N = 2! \times 4! \times 3! \times 3! = 1728.$$

If the message to be embedded is $I_{10} = 1633_{10}$, then by Eq. (2) it can be transformed by successive divisions into the

following 4-digit form:

$$I_{\text{mb}} = D_3 D_2 D_1 D_0$$

where $D_0 = 1$ because $\lfloor 1633/(3!) \rfloor = \lfloor 1633/6 \rfloor = 272$ with a remainder of 1; $D_1 = 2$ because $\lfloor 272/(3!) \rfloor = 45$ with a remainder of 2; $D_2 = 21$ because $\lfloor 45/(4!) \rfloor = 1$ with a remainder of 21; and $D_3 = 1$ as $\lfloor 1/(2!) \rfloor = 0$ with a remainder of 1. Note here that each remainder may be obtained by a modulo operation, e.g., $D_0 = 1 = (1633) \bmod (3!) = 1$. As a result, the four-based number I_{mb} equivalent to the decimal message $I_{10} = 1633_{10}$ is

$$I_{mb} = D_3 D_2 D_1 D_0 = 1_{(2!)} 21_{(4!)} 2_{(3!)} 1_{(3!)} = 1_2 21_{24} 2_6 1_6.$$

According to Eq. (3), the above successive divisions can be expressed integrally as

$$\begin{aligned} I_{10} &= \sum_{i=1}^{n} [D_i \times (\prod_{j=0}^{i-1} q_j!)] + D_0 \\ &= D_3 \times (q_2! \times q_1! \times q_0!) + D_2 \times (q_1! \times q_0!) + D_1 \times q_0! \\ &\quad + D_0 = 1 \times (4! \times 3! \times 3!) \\ &\quad + 21 \times (3! \times 3!) + 2 \times (3!) + 1 \\ &= 864 + 756 + 12 + 1 \\ &= 1633. \end{aligned}$$

Finally, binary messages which can be embedded into the selectors and declarations of the CSS file are at most with the following number of bits according to Eq. (4):

$$m = \lfloor \log_2 1728 \rfloor = 10.$$

Done. ■

IV. DATA EMBEDDING AND EXTRACTION PROCESSES

A. BASIC DATA HIDING CONCEPT

In the above discussions, there is no mention about the detail of how to embed a given multiple-based number I_{mb} into a sequence V of selectors (or declarations) as a certain item M in the list of the lexicographical rearrangements of the selectors (or declarations), or equivalently, about the detail of how to generate *directly* the desired item M which represents the number I_{mb} from the selector (or declaration) sequence V. An example of the solution proposed in this study to this problem is illustrated by the following example.

*Example 6:*Suppose that there are four selectors, A, B, C, and D from which a list L of 4!, or equivalently, 24 items of the lexicographical rearrangements of the selectors can be generated, as shown in Table 2.

Suppose that it is desired to embed the base-24 number $D_b = 15_{24}$ into the four selectors. Then, according to the previous discussions, the four selectors should be rearranged to be in the form of the $(15 + 1)$st, or equivalently, the 16th item M, namely, *CBDA*, in the list L as shown in Table 2. The problem in concern here is how this item M can be generated *automatically without constructing the entire list L in advance*. A solution found in this study is to conduct

TABLE 2. A ordered list of 24 items of lexicographical rearrangements of four selectors *A*, *B*, *C*, and *D*.

Item No.	Lexicographically rearranged item	Item No.	Lexicographically rearranged item
1	*ABCD*	13	*CABD*
2	*ABDC*	14	*CADB*
3	*ACBD*	15	*CBAD*
4	*ACDB*	**16**	***CBDA***
5	*ADBC*	17	*CDAB*
6	*ADCB*	18	*CDBA*
7	*BACD*	19	*DABC*
8	*BADC*	20	*DACB*
9	*BCAD*	21	*DBAC*
10	*BCAD*	**22**	***DBCA***
11	*BDAC*	23	*DCAB*
12	*BDCA*	24	*DCBA*

successive divisions as well but in the following way. At first, put the selectors into an ordered set S according to the *lexicographical order* of their names, resulting in $S = \{A, B, C, D\}$. Then, based on the permutation $4! = 4 \times 3 \times 2 \times 1$, perform the following steps to conduct *successive divisions* to obtain the desired item M which is preset to be an empty string initially.

(1) Divide $D_b = 15_{24}$ by 3! to get a quotient of $Q_1 = \lfloor 15/(3!) \rfloor = 2$ with a remainder of $R_1 = 15 \bmod (3!) = 3$, so that the $(Q_1 + 1)$st, or equivalently, the 3rd element in S, namely, C, is removed from S at first and appended to the end of string M to get $M = C$ with S becoming $S = \{A, B, D\}$.

(2) Divide the remainder $R_1 = 3$ by 2! to get a quotient of $Q_2 = 1$ with a remainder of $R_2 = 1$, so that the $(Q_2 + 1)$st, or equivalently, the 2nd element in S, namely, B, is removed from S and appended to the end of M to get $M = CB$ with S becoming $S = \{A, D\}$.

(3) Divide the remainder $R_2 = 1$ by 1! to get a quotient of $Q_3 = 1$ with a remainder of $R_3 = 0$, so that the $(Q_3 + 1)$st, or equivalently, the 2nd element in S, namely, D, is removed from S and appended to the end of M to get $M = CBD$ with S becoming $S = \{A\}$.

(4) Divide the remainder $R_3 = 0$ by 0! to get a quotient of $Q_4 = 0$ so that the $(Q_4 + 1)$st, or equivalently, the 1st element in S, namely, A, is removed from S and appended to the end of M to get $M = CBDA$ with S being emptied.

Done. ■

The process of embedding a base-$(n!)$ number D_b into a sequence of n selectors (or declarations) as the form of the $(D_b + 1)$st item in the $(n!)$-item list of the lexicographical rearrangements of the selectors (or declarations) as illustrated in the last example can be generalized to be an algorithm as described in Algorithm 1.

Algorithm 1 will be used as a subroutine in the data embedding algorithm described later.

Algorithm 1 Embedding a Base-($n!$) Number D_b into a Sequence of n Selectors (or Declarations)

Input:
(1) a sequence V of n selectors (or declarations) $\{V_1, V_2, \ldots, V_n\}$ already put in their lexicographical order;
(2) a base-($n!$) number D_b to be embedded into V.
Output: an ordered rearrangement of the n selectors (or declarations) in V which is the (D_b+1)st item in the $(n!)$-item list of the lexicographical rearrangements of the selectors (or declarations).
Steps.
Step 1: *//Initialization*
1.1 establish an *ordered sequence* V_b, set to be empty initially;
1.2 define an *initial remainder* $R_0 = D_b$.
Step 2: *//Embedding the base-(n!) number*
for $i = 1$ to n
compute quotient $Q_i = \lfloor R_{i-1}/(n-i)! \rfloor$ and remainder $R_i = R_{i-1} \bmod (n-i)!$;
remove the (Q_i+1)st element V_{Qi+1} from input sequence V;
append V_{Qi+1} to the end of V_b;
end for.
Step 3: *//Ending of the algorithm*
take the final V_b as the desired ordered rearrangement of the n selectors(or declarations) originally in V and exit.

Algorithm 2 Extracting a Base-($n!$) Number D_b From a Sequence of Lexicographically Rearranged n Selectors (or Declarations)

Input: an ordered sequence V of n selectors (or declarations) $\{V_1, V_2, \ldots, V_n\}$ which is one of the $n!$ items of the list of all the lexicographical rearrangements of V_1 through V_n.
Output: the base-($n!$) number D_b which is represented by V.
Steps.
Step 1: *//Initialization*
1.1 put the n selectors (or declarations) in the ordered sequence V into their lexicographical order to establish another *ordered* sequence $V' = \{V'_1, V'_2, \ldots, V'_n\}$;
1.2 define D_b as the base-($n!$) number to be extracted,set to be zero initially.
Step 2: *//Extracting the base-(n!) number*
for $i = 1$ to $n-1$
if V_i in input sequence V is the $j-$th element V'_j in sequence V', **then**
compute $v = (j-1) \times (n-i)!$;
end if;
set $D_b = D_b + v$;
end for;
Step 3: *//Ending of the algorithm*
take the final value of D_b as the desired base-($n!$) number represented by V and exit.

In addition, reversely after a multiple-based number is embedded into a sequence of selectors (or declarations) as a certain item in the list of the lexicographical rearrangements of the selectors (or declarations), it is also necessary to have a way to extract directly the multiple-based number for use in the data extraction process. The following is an illustrative example of the solution proposed in this study to this problem.

Example 7: As a reverse of the problem described in Example 6 where four selectors, A, B, C, and D are used, suppose that an item of the rearrangement of the selectors, namely, *DBCA*, is found in the CSS file. It is desired to find out *automatically* which base-(4!) number D_b is embedded in the item *DBCA*. For this, let $n = 4$ to represent the number of selectors here, set $D_b = 0$ initially, and put the selectors in *DBCA* into their lexicographic order to form an ordered set $S = \{A, B, C, D\}$. Then, conduct *successive additions* in the following way to find out the desired value of D_b.

(1) The 1st element in *DBCA* is D which is the 4th element in S, so add the value $(4-1) \times (n-1)! = 3 \times 3! = 18$ to D_b so that $D_b = 0 + 18 = 18$. Also, remove D from S, resulting in a new $S = \{A, B, C\}$.
(2) The 2nd element in *DBCA* is B which is the 2nd element in the current $S = \{A, B, C\}$, so add the value $(2-1) \times (n-2)! = 1 \times 2! = 2$ to D_b to get $D_b = 18 + 2 = 20$. Also, remove B from S, resulting in a new $S = \{A, C\}$.
(3) The 3rd element in *DBCA* is C which is the 2nd element in the current $S = \{A, C\}$, so add the value $(2-1) \times (n-3)! = 1 \times 1! = 1$ to D_b to get a new $D_b = 20 + 1 = 21$. Also, remove C from S, result in a new $S = \{A\}$.
(4) The 4th element in *DBCA* is A which is the 1st element in the current $S = \{A\}$, so add $(1-1) \times (n-4)! = 0 \times 0! = 0$ to D_b to get $D_b = 21+0 = 21$. In practice, this step may be omitted because the value of D_b is not changed.

Therefore, the (4!)-based number embedded in *DBCA* is $D_b = 21$, implying that the item *DBCA* is the $(21+1)$st, or equivalently, the 22nd item in the 24-item list of the lexicographical rearrangements of the four selectors A, B, C and D, which is correct as can be seen from Table 2. Done.

The process of extracting the base-($n!$) number embedded in an item of the ($n!$)-item list of the lexicographical rearrangements of n selectors (or declarations) as illustrated in the last example can be generalized as Algorithm 2.

Algorithm 2 will be used as a subroutine in the data extraction algorithm described later.

B. ENLARGING DATA EMBEDDING CAPACITY BY FAKE RULES

From the previous discussions, it can be seen that the more selectors and declarations are in the CSS file, the more mes-

sage bits can be embedded. In this study, *arbitrary-precision arithmetic* [23] is used to handle the big permutation values $n!$ and $q_i!$ appearing in Eq. (1) for large n and q_i to avoid computation overflows and make a full use of the embedding capacity of the selectors and declarations.

However, when there is just a *small* number of rules in the CSS file with limited numbers of selectors and declarations, the data embedding capacity might be insufficient. In such a case, it is proposed in this study to add extra rules, called *fake rules* as mentioned previously, to the CSS file to increase the data embedding capacity of the resulting CSS file. These fake rules are collected as a *fake-rule pool* in advance in this study from the CSS files of a set of real e-books published by various companies. Therefore, they look like normal rules when added. Furthermore, each of them, before being added, is checked against those rules existing in all the CSS files of the e-book to make sure that its selector does not appear in the CSS files. This is necessary to guarantee that each fake rule, after being added, will not be referenced by the XHTML file to yield effects in the resulting appearance of the e-book. In short, the fake rules, though looking *normal*, are *unused* in the resulting e-book, and so have steganographic effects.

Table 3 shows some of the rules used in this study, where Part (a) of the table shows part of the rules listed in the e-book "Alice's Adventures in Wonderland" by Lewis Carroll, which was one of the e-books tested in the experiments conducted in this study, and Part (b) shows part of the rules in the fake-rule pool constructed in this study, which are used to increase the data embedding capacity when necessary, as just mentioned.

TABLE 3. Some rules used in this study. (a) Part of the rules in the e-book "Alice's Adventures in Wonderland" by Lewis Carroll. (b) Part of the rules in the fake-rule pool for use to increase the data embedding capacity when necessary.

	Rules	#declarations
(a)	.caption {font-weight:bold;margin-bottom:2em;text-align:center;}	3
	.center {text-align:center;}	1
	hr {clear:both;margin-bottom:2em;margin-left:auto;margin-right:auto;margin-top:2em;width:33%;}	6
	img {border-style:none;}	1
	p {margin-bottom:0.75em;margin-top:0.75em;text-align:justify;}	3
(b)	#epb-logo {margin:1.5em 0; padding:0;text-align:center;border: 0;}	4
	.calibre1 {display:block;font-family:serif;line-height:130%;margin-bottom:0;margin-left:0;margin-right:0;margin-top:0;padding-bottom:0;padding-left:0;padding-right:0;padding-top:0;text-align:justify;}	12
	.calibre2 {display: block;font-size:1.66667em;font-weight:bold;line-height:130%;margin-bottom:0.67em;margin-left:0;margin-right:0;margin-top:0.67em;text-align:center;}	9
	.CI {text-align:center;margin-top:0px;margin-bottom:0px;padding:0px;}	4

C. DATA EMBEDDING PROCESS

The proposed method embeds message data into a CSS file of an e-book with the EPUB format without changing the original appearance of the e-book. More specifically, at first the e-book is decomposed into CSS files and others. Next, a preprocessing procedure is applied on a CSS file selected for data embedding to merge each set of those selectors all with an identical name into a single selector, and to delete duplicate declarations associated with the selector [21]. Also, each resulting selector together with its associated declarations is rewritten as a single line; and the comments and unnecessary blank spaces in the CSS file are deleted, into order to reduce the storage required to keep the file. Furthermore, a random number generator controlled by a secret key is used to randomize the bits in the input message and the result is then converted into a large integer number. Then, the embedding capacity of the preprocessed CSS file is estimated, and if it is insufficient to embed the input message bits, fake rules are added to enlarge the capacity. Finally, the randomized message bits are transformed into a multiple-based number which is then embedded by the way of rearranging the declarations and selectors into proper lexicographical orders, yielding a stego-CSS file as the output. A flowchart of these major steps of the proposed data embedding process is shown in Fig. 3.

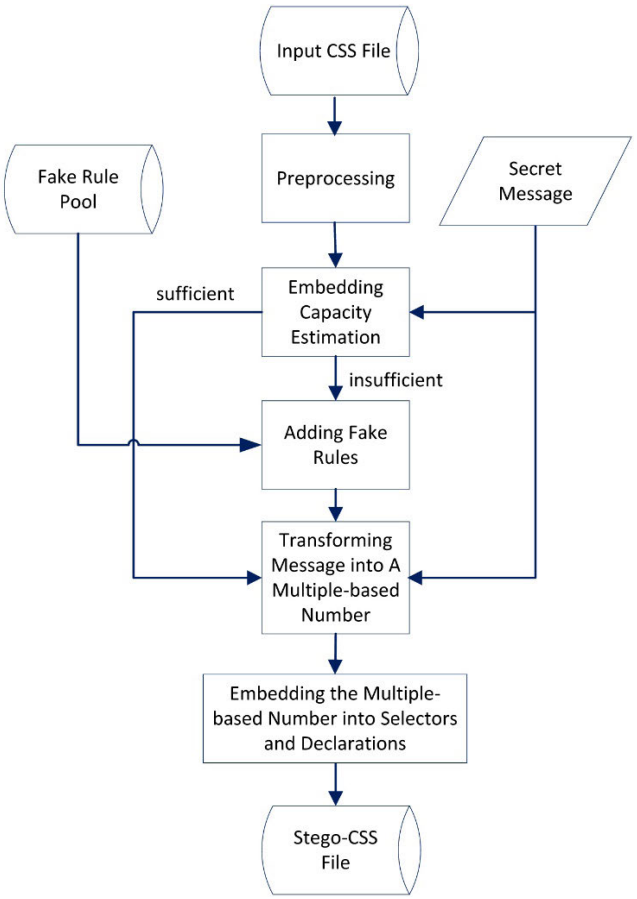


FIGURE 3. A flowchart of the proposed data embedding process.

The details of the above-described message data embedding process are presented as Algorithm 3.

Algorithm 3 Data Embedding

Input:
(1) a binary message$T$ with t bits;
(2) a CSS file C of an e-book B;
(3) a fake-rule pool P;
(4) a secret key K; and
(5) a random number generator (RNG) G.
Output: a stego-CSS file of B with T embedded in C.
Steps.
Step 1: *//Initialization and preprocessing*
1.1 initialize $i = 0$ and define $C' = C$;
1.2 **repeat**
set $i = i + 1$;
scan sequentially C' to find selectors all with an identical name,take them out of C', and put them into a set named S_i;
merge all declarations in S_i as a set U_i and delete duplicate ones in U_i;
regard S_i as a single selector with its declarations taken to be thosein U_i;
until C' is empty;
1.3 define $S = \{S_1, S_2, \ldots, S_n\}$ with each S_i having a set of q_i declarations $\{p_{i1}, p_{i2}, \ldots, p_{iqi}\}$ where $i = 1$ to n.
Step 2: *//Preparing sufficient data to be embedded*
2.1 **while** $\lfloor \log_2(q_n! \times q_{n-1}! \times \ldots \times q_1! \times n!) \rfloor < t$ **do**
//The term $\lfloor \log_2(q_n! \times q_{n-1}! \times \ldots \times q_1! \times n!) \rfloor$ *is the min* Eq. (5) and t *is the number of bits in message* T
repeat
use key K and RNG G to choose randomly an *unselected* fake rule r from pool P;
until r is not in any CSS file of e-book B;
insert r into S;
set $n = n + 1$;
end while;
2.2 rearrange the n selectors S_1 through S_n of S into their lexicographical order;
2.3 **for** $i = 1$ to n
rearrange the q_i declarations of selector S_i into their lexicographical order;
end for;
2.4 randomize the positions of the m bits in message T by key K and RNG G and transform the result into a decimal integer I_{10}.
Step 3: *//Computing a multiple-based number*
3.1 divide I_{10} by $q_0!$ to obtain a quotient $Q_0 = \lfloor I_{10}/q_0! \rfloor$ and a remainder $D_0 = (I_{10}) \bmod (q_0!)$;
3.2 **for** $i = 1$ to n
divide Q_{i-1} by $q_i!$ to obtain a quotient $Q_i = \lfloor Q_{i-1}/(q_i!) \rfloor$ and a remainder $D_i = (Q_{i-1}) \bmod (q_i!)$;
end for;
3.3 compose all D_1 through D_n and D_0 in order to form an $(n+1)$-digit multiple-based number I_{mb} of the form of Eq. (2) as follows:

$$I_{\text{mb}} = D_n D_{n-1} \ldots D_1 D_0,$$

Algorithm 3 *(Continued.)* Data Embedding

such that the equality below according to Eq. (3) is true:

$$I_{10} = \sum_{i=1}^{n} [D_i \times (\prod_{j=0}^{i-1} q_j!)] + D_0$$

where D_i is a base-$(q_i!)$ number, $i = 0$ to n, and $q_0! = n!$.
Step 4: *//Embedding the multiple-based number*
4.1 **for** $i = 1$ to n
apply Algorithm 1 with $S_i = \{p_{i1}, p_{i2}, \ldots, p_{iqi}\}$ and D_i as the input to embed D_i, yielding an ordered rearrangement A_i of the q_i declarations in S_i;
//A_i *is the* (D_i+1)*st item in the* $(q_i!)$*-item list of the lexicographical rearrangements of all the* q_i*declarations*
end for;
4.2 applying Algorithm 1 with$S = \{S_1, S_2, \ldots, S_n\}$ and D_0 as the input to embed D_0, yielding an ordered rearrangement A_0 of the n selectors in S;
//A_0 *is just the* (D_0+1)*st item in the* $(q_0!)$*-item list of the lexicographical rearrangements of all the nselectors*
Step 5: *//Ending of the algorithm*
take as the output the final CSS file C with its selectors and declarations rearranged as described in Step 4.

In Step 2.1 of Algorithm 3, the left term in the condition $\lfloor \log_2(q_n! \times q_{n-1}! \times \ldots \times q_1! \times n!) \rfloor < t$ of the **while**loop comes from Eq. (5): $m = \lfloor \log_2(q_n! \times q_{n-1}! \times \ldots \times q_1! \times n!) \rfloor$. And the condition means that the data embedding capacity of the original number of rules are insufficient to embed the t bits of the input message T, and so the loop tries to select appropriate fake rules and add them into the sequence S of the rules in the CSS file to enlarge the data embedding capacity.

D. DATA EXTRACTION PROCESS

The data extraction process is basically a reverse version of the data hiding process described previously. At first the selectors and their declarations in the input stego-CSS file are collected. Then, the embedded multiple-based number is extracted from the arrangements of these selectors and declarations in the stego-CSS file. Finally, the multiple-based number is transformed into a binary string which is then taken to be the desired binary secret message. A flowchart illustrating these steps are shown in Fig. 4, and a corresponding detailed algorithm is described as Algorithm 4 subsequently.

V. EXPERIMENTAL RESULTS AND COMPARISONS WITH OTHER METHODS

In this section, firstly a report of some experimental results yielded by the proposed method is described, followed by a quantitative analysis and a security analysis of the results, as well as some comparisons of the results with those yielded by six other related methods.

Algorithm 4 Data Extraction

Input:
(1) a stego-CSS file C of an e-book B; and
(2) the secret key K and the RNG G used in generating C.

Output: binary message T with t bits embedded in C.

Steps.

Step 1: *//Extracting the embedded multiple-based number*

1.1 scan C sequentially to collect all the selectors $S_1, S_2, \ldots, S_n$ in C;

1.2 apply Algorithm 2 with S_1 through S_n as the input to extract a base-$(q_0!)$ number D_0 where $q_0! = n!$;

1.3 rearrange S_1 through S_n into their lexicographical order, resulting in a new sequence S'_1 through S'_n with each S'_i having q'_i declarations $p'_{i1}, p'_{i2}, \ldots, p'_{iqi}$;

1.4 **for** $i = 1$ to n
apply Algorithm 2 with declarations p'_{i1} through p'_{iqi} of S'_i as the input to extract a base$-(q'_i)$ number D_i;
end for;

1.5 compose D_1 through D_n and D_0 to form a multiple-based number as follows:

$$I_{mb} = D_n D_{n-1} \ldots D_1 D_0.$$

Step 2: *//Retrieving the embedded secret message*

2.1 transform I_{mb} into a decimal number I_{10} according to Eq. (3) as follows:

$$I_{10} = \sum_{i=1}^{n} [D_i \times (\prod_{j=0}^{i-1} q_j!)] + D_0;$$

2.2 transform I_{10} into a $t-$bit binary value I_b.

Step 3:
de-randomize the positions of the bits in I_b using key K and RGN G, and take the resulting $t-$bit string as the desired message T and exit.

A. EXPERIMENTAL RESULTS

In the experiments conducted in this study, a total of 40 CSS files were tested. The proposed algorithms were coded in the language C# under the Microsoft Visual Studio 2012 environment, and run in the Window 10 OS using a PC with an Intel i7 CPU and 8G RAM. The 40 CSS files were collected from a set of e-books with the EPUB format offered freely on the Internet [24]. Table 4 shows some statistics about the data of these 40 CSS files before and after Algorithm 3 was applied to them. It is mentioned firstly that the CSS files collected from the websites have already been well processed so that the numbers of the selectors in these files were not changed after the preprocessing steps in Algorithm 3 are applied to them. But the files sizes were shrunk indeed by the preprocessing, as seen from Table 4, to be about 70% of the original ones. In addition, the estimated data embedding capacity values shown in the table were computed by the proposed method with no fake rules added. It is mentioned by the way that the e-book "Harry Potter & the Philosopher's Stone" written by J. K. Rowling has various selectors with numerous declarations. Therefore, the e-book has a larger data embedding capacity. Fig. 5 is an example of the results of preprocessing, with the rules listed in Fig. 2 as the input. Fig. 6 is a scatter diagram showing the relations between the values of the data embedding capacity and the numbers of the rules in the CSS files listed in Table 4. From the figure, it can be seen that the data embedding capacity of a CSS file

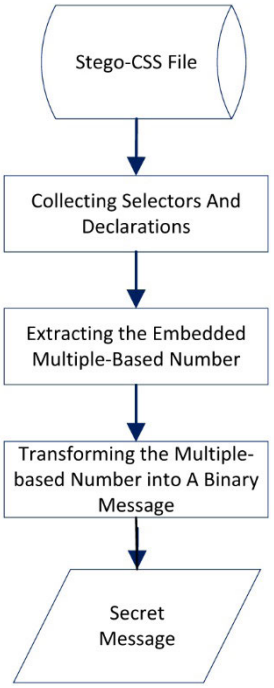


FIGURE 4. A flowchart of the proposed data extraction process.

TABLE 4. Some statistics about the CSS files of the 40 tested e-books with the EPUB format.

Name of e-book		Number of selectors	File size (bytes)	File size after pre-processing (bytes)	Estimated data embedding capacity (bits)
Gulliver's Travers		23	2,102	1,184	93
Alice's Adventures in Wonderland		21	959	713	75
Harry Potter & the Philosopher's Stone		48	2,132	2,093	274
The Little Mermaid		16	1,002	846	75
Red Riding Hood		15	1,392	1,015	92
⋮		⋮	⋮	⋮	⋮
Summary	Max	50	3,934	2,852	306
	Min	4	263	181	11
	Avg.	20.1	1,538.7	1,117.0	114.1

```
#guide {display:none;}
.formalpara-title {font-weight:bold;}
div.blockquote-title {font-weight:bold;margin-bottom:1em;margin-
  top:1em;}
nav ol li a {color:black;font-family:sans-serif;text-decoration:none;}
span.msgexplan-title {font-weight:bold;}
```

FIGURE 5. A demonstration of the rules shown in Fig. 2 after pre-processing.

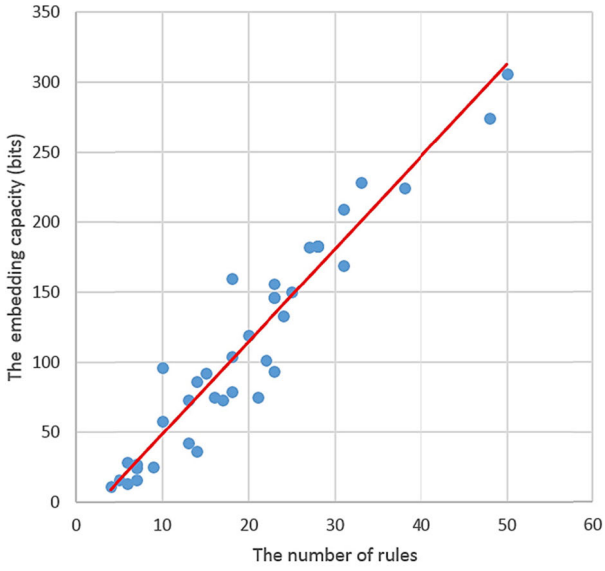


FIGURE 6. The relations between the data embedding capacity and the number of rules of the 40 CSS files in the e-books with the EPUB format tested in the experiments of this study.

is roughly linear with respective to the number of rules in the file.

B. QUANTITATIVE ANALYSIS OF DATA EMBEDDING CAPACITIES

Table 5 shows the results of respective size changes of five CSS files selected from the above-mentioned 40 ones after 24, 48, 72, 96, 200, 400, 600 and 800 random message bits are embedded into them. When the numbers of bits to be embedded into the CSS files exceed their estimated data embedding capacities (for example, 96, 200, 400, 600 and 800), respectively, fake rules were added by the proposed method to each of the files to enlarge the embedding capacity, as shown in Table 6. And Table 7 shows the resulting size changes of the e-books files.

By a more detailed inspection of the three tables, it can be seen from Table 5 that when the numbers of message bits to be embedded into the CSS files do not exceed their respective estimated embedding capacities, the sizes of the CSS files resulting from data embedding keep unchanged; on the contrary, when the non-exceeding condition is not met, the resulting CSS file sizes may be expanded highly up to several times of the original ones, partly due to the

```
div.c5 {padding-top:4em;}
h1.c4 {padding-bottom:1em;padding-top:2em;}
h3.c3 {font-style:italic;}
```

FIGURE 7. The first three rules in the CSS file of the e-book "Alice's Adventures in Wonderland".

```
.toc{margin-left:3em;margin-top:1em;display:block;font-
  size:1.2em;margin-bottom:1em;text-align:left;margin-right:3em;}
.note{margin-top:1em;display:block;font-style:italic;margin-
  bottom:1em;text-align:left;}
pre {font-family:inherit;}
```

FIGURE 8. The first three rules in the CSS file of the e-book "Alice's Adventures in Wonderland" after embedding.

inclusion of the fake rules shown in Table 6. However, even in such cases, Table 7 shows that the expansions of the sizes of the e-book files, which are what really visible to the user, instead are almost negligible ćw all the expansion rates are smaller than 1% even when the maximum of 800 message bits are embedded! Furthermore, the browsing results of all the stego-versions of the tested e-books appear to be totally identical to the original versions of the e-books as shown by our experimental results (an example will be shown later in Figures 9 and 10). Hopefully this will arouse no notice from a hacker.

It is noted here that an e-book file with the EPUB format may be produced by use of different compression software programs. To have fair comparisons, the e-book files tested in this study as shown in Table 7 are all generated by an identical compression program, namely, WinRAR [25], followed by the preprocessing operations described in Step 1 of Algorithm 3.

Figures 7 and 8 show parts of a CSS file of the e-book "Alice's Adventures in Wonderland" after pre-processing and after embedding, respectively. Although some fake rules were added to the CSS files and the contents of the CSS files were rearranged by the proposed data hiding method (described by Algorithm 3), the exterior appearances of the e-books do not change, as shown by the example illustrated by Figs. 9 and 10 which include parts of the exterior appearances of the e-book "Alice's Adventures in Wonderland" before and after embedding 800 message bits, respectively.

C. A COMPARISON WITH OTHER METHODS ABOUT DATA EMBEDDING CAPACITIES

To compare the proposed method with others in the aspect of data embedding capacity, six methods surveyed previously [12]–[18] and the one proposed in this study were used to embed as many message bits as possible into a CSS file of each of five e-books downloaded from the websites listed in [24]. The resulting numbers of message bits embedded in these e-books using the seven methods are listed

TABLE 5. The file sizes and change rates of the CSS files after embedding various numbers of message bits.

Name of e-book	Ori-ginal	After embedding 24 bits		After embedding 48 bits		After embedding 72 bits		After embedding 96 bits		After embedding 200 bits		After embedding 400 bits		After embedding 600 bits		After embedding 800 bits	
	File size (bytes)	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate
Gulliver's Travers	1,184	1,184	0%	1,184	0%	1,184	0%	1,260	6.42%	2,034	72%	3,371	185%	4,600	289%	5,910	399%
Alice's Adventures in Wonderland	713	713	0%	713	0%	713	0%	993	39.27%	1,737	144%	3,057	329%	4,252	496%	5,597	685%
Harry Potter & the Philosopher's Stone	2,093	2,093	0%	2,093	0%	2,093	0%	2,093	0%	2,093	0%	3,020	44%	4,232	102%	5,410	159%
The Little Mermaid	846	846	0%	846	0%	846	0%	1,126	33.10%	1,892	124%	3,333	294%	4,478	429%	5,874	594%
Red Riding Hood	1,015	1,015	0%	1,015	0%	1,015	0%	1,091	7.49%	1,942	91%	3,296	225%	4,554	349%	5,899	481%

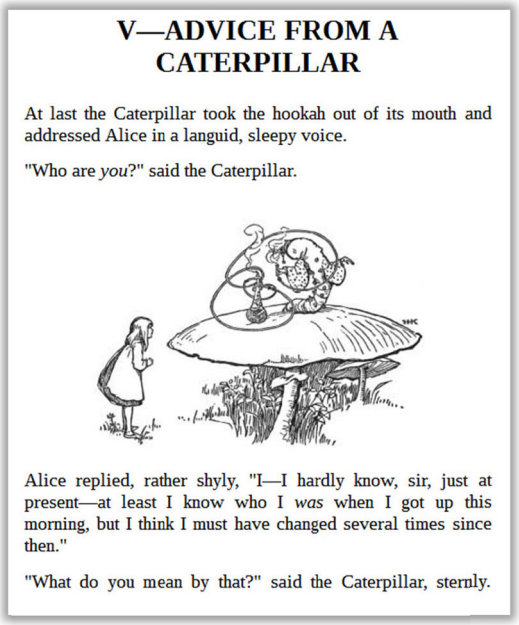
TABLE 6. The number of fake rules used when embedding different numbers of message bits as shown in Table 5.

Name of e-book	No. of bits embedded in the CSS file							
	24	48	72	96	200	400	600	800
Gulliver's Travers	0	0	0	1	12	42	70	97
Alice's Adventures in Wonderland	0	0	0	3	15	45	73	99
Harry Potter & the Philosopher's Stone	0	0	0	0	0	13	41	67
The Little Mermaid	0	0	0	3	15	47	75	102
Red Riding Hood	0	0	0	1	13	44	73	99

TABLE 7. The file sizes and change rates of the e-book files after preprocessing and after embedding various numbers of message bits.

Name of e-book	After pre-processing	After embedding 24 bits		After embedding 48 bits		After embedding 72 bits		After embedding 96 bits		After embedding 200 bits		After embedding 400 bits		After embedding 600 bits		After embedding 800 bits	
	File size (bytes)	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate	File size (bytes)	File size change rate
Gulliver's Travers	3,832,242	3,832,249	0.00%	3,832,253	0.00%	3,832,258	0.00%	3,832,299	0.00%	3,832,562	0.01%	3,832,951	0.02%	3,833,219	0.03%	3,833,435	0.03%
Alice's Adventures in Wonderland	541,405	541,409	0.00%	541,406	0.00%	541,416	0.00%	541,514	0.02%	541,763	0.07%	542,134	0.13%	542,386	0.18%	542,586	0.22%
Harry Potter & the Philosopher's Stone	778,800	778,805	0.00%	778,801	0.00%	778,807	0.00%	778,805	0.00%	778,832	0.00%	779,104	0.04%	779,465	0.09%	779,738	0.12%
The Little Mermaid	593,787	593,790	0.00%	593,788	0.00%	593,788	0.00%	593,896	0.02%	594,101	0.05%	594,484	0.12%	594,713	0.16%	594,932	0.19%
Red Riding Hood	121,373	121,376	0.00%	121,380	0.01%	121,380	0.01%	121,399	0.02%	121,626	0.21%	121,998	0.51%	122,249	0.72%	122,464	0.90%

in Table 8 and shown as a graph in Fig. 11. As can be seen from the table and the figure, the proposed method has a data embedding capacity larger than those of all the other methods except that of Dey *et al.* [12]. However, Day *et al.* [12] use the HTML characteristic of ''allowing the tag and attribute names to be written alternatively in capital or small letters'' to embed data, so that a resulting tag or attribute name might become a mixture of small and capital letters, for example,


V—ADVICE FROM A CATERPILLAR

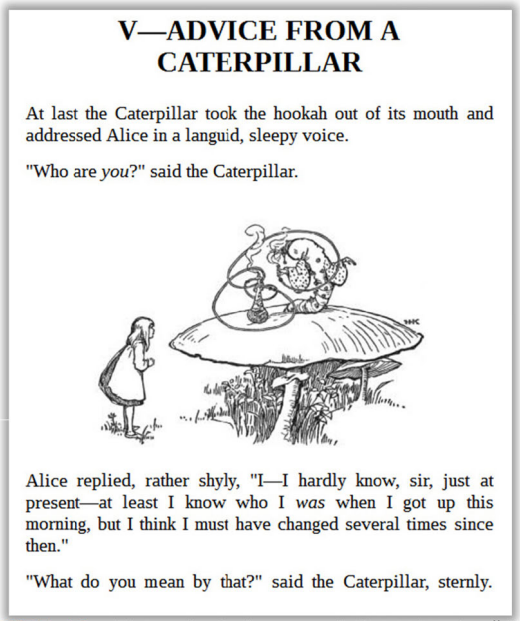
At last the Caterpillar took the hookah out of its mouth and addressed Alice in a languid, sleepy voice.

"Who are *you*?" said the Caterpillar.

Alice replied, rather shyly, "I—I hardly know, sir, just at present—at least I know who I *was* when I got up this morning, but I think I must have changed several times since then."

"What do you mean by that?" said the Caterpillar, sternly.

FIGURE 9. A selected page of the unprocessed e-book "Alice's Adventures in Wonderland" by Lewis Carroll (downloaded from Project Gutenberg [26]), as seen via an e-book viewer.


V—ADVICE FROM A CATERPILLAR

At last the Caterpillar took the hookah out of its mouth and addressed Alice in a languid, sleepy voice.

"Who are *you*?" said the Caterpillar.

Alice replied, rather shyly, "I—I hardly know, sir, just at present—at least I know who I *was* when I got up this morning, but I think I must have changed several times since then."

"What do you mean by that?" said the Caterpillar, sternly.

FIGURE 10. The selected page of the e-book "Alice's Adventures in Wonderland" by Lewis Carroll shown in fig. 9 with 800 secret bits embedded, as seen via an e-book viewer.

yielding 'tEXt-aLiGN' from embedding nine secret bits into the property name of 'text-align', which seems unnatural and is likely to arouse a hacker's notice.

On the other hand, the proposed method yields better data embedding capacities than the five methods other than [12]. The reason is that the proposed method rearranges both the selectors and declarations in the rules of the CSS file into their lexicographical orders for data embedding, while

TABLE 8. A comparison of the data embedding capacities of six existing methods and the proposed method for message embedding.

Name of e-book	No. of bits embedded in a CSS file						
	Dey et al. [12]	Jie [14]	Wu & Lai [15]	Huang et al. [16]	Kabetta et al. [17]	Lai et al. [18]	Proposed method
Gulliver's Travers	372	74	12	13	37	60	93
Alice's Adventures in Wonderland	273	65	6	6	29	24	75
Harry Potter & the Philosopher's Stone	919	202	38	43	100	166	274
The Little Mermaid	356	44	15	19	38	36	75
Red Riding Hood	460	40	19	21	48	58	92

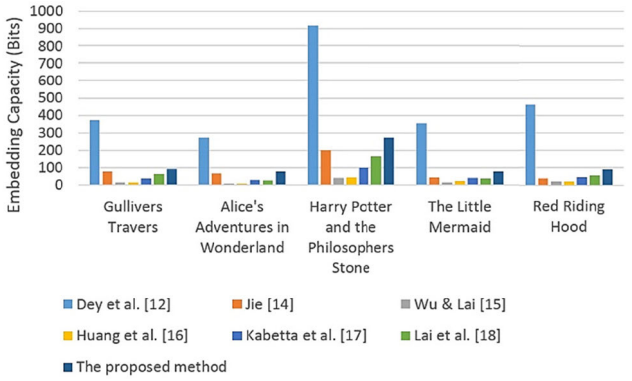


FIGURE 11. A graph representation of the data shown in Table 8 illustrating a comparison of the data embedding capacities of six existing methods and the proposed method for message embedding.

the other five methods, generally speaking, manipulate various entities (rules, attributes, etc.) in the documents (CSS, XML, or HTML) whose numbers are comparatively *smaller* for data embedding so that smaller embedding capacities are yielded. Specifically, as mentioned previously in the literature survey in Section I, Jie [14] permutes the so-called sub-elements in the XML to embed message bits, Wu and Lai [15] and Huang *et al.* [16] permute the attributes in the HTML for message embedding, Kabetta *et al.* [17] insert spaces or tabs after the semi-colon characters at the ends of the lines of the CSS files to embed bits, and finally Lai *et al.* [18] use different settings of the CSS attributes related to margins and paddings to embed bit data.

It is worth emphasis that in the experiments conducted for the above comparison, fake rules were *not* used by the proposed method to expand the data embedding capacity; otherwise, theoretically an *unlimited* data embedding capacity may be obtained by the proposed method to outperform all the compared methods, including the method by Dey *et al.* [12].

D. SECURITY ANALYSIS ABOUT ATTACKS AND A COMPARISON WITH OTHER METHODS

There are many text editing software packages which can be used to reformat the contents of the CSS files of an

```
.caption    {
    font-weight: bold;
        text-aLiGN: center  ;
    mARgiN-bottom  :  2em;}

 h1.c4 {padding-bottom:1em  ;  padding-top:2em;}
```

(a)

```
.caption {
  font-weight: bold;
  text-align: center;
  margin-bottom: 2em;
}
h1.c4 {
  padding-bottom: 1em;
  padding-top: 2em;
}
```

(b)

FIGURE 12. Effect of reformatting a sample part of a CSS file. (a) Before reformatting. (b) After reformatting.

TABLE 9. A comparison of the resistance capabilities of six methods and the proposed method to attacks by text reformatting.

Attacks	Dey et al. [12]	Jie [14]	Wu & Lai [15]	Huang et al. [16]	Kabetta et al. [17]	Lai et al. [18]	Proposed method
Eliminating excessive blank spaces and tabs between words or at line ends	Yes	Yes	Yes	Yes	**No**	Yes	Yes
Removing blank lines between text lines	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Normalizing property names into small letters	**No**	Yes	Yes	Yes	Yes	Yes	Yes
Indenting and reformatting text content	Yes	Yes	Yes	Yes	**No**	Yes	Yes

e-book, including the operations of eliminating the excessive blank spaces or tabs between the words or at the ends of text lines, removing blank lines between text lines, normalizing the property name by changing its characters to be small letters, and indenting the text lines. These operations may be regarded as attacks to the CSS file contents. A real example yielded by our experiment is shown in Fig. 12. Their effects to the proposed method as well as the previously-mentioned six methods have been checked and listed in Table 9. As can be seen from the table, the proposed method can resist all of the four types of attacks, while some of the others ([12] and [17]) cannot.

VI. CONCLUSION AND DISCUSSION

A data hiding method via e-books of the EPUB format is proposed. The method embeds a given secret message imperceptibly into the CSS file of an e-book by rearranging the selectors and declarations in the rules in the CSS file to represent the digits in the multiple-based number resulting from converting the message bits. The resulting exterior appearance of the e-book is not changed. The method also flexibly inserts fake rules into the CSS file to enlarge the data embedding capacity whenever necessary. This mechanism can embed a message with no limitation on the data volume. Secret keys are also used to randomize the positions of the message bits before they are embedded to enhance the hidden data security. Fake rules are selected from a pre-constructed fake rule pool which contains ordinary rules collected from common e-books. This increases the steganographic effect of the modified CSS file. Good experimental results show the correctness and feasibility of the proposed method, and comparisons with six other related methods in the aspects of data embedding capacity and reformatting attack show the superiority of the proposed method.

ACKNOWLEDGMENT

The basic idea of the proposed method in this paper has been published previously in Su and Wu [27] as a 4-page short paper, which includes limited details and few experimental results using the CSS files of only two e-books as the tested data. In this study, a great deal of improvement has been made, including (1) enlarging the survey of related studies and adding more references; (2) elaborating the principle behind the proposed method like embedding the message as a multiple-based number; (3) including additionally a data extraction algorithm (Algorithm 4) and describing its detailed steps and those of the data embedding algorithm; (4) proposing two new algorithms (Algorithms 1 and 2) to construct lexicographically-rearranged items directly to represent multiple-based numbers for message embedding and to extract message data in a reverse manner, respectively; (5) adding many examples to illustrate the ideas of the proposed method; (6) increasing the tested data used in the experiments; (7) providing more experimental data and the analysis of all the experimental results by graphs or diagrams; (8) adding data-security enhancement measures using a secret key and a random number generator; and (9) comparing the proposed method with six other ones in aspects of data embedding capacity and text reformatting attack to show the superiority of the proposed method.

REFERENCES

[1] D. Kahn, "The history of steganography," in *Proc. 1st Int. Workshop Inf. Hiding*, in Lecture Notes in Computer Science, vol. 1174, Cambridge, U.K., R. Anderson, Ed. Berlin, Germany: Springer, May/Jun. 1996, pp. 1–5.

[2] D. Artz, "Digital steganography: Hiding data within data," *IEEE Internet Comput.*, vol. 5, no. 3, pp. 75–80, 2001.

[3] Y.-Q. Shi, X. Li, X. Zhang, H.-T. Wu, and B. Ma, "Reversible data hiding: Advances in the past two decades," *IEEE Access*, vol. 4, pp. 3210–3237, 2016.

[4] I. Cox, M. Miller, and A. Mckellips, "Watermarking as communications with side information," *Proc. IEEE*, vol. 87, no. 7, pp. 1127–1141, Jul. 1999.

[5] K. Muhammad, M. Sajjad, I. Mehmood, S. Rho, and S. W. Baik, "Image steganography using uncorrelated color space and its application for security of visual contents in online social networks," *Future Gener. Comput. Syst.*, vol. 86, pp. 951–960, Sep. 2018.

[6] M. Taleby Ahvanooey, Q. Li, J. Hou, H. Dana Mazraeh, and J. Zhang, "AITSteg: An innovative text steganography technique for hidden transmission of text message via social media," *IEEE Access*, vol. 6, pp. 65981–65995, 2018.

[7] F. Bertini, S. G. Rizzo, and D. Montesi, "Can information hiding in social media posts represent a threat?" *Computer*, vol. 52, no. 10, pp. 52–60, Oct. 2019.

[8] D.-C. Wu and W.-H. Tsai, "Data hiding in images via multiple-based number conversion and lossy compression," *IEEE Trans. Consum. Electron.*, vol. 44, no. 4, pp. 1406–1412, Nov. 1998.
[9] D.-C. Wu and W.-H. Tsai, "Embedding of any type of data in images based on a human visual model and multiple-based number conversion," *Pattern Recognit. Lett.*, vol. 20, no. 14, pp. 1511–1517, Dec. 1999.
[10] H.-M. Chao, C.-M. Hsu, and S.-G. Miaou, "A data-hiding technique with authentication, integration, and confidentiality for electronic patient records," *IEEE Trans. Inform. Technol. Biomed.*, vol. 6, no. 1, pp. 46–53, Mar. 2002.
[11] I. S. Lee and W. H. Tsai, "Secret communication through Web pages using special space codes in HTML files," *Int. J. Appl. Sci. Eng.*, vol. 6, no. 2, pp. 141–149, 2008.
[12] S. Dey, H. Al-Qaheri, and S. Sanyal, "Embedding secret data in html Web page," in *Proc. Image Process. Commun. Challenges Conf.*, Poland, Apr. 2010, pp. 474–481.
[13] S. Inoue, K. Makino, I. Murase, O. Takizawa, T. Matsumoto, and H. Nakagawa, "A proposal on information hiding methods using XML," in *Proc. 1st NLP XML Workshop*, Japan, 2001, pp. 55–62.
[14] Y. Jie, "Algorithm of XML document information hiding based on equal element," in *Proc. 3rd Int. Conf. Comput. Sci. Inf. Technol.*, Jul. 2010, pp. 250–253.
[15] D. C. Wu and P. H. Lai, "Novel techniques of data hiding in HTML documents," in *Proc. Conf. Digit. Contents Manage. Appl.*, Kaohsiung, Taiwan, 2005, pp. 21–30.
[16] H. Huang, S. Zhong, and X. Sun, "An algorithm of Webpage information hiding based on attributes permutation," in *Proc. Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, Harbin, China, Aug. 2008, pp. 257–260.
[17] H. Kabetta, B. Y. Dwiandiyanta and Suyoto, "Information hiding in CSS: A secure scheme text-steganography using public key cryptosystem," *Int. J. Cryptogr. Inform. Secur.*, vol. 1, no. 1, Dec. 2011.
[18] J. X. Lai, Y. C. Chou, C. C. Tseng, and H. C. Liao, "A Large Payload Webpage Data Embedding Method Using CSS Attributes Modification," in *Proc. Adv. Intell. Inf. Hiding Multimedia Signal Process., 12th Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, vol. 1. Springer: Kaohsiung, Taiwan, Nov. 2016, pp. 91–98.
[19] Wikipedia. *The Free Encyclopedia 'EPUB*. Accessed: Sep. 10, 2019. [Online]. Available: https://en.wikipedia.org/wiki/EPUB
[20] *W3C, EPUB 3.2*. Accessed: Oct. 15, 2019. [Online]. Available: https://www.w3.org/publishing/epub3/epub-spec.html
[21] *W3C, 'Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification'*. Accessed: Jul. 26, 2012. [Online]. Available: http://www.w3.org/TR/CSS22/
[22] w3schools.com. *CSS Tutorial—CSS Syntax*. Accessed: Oct. 11, 2018. [Online]. Available: https://www.w3schools.com/css/css_syntax.asp
[23] Wikipedia. *Arbitrary-Precision Arithmetic*. Accessed: Aug. 8, 2013. [Online]. Available: http://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic
[24] Eupor. *25 Sites To Download Free EPUB Ebook*. Accessed: Oct. 3, 2019. [Online]. Available: https://www.epubor.com/25-sites-to-download-free-epub-ebooks.html
[25] A. Roshal. *WinRAR Archiver, a Powerful Tool to Process RAR and ZIP Files*. Accessed: Sep. 22, 2019. [Online]. Available: https://rarlab.com/
[26] Project Gutenberg. *Free eBooks—Project Gutenberg*. Accessed: Oct. 13, 2019. [Online]. Available: https://www.gutenberg.org/
[27] D.-C. Wu and H.-Y. Su, "Information hiding in EPUB files by rearranging the contents of CSS files," in *Proc. 9th Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, Beijing, China, Oct. 2013, pp. 80–83.

DA-CHUN WU was born in Taiwan, in 1959. He received the B.S. degree in computer science and the M.S. degree in information engineering from Tamkang University, Taipei, Taiwan, in 1983 and 1985, respectively, and the Ph.D. degree in computer and information science from National Chiao Tung University, Hsinchu, in 1999.

He joined the faculty of the Department of Information Management, Ming Chuan University, Taipei, in 1987. From 2002 to 2018, he was with the National Kaohsiung First University of Science and Technology (NKFUST), Kaohsiung, Taiwan, where he was the Director of the Library and Information Center, from 2010 to 2014, and the Head of the Department of Computer and Communication Engineering, from 2015 to 2018. He is currently an Associate Professor with the Department of Computer and Communication Engineering, National Kaohsiung University of Science and Technology (NKUST), Kaohsiung. His recent interests include multimedia security, image processing, machine learning, and artificial intelligence.

HSIU-YANG SU was born in Taiwan, in 1987. He received the B.S. degree from the Department of Computer Science and Engineering, Tatung University, Taipei, Taiwan, in 2010, and the M.S. degree from the Department of Computer and Communication Engineering, National Kaohsiung First University of Science and Technology, Kaohsiung, Taiwan, in 2012.

His current research interests include electronic trading and FinTech.