A New Data Hiding Method via Revision History Records on Collaborative Writing Platforms

YA-LIN LEE[1] AND WEN-HSIANG TSAI[1, 2], [1]National Chiao Tung University, [2]Asia University, Taiwan

A new data hiding method via collaboratively-written articles with forged revision history records on collaborative writing platforms is proposed. The hidden message is camouflaged as a stego-document consisting of a stego-article and a revision history created through a simulated process of collaborative writing. The revisions are forged using a database constructed by mining word sequences used in real cases from an English Wikipedia XML dump. Four characteristics of article revisions are identified and utilized to embed secret messages, including the author of each revision, the number of corrected word sequences, the content of the corrected word sequences, and the word sequences replacing the corrected ones. Related problems arising in utilizing these characteristics for data hiding are identified and solved skillfully, resulting in an effective multi-way method for hiding secret messages into the revision history. To create more realistic revisions, Huffman coding based on the word sequence frequencies collected from Wikipedia is applied to encode the word sequences. Good experimental results show the feasibility of the proposed method.

Categories and Subject Descriptors: H.4.3 [**Information Systems Applications**]: Communications Applications; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Linguistic processing, dictionaries*; H.2.8 [**Database Management**]: Database Applications—*Data mining, statistical databases*; I.7 [**Document and Text Processing**]

General Terms: Security, Algorithms

Additional Key Words and Phrases: Data hiding, Wikipedia mining, collaborative writing, revision history, Huffman coding

ACM Reference Format:

Lee, Y. L. and Tsai, W. H. 2013. A new data hiding method via revision history records on collaborative writing platforms.

1. INTRODUCTION

Data hiding is the art of hiding secret messages into cover media for the applications of covert communication, secret data keeping, access control, database protection, and so on. Types of cover media include image, video, audio, text, etc. Attacking the weaknesses of human auditory and visual systems, many researches on data hiding focused on non-text cover media, such as [Cheddad et al. 2010; Doerr and Dugelay 2003; Lie and Chang 2006; Lin et al. 2011; Mohanty and Bhargava 2008; Tai et al. 2009]. Less data hiding techniques using text-type cover media have been proposed. Bennett [2004] made a good survey about hiding data in text and classified related techniques into three categories: format-based methods, random and statistical generation, and linguistic methods.

Format-based methods use the physical formats of documents to hide messages. Some of them utilize spaces in documents to encode message data. For example, Alattar and Alattar [2004] proposed a method that adjusts the distances between words or text lines using spread-spectrum and BCH error-correction techniques, and Kim et al. [2003] proposed a word-shift algorithm that adjusts the spaces between words based on concepts of word classification and statistics of inter-word spaces. Some other methods utilize non-displayed characters to hide messages, such as Lee and Tsai [2010] who encode message bits using special ASCII codes and hide the result between the words or characters in PDF files.

Random and statistical methods generate directly camouflage texts with hidden messages to prevent the attack of comparison with a known plaintext. For example, Wayner [Wayner 1992;

This research was supported in part by the NSC, Taiwan under Grant No. 101-3113-P-009-006 and in part by the Ministry of Education, Taiwan under the 5-year Project II of "Aiming for the Top University" from 2011 through 2015.
Authors' addresses: Y. L. Lee, Institute of Computer Science and Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan; email: yllee.cs98g@g2.nctu.edu.tw; W. H. Tsai, Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan; email: whtsai@cis.nctu.edu.tw.

Wayner 2002] proposed a method for text generation based on the use of context-free grammars and tree structures. A method available on a website [Spammimic.com 2010] extends the idea to generate fake spam emails with hidden messages, which are usually ignored by people.

Linguistic methods use written natural languages to conceal secret messages. For example, Chapman et al. [2001] proposed a synonym replacement method that generates a cover text according to a secret message using sentence models and a synonym dictionary. Bolshakov [2004] extended the synonym replacement method by using a specific synonymy dictionary and a very large database of collocations to create a cover text, which is more believable to a human reader. Shirali-Shahreza and Shirali-Shahreza [2008] proposed a third synonym replacement method that hides data in a text by substituting the words which have different terms in the UK and the US. Stutsman et al. [2006] proposed a method to hide messages in the noise that is inherent in natural language translation results without the necessity of transmitting the source text for decoding.

Recently, more and more collaborative writing platforms are available, such as Google Drive, Office Web Apps, Wikipedia, etc. On these platforms, a huge number of revisions generated during the collaborative writing process are recorded. Furthermore, many people work collaboratively on these platforms. Thus, these platforms are very suitable for data hiding applications, such as covert communication, secret data keeping, etc. It is desired to propose a new method which is useful for covert communication or secure keeping of secret messages on collaborative writing platforms However, the above-mentioned methods can only be applied to documents with single authors and single revision versions, meaning that they are not suitable for hiding data on collaborative writing platforms. Therefore, the goal of this paper is to propose a new data hiding method which can hide data into documents created on collaborative writing platforms. In more detail, a new data hiding method is proposed, which simulates a collaborative writing process to generate a fake document, consisting of an article and its revision history, as a camouflage for message bit embedding. As shown in Figure 1, with the input of an article and a secret message, the proposed method utilizes multiple virtual authors to collaboratively revise the article, generating artificially a history of earlier revisions of the article according to the secret message. An ordinary reader will consider the resulting stego-document as a normal collaborative writing output, and cannot realize the existence of the secret message hidden in the document.

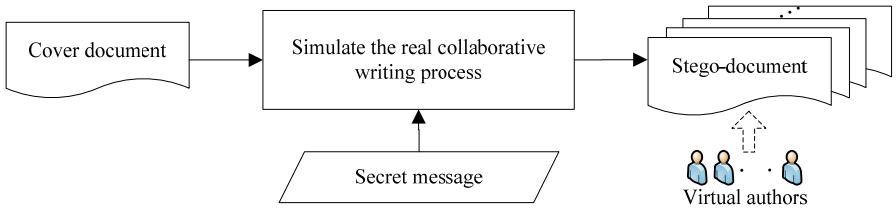

Fig. 1. Basic idea of proposed method that generates a revision history of a stego-document as a camouflage for data hiding.

Moreover, the previously-mentioned linguistic methods use written natural languages to generate stego-documents and can produce more innocuous stego-texts than other data hiding methods, but an issue common to them is how to find a nature way for simulating the writing process and how to obtain large-volume written data *automatically*. Hence, another goal of this paper is to find a nature way to generate the revision history and to obtain large-volume collaborative writing data *automatically*. In recent years, some researches have been conducted to analyze the revision history data of Wikipedia articles for various natural language processing applications [Bronner and Monz 2012; Bronner et al. 2012; Dutrey et al. 2010; Erdmann et al. 2009; Max and Wisniewski 2010; Nelken and Yamangil 2008; Viégas et al. 2004], such as spelling corrections, reformulations, text summarization, user edits classification, multilingual content synchronization, etc. In addition to being useful for these applications, the collaboratively written data in Wikipedia are also very suitable,

as found in this study, for simulating the collaborative writing process for the purpose of data hiding since it is the largest collaborative writing platform nowadays.

In [Liu and Tsai 2007], Liu and Tsai proposed a data hiding method via Microsoft Word documents by the use of the change tracking function, which embeds a secret message by mimicking a pre-draft document written by an author with an inferior writing skill and encoding the secret message by choices of degenerations in the writing. Although they used three databases for degenerations, the sizes of them are quite small when compared to that of the database constructed from Wikipedia which we make use for data embedding in this study. It is noted by the way that a data hiding method can, as well known, embed more bits by making use of a larger database. Furthermore, in [Liu and Tsai 2007] a stego-document is generated by only two virtual persons and the change tracking data are made by the one with a better writing skill. This scenario is *insufficient* for simulating a normal collaborative writing process. Therefore, in this paper we propose a new *framework* that uses the revision-history data from Wikipedia and simulates real collaborative writing processes to hide secret messages. Four characteristics of collaborative writing processes are analyzed and utilized for message hiding, including the author of each revision, the number of corrected word sequences, the content of the corrected word sequences, and the word sequences replacing the corrected ones. The proposed method is useful for covert communication or secure keeping of secret messages on collaborative writing platforms.

In the remainder of this paper, the idea of the proposed method is described in Section 2. Detailed algorithms for collaborative writing database construction, secret message embedding, and secret message extraction are given in Section 3. In Section 4, some experimental results are presented to show the feasibility of the proposed method, and in Section 5, we discuss the security issue, followed by conclusions in Section 6.

2. BASIC IDEA OF PROPOSED METHOD

Collaborative writing means an activity involving more than one author to create an article cooperatively on a common platform. The purposes of establishing a *collaborative writing platform* includes knowledge sharing, project management, data keeping, etc. Many collaborative writing platforms are available, such as Google Drive, Office Web Apps, Wikipedia, etc., which record revisions generated during the collaborative writing process. In general, the recorded information of a revision includes: 1) the author of the revision, 2) the time the revision was made, and 3) the content of the revision.

To achieve the goal of creating camouflage revisions in collaborative writing for message hiding in this study, we analyze the existing revision-history data of articles on Wikipedia, which is the largest collaborative writing platform on the Internet currently in the world. The aim is to get real and large *collaborative writing data* contributed by people all over the world and use them to create more realistic revision histories to enhance the resulting effect of data embedding. However, since the collaborative writing process is very complicated, it is hard to find a unified model to simulate it. Many different types of modifications may be made during the collaborative writing process [Bronner and Monz 2012; Dutrey et al. 2010], such as error corrections, paraphrasing, factual edits, etc. Moreover, different languages usually require different models to represent due to their distinctive grammatical structures. Therefore, in order to get useful collaborative writing data *automatically* from the revision history data on Wikipedia without building models manually and to generalize a method that can be applied to multiple languages, we assume that only *word sequence corrections* occur during a revision. Some characteristics in collaborative writing based on this assumption for data embedding are identified, which will be discussed in the following. It is noted that various text articles, not only in English but also in other languages, can be utilized as cover media in this study.

The revision history of each article in Wikipedia is stored in a database, and one can recover any previous revision version of the article by an interface provided on the site. For this study, we have collected a large set of revision-history data from Wikipedia, and in the proposed method we mine this set to get useful information about word usages in the revisions. Then, we use the acquired information to simulate a collaborative writing process, starting from a *cover article*; and generate a *stego-article* with a sequence of revisions according to the secret message and a secret key. The resulting *stego-document*, including the stego-article and the revision history, looks like a work created by a group of real authors, achieving an effect of camouflage. In contrast, we call the original article with an initially-empty history a *cover document* in the sequel.

More specifically, the proposed method includes three main phases as shown in Figure 2: 1) construction of a collaborative-writing database; 2) secret message embedding; and 3) secret message extraction. In the first phase, a large number of articles acquired from Wikipedia are analyzed and useful collaboratively written data about word usages are mined using a natural language processing technique. The mined data then are used to construct a database, called the *collaborative writing database*, denoted as DB_{cw} subsequently. In the second phase, with the input of a cover document, a secret message, and a secret key, a stego-document with a *fake* revision history is generated by simulating a real collaborative writing process using DB_{cw}. The revisions in the history are supposed to be made by multiple virtual authors; and the following *characteristics* of each revision are decided by the secret message: 1) the author of the revision; 2) the number of *changed word sequences* of the revision; 3) the changed word sequences in the revision; and 4) the word sequences selected from the collaborative writing database DB_{cw}, which replace those of 3), called the *replacing word sequences* in the sequel. And in the third phase, an authorized person who has the secret key can extract the secret message from the stego-document, while those who do not have the key cannot do so. They even could not realize the existence of the secret message because the secret message is disguised as the revision history in the stego-document. Note that the second and third phases can be applied on *any* collaborative writing platforms, not just on Wikipedia; Wikipedia is merely utilized in the first phase to construct the collaborative writing database DB_{cw} in this study.

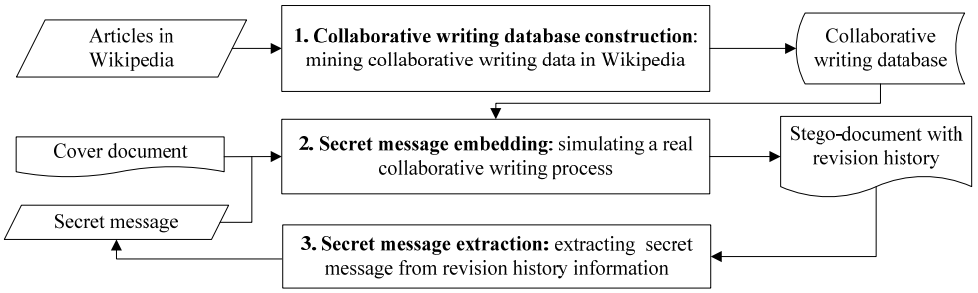

Fig. 2. Flow diagram of the proposed method.

3. DATA HIDING VIA REVISION HISTORY

In this section, the details of the proposed method for using the analyzed characteristics of collaborative writing to hide secret messages are described in the following, where the first part is collaborative writing database construction, the second part is secret message embedding, and the final part is secret message extraction.

3.1 Collaborative Writing Database Construction

To construct the aforementioned collaborative writing database DB_{cw}, we try to mine the revision data collected from Wikipedia. There were about 4.2 million articles in the English Wikipedia in May 2013, which is a very large knowledge repository; therefore, it is suitable to use it as a source for

constructing the database DB_{cw} desired in the study. Specifically, at first we downloaded part of the English Wikipedia XML dump with the complete revision histories of all the articles on August 3, 2011. Then, we mine the useful collaborative writing data from the downloaded data set under the assumption that only *word sequence corrections* will occur during a revision.

As illustrated in Figure 3, each downloaded article P has a set of revisions $\{D_0, D_1, \ldots, D_n\}$ in its revision history, where a newer revision D_i has a smaller index i with D_0 being the latest version of the article. For every two consecutive revisions D_i and D_{i-1}, we find all the *correction pairs* between D_i and D_{i-1}, each denoted as $<s_j, s_j'>$, where s_j is a word sequence in revision D_i and was corrected to become another, namely, s_j', by the author of revision D_{i-1}. Then, we collect all correction pairs so found to construct the database DB_{cw}. For example, assume D_i = "National Chia Tang University" and D_{i-1} = "National Chiao Tung University." Then, the correction pair $<s_1, s_1'>$ = <"Chia Tang", "Chiao Tung"> is generated and included into DB_{cw}.

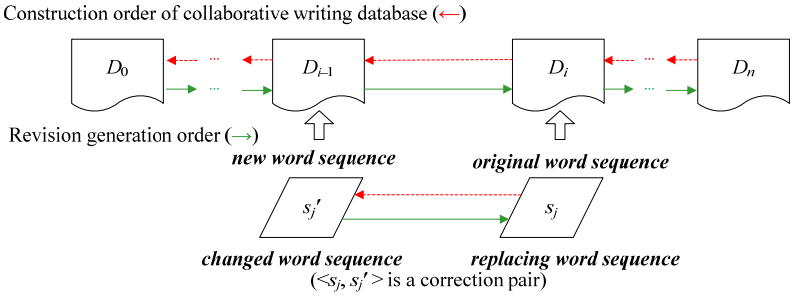

Fig. 3. Illustration of used terms and notations.

Moreover, about the properties of correction pairs, it was observed that if the *context* of a word sequence s_j in revision D_i is the same as that of a word sequence s_j' in revision D_{i-1} (that is, if the preceding word of s_j is the same as that of s_j' and the succeeding word of s_j is the same as that of s_j' as well), then $<s_j, s_j'>$ is a correction pair. Accordingly, a novel algorithm is proposed in this study for finding *automatically* all of the correction pairs between every two consecutive revisions for inclusion in DB_{cw}. The algorithm is an extension of the longest common subsequence (LCS) algorithm [Bergroth et al. 2000]. The details are described in Algorithm 1.

Algorithm 1. Finding correction pairs

Input: two consecutive revisions D_i and D_{i-1} in the revision history of an article P.
Output: the correction pairs between D_i and D_{i-1}.
Stage 1 — finding the longest common subsequence.

1) (*Splitting revisions into word sets*) Split D_i and D_{i-1} into two sets of words, $W = \{w_1, w_2, \ldots, w_n\}$ and $W' = \{w_1', w_2', \ldots, w_m'\}$, respectively.
2) (*Constructing* a *counting table by dynamic programming*) Construct an $n \times m$ counting table T to record the lengths of the common subsequences of W and W' as follows.
   a) Initialize all elements in table T to be zero.
   b) Compute the values of table T from the upper left and denote the currently-processed entry in T by $T(x, y)$ with $x = 1$ and $y = 1$ initially.
   c) If the content of w_x is identical to that of w_y', then let $T(x, y) = T(x - 1, y - 1) + 1$; else, let $T(x, y) = \max(T(x - 1, y), T(x, y - 1))$.
   d) If x is not larger than n, then let $x = x + 1$ and go to Step 2c); else, if y is not larger than m, then let $x = 1$ and $y = y + 1$ and go to Step 2c); else, regard table T as being filled up and continue.

3) (*Finding the longest common subsequence*) Apply a backtracking procedure to table T, starting from $T(m, n)$, to find the longest common subsequence $L = \{l_1, l_2, \ldots, l_t\}$, where each element l_i in L is a word common to W and W'.

Stage 2 —finding the correction pairs.

4) (*Finding the correction pairs*) Starting from the first element l_1 of L with the currently-processed element in L being denoted by l_p, find the correction pairs as follows.
   a) If the word sequence s_j in D_i with its preceding and succeeding words being l_p and l_{p+1}, respectively, is not empty and if the word sequence s_j' in D_{i-1} with the same context condition is not empty, either, then take $<s_j, s_j'>$ as a correction pair.
   b) Increment p by 1 and go to Step 4) until $p > t$.

We run Algorithm 1 for every two consecutive revisions of all the articles downloaded from Wikipedia to obtain a large set of correction pairs and write them into the database DB_{cw}. Furthermore, we count the total number N_{cp} of times that each correction pair CP is so obtained, and call the number N_{cp} the *correction count* of CP. The correction counts are also kept in the database DB_{cw} for use in the proposed data hiding process.

As a summary, we use a *record* in the database DB_{cw} to keep the following information about a correction pair $<s_j, s_j'>$: 1) an *original* word sequence s_j; 2) a *new* word sequence s_j'; and 3) the correction count N_{cp} of the pair. Moreover, we define a *chosen set* of a word sequence s' in DB_{cw} to be the one which include all the correction pairs $<s, s'>$ with s' as their identical new word sequences. For example, Table III (shown in Section IV) shows a chosen set of the word sequence "such as."

3.2 Secret Message Embedding

In the phase of message embedding with a cover document D_0 as the input, the proposed system is designed to generate a stego-document D' with consecutive revisions $\{D_0, D_1, D_2, \ldots, D_n\}$ by producing a *previous* revision D_i from the *current* revision D_{i-1} repeatedly until the entire message is embedded, as shown in Figure 3 where the direction of revision generation is indicated by the green arrows. The stego-document D' including the revision history $\{D_0, D_1, D_2, \ldots, D_n\}$ then is kept on a collaborative writing platform, which may be Wikipedia or others. To simulate a collaborative writing process more realistically, we utilize the four aforementioned characteristics of revisions to "hide" the message bits into the revisions sequentially: 1) the author of the previous revision D_i, 2) the number of changed word sequences in the current revision D_{i-1}, 3) the changed word sequences in the current revision D_{i-1}, and 4) the replacing word sequences in the previous revision D_i, as described in the following.

3.2.1 *Encoding the Authors of Revisions for Data Hiding.* We encode the authors of revisions to hide message bits in the proposed method. For this, at first we select a group of simulated authors, with each author being assigned a unique code a, called *author* a. Then, if the message bits to be embedded form a code a_j, then we assign author a_j to the previous revision D_i as its author to achieve embedding of message bits a_j into D_i. For example, assume that four authors are selected and each is assigned a unique code a as shown in Figure 4, respectively. If the message bits a_j to be embedded is "01," then Jessy with author code "01" is selected to be the author of the revision D_i. Moreover, every revision of D_0 through D_n will be assigned an author according to the corresponding message bits, and so an author can be assigned to conduct more than one revision or reversely no revision in the generated revisions.

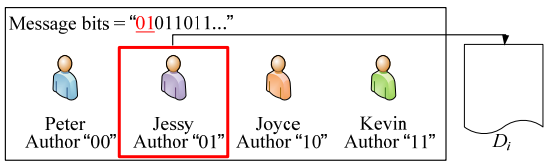

Fig. 4. Illustration of encoding authors of revisions for data hiding.

3.2.2 *Using the Number of Changed Word Sequences for Data Hiding.* In the process of generating the previous revision D_i from the current one D_{i-1}, we select some word sequences in D_{i-1} and changed them into other ones in D_i. It is desired to use as well the *number* N_g of word sequences changed in this process as a *message-bit carrier*.

To implement this aim, at first we set on the magnitude of N_g a limit N_c taken to be the *maximum* allowed number of word sequences in D_{i-1} that can be changed to yield D_i. This limitation makes the simulated step of revising D_{i-1} to become D_i look more realistic because usually not very many words are corrected in a single revision. Next, we scan the word sequences in the text of the current revision D_{i-1} sequentially and search DB_{cw} to find all the correction pairs $<s_j, s_j'>$ with s_j' in D_{i-1}. Then, we collect all s_j' in these pairs as a set Q_r, which we call the *candidate set of word sequences for changes in* D_{i-1}. Finally, we select N_g word sequences in Q_r to form a set Q_c such that the binary version of the number N_g is just the current message bits to be embedded.

But for this process of using N_g as a message-bit carrier to be feasible, several problems must be solved beforehand, including: 1) the *dependency problem*, 2) the *selection problem*, 3) the *consecutiveness problem*, and 4) the *encoding problem*, as described in the following.

3.2.2.1. *The dependency problem.* We say that two word sequences in D_{i-1} are *dependent* if some identical words appear in both of them, and changing word sequences with this property in D_{i-1} will cause conflicts, leading to a *dependency problem* which we explain by an example as follows.

As shown in Figure 5(a), D_{i-1} = "you are not wrong, who deem that my days have been a dream" and Q_r includes 11 word sequences denoted as q_1 through q_{11}, respectively. From Figure 5(a) we can see that the word sequences q_2, q_3, and q_5 in Q_r are dependent on the word sequence q_4 because the intersection of each of the former three with the latter one is non-empty. If we correct q_4 = "are not wrong" in D_{i-1} to be another, say "is right," then the dependent word sequences q_2, q_3, and q_5 in D_{i-1} *cannot* be selected and changed anymore because they include word sequences in q_4 which have already been changed and disappeared. That is, any part of a changed word sequence cannot be changed again; otherwise, a dependency problem will occur.

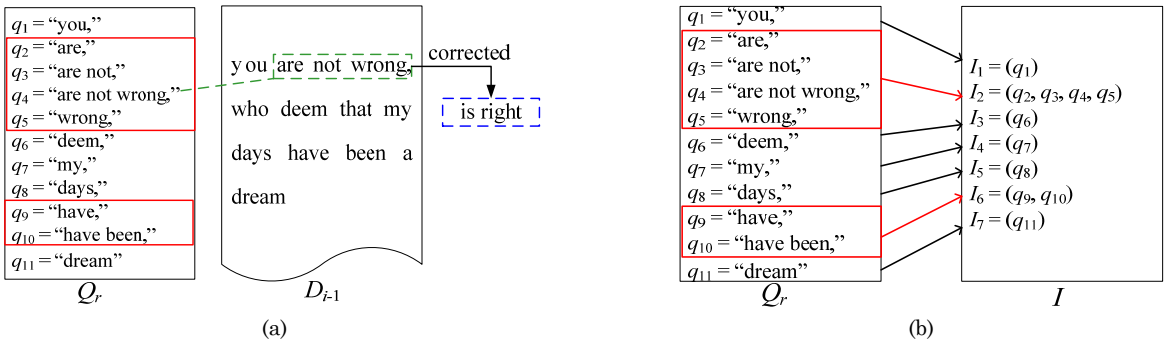

Fig. 5. Illustration of the dependency problem. (a) Revision D_{i-1} and candidate set Q_r where the dependent word sequences are surrounded by red squares. (b) Set I that corresponds to the set Q_r for solving the dependency problem.

To avoid this problem in creating D_i from D_{i-1}, we propose a two-step scheme: 1) decompose Q_r into a set of lists, $I = \{I_1, I_2, \ldots, I_u\}$, with each list I_i including a group of mutually dependent word sequences (i.e., with every word sequence in each I_i being *dependent* on another in the same list) and every two word sequences in two different lists, respectively, in I being *independent* of each other; and 2) select only word sequences from different lists in set I and change them to construct a new revision. The details to implement the first step is described in Algorithm 2. After applying the first step on the set Q_r as shown in Figure 5(b), it will be transformed into $I = \{(q_1), (q_2, q_3, q_4, q_5), (q_6), (q_7), (q_8), (q_9, q_{10}), (q_{11})\}$ where each pair of parentheses encloses a list of mutually dependent word sequences. With I

ready, we can now select word sequences from distinct lists I_i in it, such as q_1, q_2, q_6, and q_9, to simulate changes of word sequences in revision D_{i-1} without causing the dependency problem.

3.2.2.2. *The selection problem.* It is desired to select word sequences for use in the simulated revisions according to their *usage frequencies* in DB_{cw}, so that a more frequently-corrected word sequence has a larger probability to be selected, forging a more realistic revision. For this aim, following [Liu and Tsai 2007; Wayner 1992], we adopt the Huffman coding technique to create Huffman codes uniquely for the word sequences in Q_r according to their usage frequencies, and select word sequences with their codes identical to the message bits to be embedded. Specifically, according to a property of Huffman coding, the lengths of the resulting Huffman codes of word sequences are in reverse proportion to the usage frequencies of the word sequences. So a word sequence with a shorter Huffman code will have a larger probability to be selected, which can be computed as $(1/2)^L$ where L denotes the number of bits of the code. That is, the use of Huffman coding indeed can achieve the aim of selecting word sequences in favor of those which are more frequently corrected in real cases.

But a problem arises here — after we select one word sequence q_y in this way, q_y *cannot* be used in the revision *again* for encoding an identical succeeding code in the message because q_y has already been changed into another word sequence, causing a problem which we call the *selection problem*. This problem comes partially from the *unique decidability* property of Huffman coding. To illustrate this problem, for the previous example as shown in Figure 5 again, the Huffman codes for word sequences q_1 through q_{11} are shown in Figure 6(a), and the message bit sequence to be embedded currently is "100100…" with the first six bits being just two repetitions of the code "100." For this, at first we select word sequence q_4 and change it into another in the revision because the first three message bits to be embedded, "100," are just the code for q_4 (indicated by red color). After this, the next three message bits to be embedded are *again* the code "100" (the blue color of message bits in Figure 6(a)); however, the corresponding word sequence q_4 *cannot* be selected any further because it has already been changed in the current revision version, and other word sequences *cannot* be selected, either, because their codes are not the same as the current message bits "100" to be embedded.

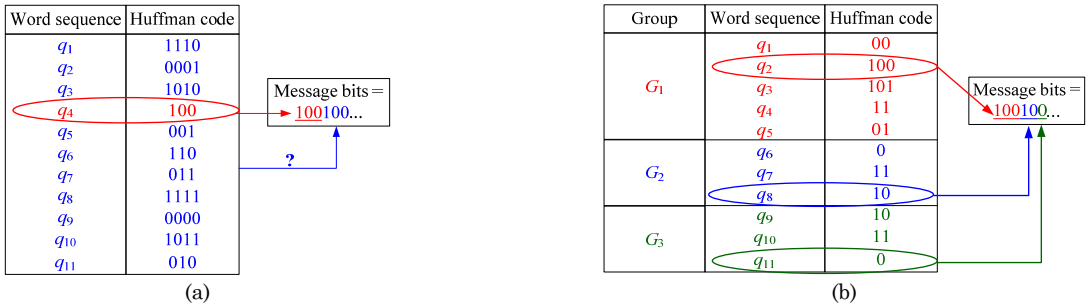

Fig. 6. Illustration of the selection problem. (a) Huffman codes for the word sequences and the message bits that are encountered in the selection problem. (b) Dividing of the word sequences into groups to solve the selection problem.

To solve this selection problem, suppose that based on the use of a key, we assign randomly the word sequences in Q_r consecutively into N_g groups G_1 through G_{N_g}, each group including multiple, but distinct, word sequences, where N_g is the number of word sequences changed in D_{i-1}. Then, starting from group G_1, we apply Huffman coding to assign codes to all word sequences in the currently-processed group G_k according to their *usage frequencies*, and select a word sequence in G_k with its assigned code identical to the leading message bits for use in the revision. We apply this step repetitively until all groups are processed. In this process, Huffman coding is applied to *each* G_k with word sequences distinct from those in the other groups, so that the selection problem of choosing a word *twice* to change due to code repetition in the message will not happen any more. For example, as shown in Figure 6(b), Q_r is divided into three groups: G_1, G_2, and G_3, represented by red, blue, and green colors, respectively. Starting from G_1, we assign Huffman codes to the elements in each group as

shown in Figure 6(b). Then, q_2 will be selected because the code of q_2 is the same as the first three bits "100…" of the message to be embedded. Then, next in G_2, q_8 will be selected because the message bits to be embedded are currently "100…" Finally, q_{11} in G_3 will be selected because the current message bits to be embedded are "0…" In this way, the previous problem of being unable to embed the repetitive code "100" is solved automatically. In short, by decomposing randomly the candidate set Q_r of word sequences for changes into groups and representing each group by a Huffman code, we can embed message bits sequentially by changing only one word sequence in each group without causing the selection problem.

However, the above process is insufficient; it must be modified in such a way that word sequences which have *mutual dependency relations* are divided into an identical group in order to avoid the dependency problem as discussed in Section 3.2.2.1. For this aim, instead of decomposing the word sequences in Q_r directly into random groups as mentioned previously, we divide randomly the *mutually-independent* list elements of I mentioned in Section 3.2.2.1 into N_g groups, where each group is denoted by G_{Ik}. Then, we take out all the word sequences in the lists in each G_{Ik} to form a new group of word sequences, denoted as G_k, resulting again in N_g groups of word sequences. For instance, for the previous example as shown in Figure 5, let $N_g = 2$ and suppose that the list elements of I are decomposed randomly into two groups: $G_{I1} = \{I_1, I_2, I_3, I_4\}$ and $G_{I2} = \{I_5, I_6, I_7\}$. Then, this procedure will yield the two groups of $G_1 = \{q_1, \ldots, q_7\}$ and $G_2 = \{q_8, q_9, q_{10}, q_{11}\}$.

3.2.2.3. *The consecutiveness problem.* As shown in Figure 7(a), for example, the word sequence "increase in" in revision D_{i-1} is seen to become "improve themselves" in revision D_i. This effect comes from two changes made during message embedding: the word sequence "increase" in D_{i-1} was changed to be "improve" in D_i; and the word sequence "in" in D_{i-1} was changed to be "themselves" in D_i. However, because of the *consecutiveness* of the two words "improve" and "themselves" in D_i, the two changes might be considered *as a single one* during secret message extraction, i.e., the word sequence "increase in" in D_{i-1} might be regarded to have been changed to be "improve themselves" in D_i. This ambiguity causes a problem, namely, we cannot know whether a change from a word sequence in D_{i-1} to be another in D_i is from one group or two, or equivalently, we cannot know the *true* number N_g of changed word sequences in D_{i-1}, so that we cannot extract later the embedded messages bits correctly. We call this difficulty in message extraction a *consecutiveness problem*.

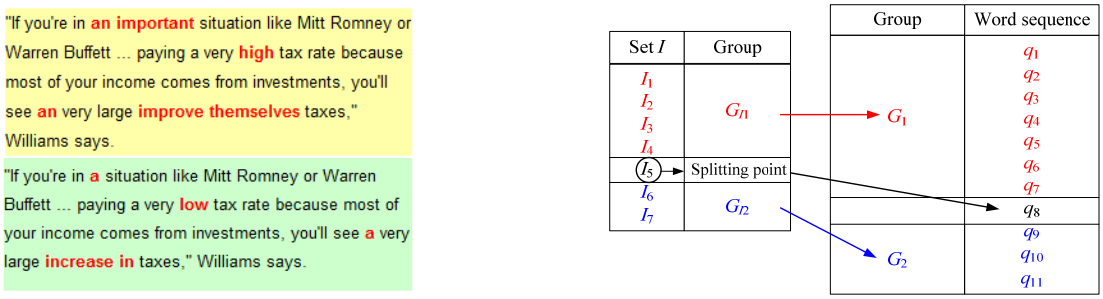

Fig. 7. Illustration of the consecutiveness problem. (a) An example for illustration of the consecutiveness problem. (b) Choosing splitting points randomly to solve the consecutiveness problem.

Obviously, word sequences in different groups must be made *non-consecutive* in order to solve the problem. For this aim, the previously-mentioned solution to the selection problem is modified further. Specifically, by the use of a key again we choose randomly $N_g - 1$ lists, say $I_{i_1}, I_{i_2}, \ldots, I_{i_N}$ (with $N = N_g - 1$), of the set I for use as *splitting points* to divide I into N_g groups with I_{i_1} through I_{i_N} not included in any of the N_g groups. For instance, let $N_g = 2$ for the previous example as shown in Figure 5 and the number of splitting points may be computed accordingly to be $N_g - 1 = 1$. Consequently, as shown in

Figure 7(b), we choose a splitting point, say I_5, to divide the set I into two groups: G_{I1} and G_{I2}, both not including I_5. The final groups of word sequences then become: $G_1 = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$ and $G_2 = \{q_9, q_{10}, q_{11}\}$. Because of the existence of the splitting point $I_5 = (q_8)$, groups G_1 and G_2 are non-consecutive, and accordingly uses of them for creating word sequence changes in revisions will now cause no consecutiveness problem.

3.2.2.4. *The encoding problem.* The issue up to now is how to determine the aforementioned number N_g of word sequences to be changed in D_{i-1}. Although a limit N_c is set for N_g, the maximum number N_m of word sequences that can be selected in D_{i-1} may even be smaller than N_c. Therefore, we must compute N_m first before we can embed message bits according to the number N_g. After N_m is decided, N_g may then be taken to be a number not larger than N_m. The actual value of N_g is decided by the leading secret message bits, say n_m ones. Consequently, we may assume that N_m satisfies the two constraints of 1) $N_m = 2^{n_m}$ and 2) $1 \le N_m \le N_c$, where n_m is a positive integer. In addition, in order to embed message bits by selecting a word sequence from a group G_k, the *number* of elements in G_k should not be smaller than two so as to embed at least one message bit by Huffman coding; hence, each group G_{Ik} mentioned previously should be created to include at least two elements of I. Accordingly, the maximum number N_m of word sequences to be changed in D_{i-1} can be figured out to satisfy the following formula:

$$[N_I - (N_m - 1)]/N_m \ge 2, \tag{1}$$

where N_I is the number of elements in set I and $N_m - 1$ represents the aforementioned number of chosen splitting points. The inequality (1) can be reduced to

$$N_m \le (N_I + 1)/3. \tag{2}$$

Accordingly, we can compute N_m by the following rule:

$$\text{if } (N_I + 1)/3 > N_c, \text{ set } N_m = N_c;$$
$$\text{if } 1 \le (N_I + 1)/3 \le N_c, \text{ set } N_m = 2^{\lfloor \log_2 (N_I+1)/3 \rfloor}. \tag{3}$$

Furthermore, the content of D_{i-1} might be too little for N_m to be decided by Eq. (3). In that case, we abandon the original cover document D_0 from which D_{i-1} is generated, and use another longer cover document as the input. After the value of N_m is computed, we can then use the leading n_m bits of the message to decide the number N_g of changed word sequences in D_{i-1} by two steps: 1) express the first n_m message bits as a decimal number; and 2) increment the decimal number by one. The second step is required to handle the case that the first n_m message bits are all zeros, which leads to the undesired result of no word sequence being changed in the current revision. In this way, N_g becomes really a carrier of n_m message bits. For example, the number of elements of the set I for the previously-mentioned example as shown in Figure 5 is $N_I = 7$. Let $N_c = 4$. Because $(N_I + 1)/3 = (7 + 1)/3 \approx 2.67 \le 4 = N_c$, N_m is computed to be $2^{\lfloor \log_2 (7+1)/3 \rfloor} = 2^1$ according to Eq. (3). So, $n_m = \log_2 N_m = 1$. And if the secret message is "101001…," then the number N_g of changed word sequences should be taken, according to the above two steps, to be $N_g = (1)_2 + (1)_{10} = 2$ because the first bit of the secret message is "1."

3.2.3 *Encoding the Changed Word Sequences in the Current Revision for Data Hiding.* According to the previous discussions, we may assume that we have computed the number N_g of word sequences which should be changed in the current revision D_{i-1} according to the first n_m bits of the secret message, and that we have classified the available word sequences in Q_r into N_g groups, where each group G_k includes at least two word sequences and all word sequences in G_k are encoded by Huffman coding according to their usage frequencies. Specifically, the usage frequency of a word sequence s_j' is taken to be the summation of the correction counts of all the correction pairs in the chosen set of s_j', which have s_j' as their common new word sequence. Then, starting from G_1, we may select from each group G_k one word sequence with a Huffman code identical to the leading bits of the message to be embedded, achieving the goal of data hiding via changing word sequences in D_{i-1}.

For example, assume that the usage frequencies of the word sequences in group G_2 as shown in Figure 7(b) are: q_9 = 100, q_{10} = 50, and q_{11} = 150; and the message is "10100…." Then, the Huffman codes assigned to q_9, q_{10}, and q_{11} are "01," "00," and "1," respectively; and so we select q_{11} to hide the first bit "1" of the message because the code of q_{11} is "1."

3.2.4 *Encoding the Replacing Word Sequences in the Previous Revision for Data Hiding.* Symmetrically, we may use as well the replacing word sequences in D_i to embed message data, where each replacing word sequence s_j in D_i corresponds to a changed word sequence s_j' in D_{i-1}, forming a correction pair $<s_j, s_j'>$. Specifically, recall that for each s_j', we can find a chosen set of correction pairs from DB_{cw}. From this set, we can collect all the *original* word sequences of the correction pairs as another set Q_c', with each word sequence in Q_c' being appropriate for use as the replacing word sequence s_j. Let $Q_c' = \{s_1, s_2, \ldots, s_w\}$. Then, to carry out message data hiding, we encode all s_j in Q_c' by Huffman coding according to their usage frequencies as well, and choose the one with its code identical to the leading message bits for use as the word sequence s_j replacing s_j'. Here the usage frequency of each s_j is the correction count of the correction pair $<s_j, s_j'>$. For example, Table III shows the chosen set of the word sequence "such as" with all included original word sequences already assigned Huffman codes according to their usage frequencies. Based on the table, if the message to be embedded currently is "01001001…," then we change the word sequence "such as" in the current revision D_{i-1} to be the word sequence "for example" in the previous revision D_i because the Huffman code for "for example," namely, 0100, is the same as the first four bits of the secret message.

3.2.5 *Secret Message Embedding Algorithm.* As a summary, we have demonstrated the usability of the aforementioned four characteristics of revisions for data hiding. Therefore, we can generate a stego-document with a forged revision history which looks like a realistic work written by people collaboratively. The details of the proposed message embedding process are described in Algorithm 2 below.

Algorithm 2. Secret message embedding

Input: a cover document D_0 with an article to be revised collaboratively, a binary message M of length t, a secret key K, and a collaborative writing database DB_{cw} constructed by Algorithm 1.

Output: a stego-document D' with a revision history $\{D_0, D_1, D_2, \ldots, D_n\}$.

Stage 1 — message preparation and parameter determination.

1) (*Message composition*) Affix an s-bit binary version of t to the beginning of M to compose a new binary message M', where the value s is agreed by the sender and the receiver beforehand.
2) (*Message encryption*) Randomize M' to yield a new binary message M'' using K.
3) (*Parameter determination*) Use K to decide randomly both the number N_a of authors and the limit N_c on the number N_g of word sequences to be changed in every revision.
4) (*Author encoding*) Use K to select N_a authors from those who were involved in works conducted on the collaborative writing platform to form an author list I_a, and assign a unique n_a-bit code to each selected author in I_a.

Stage 2 — revision generation and message embedding.

5) (*Message embedding and revision generation*) Generate the *previous revision* D_i from the current revision D_{i-1} repeatedly while embedding the binary message M'' by running Algorithm 4 which was designed according to the schemes described in Sections 3.2.1 through 3.2.4 and is shown in the Online Appendix with the inputs D_{i-1}, M'', K, and I_a until all bits in M'' are embedded, where i = 1 initially.
6) If message M'' is not exhausted, then repeat the above process to generate more revisions; otherwise, collect the finally-revised article and the history of all the revisions, D_0 through D_n, as a stego-document D'; and take D' as the output for use on the collaborative writing platform.

3.3 Secret Message Extraction

We can extract the secret message in the stego-document by a reverse version of the message embedding process described by Algorithm 2. The details are described by Algorithm 3 in the following.

Algorithm 3. Secret message extraction

Input: a stego-document D' including revision history $\{D_0, D_1, D_2, \ldots, D_n\}$ of a collaboratively-revised article, the secret key K used in Algorithm 2, and the database DB_{cw} constructed by Algorithm 1.
Output: a binary message M.

Stage 1 — preparation.

1) (*Parameter determination and author encoding*) Use K to decide randomly the number N_a of authors, the limit N_c on the number N_g of word sequences to be changed in every revision, and the list I_a of N_a authors, in the same way as Steps 3 and 4 of Algorithm 2.

Stage 2 — encrypted message extraction.

2) (*Message bit extraction*) For each revision D_{i-1} with $i = 1$ initially, extract the binary message m by running Algorithm 5 which is essentially a reverse version of Algorithm 4 and shown in the Online Appendix with the inputs D_{i-1}, D_i, K, and I_a; and append the result m to a bitstream M'' until $i > n$ where M'' is set empty initially.

Stage 3 — message content recovery.

3) (*Message decryption*) Decrypt the bitstream M'' to get M' using K.
4) (*Message extraction*) Express the first s bits of M' in decimal form as t and output the $(s + 1)$th through $(s + t)$th message bits of M' as the secret message M.

4. EXPERIMENTAL RESULTS

A collaborative writing database DB_{cw} was constructed by mining the huge collaborative writing data in Wikipedia using Algorithm 1 described previously. Note that this is a *totally automatic* work and need be performed only once for building the database DB_{cw} using Algorithm 1, where 3,446,959 different correction pairs were mined from 2,214,481 pages with 33,377,776 revisions in English Wikipedia XML dump. The total size of the downloaded Wikipedia data is about 210.3 GB and the size of the mined data is just 888 MB. Moreover, some revisions might suffer from vandalism [Bronner and Monz 2012; Dutrey et al. 2010], and by the method proposed by Bronner and Monz [2012], such revisions were ignored if they have been reverted due to vandalism. Also, keywords in Wiki markup[1] were ignored as well. Table I shows the top 20 most frequently used correction pairs, where the one in the first place is the pair <"BCE", "BC"> with a correction count of 19,430. Table II shows some correction pairs, each having more than one word either in its original word sequence or in its new word sequence. One of the correction pairs in this table is <"like", "such as"> with a correction count of 773.

The constructed database DB_{cw} contains 1,688,732 chosen sets of correction pairs where all the correction pairs in a chosen set have identical new word sequences, meaning that there are 1,688,732 word sequences which can be chosen and changed to other word sequences in the message embedding phase. Figure 8 shows an illustration of the numbers of entries in the chosen sets with sizes from 2 to 40. Table III shows the content of a chosen set with the new word sequence "such as," as well as the usage frequency and Huffman code for each original word sequence which may be replaced by "such

[1] http://en.wikipedia.org/wiki/Help:Wiki_markup

as" during message embedding. From the table, we can see that the most frequently used original word sequence is "like," so it has the shortest code "1" and the largest probability to be chosen.

Table I. Top Twenty Frequently Used Correction Pairs

Original word sequence	New word sequence	Usage frequency	Original word sequence	New word sequence	Usage frequency
BCE	BC	19,430	the	a	7,009
BC	BCE	17,878	is	are	6,908
color	colour	15,356	a	the	6,278
colour	color	14,852	are	is	5,430
The	the	14,232	colors	colours	5,301
a	an	9,792	colours	colors	5,078
it's	its	9,658	CE	AD	4,833
is	was	9,607	AD	CE	4,262
an	a	8,954	image	Image	4,259
was	is	7,407	was	were	3,924

Table II. Some Correction Pairs Each with More Than One Word Either in the Original Word Sequence or in the New Word Sequence

Original word sequence	New word sequence	Usage frequency	Original word sequence	New word sequence	Usage frequency
Irish evil	Evil	2,367	due to	because of	933
Evil	Irish evil	2,253	like	such as	773
US	United States	1,094	didn't	did not	665
It's	It is	1,052	passed away	died	374
due to the fact that	because	359	doesn't	does not	489
have been	were	348	WWII	World War II	395
will be	was	903	UK	United Kingdom	599

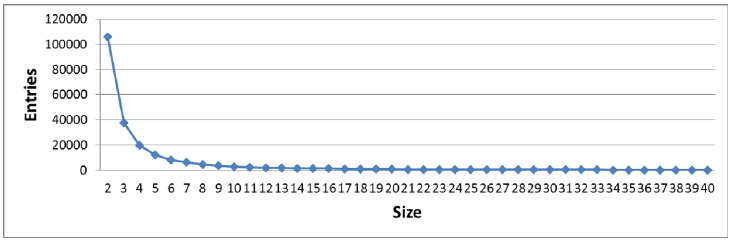

Fig. 8. The number of entries of chosen sets with the size from 2 to 40.

After the message embedding phase, the proposed system will generate a stego-document to be kept in a collaborative writing platform and a user can later extract the embedded message from it using a key. Each generated stego-document including its revision history was kept on a Wiki site which was constructed in this study using the free software: MediaWiki[2]. Note that though here the pre-selected collaborative writing platform is the constructed Wiki site, yet the proposed method can be used on *other* collaborative writing platforms as well. As an example, with a cover article as shown in Figure 9(a), the message "Art is long, life is short," and the key "1234" as inputs into Algorithm 2, a stego-article as shown in Figure 9(c) together with a revision history as shown in Figure 9(b) was

[2] http://www.mediawiki.org/wiki/MediaWiki.

generated by the proposed method. We can see from Figure 9(b) that five revisions have been created in order to embed the secret message. And Figures 9(d) and 9(e) show the extracts of the differences between the two newest revisions, where the words in red in Fig. 9(d) were corrected to be those in red in Figure 9(e) by the author "Natalie." Figures 9(f) and 9(g) shows respectively the messages extracted by Algorithm 3 using a right key and a wrong one. These results show that when a user uses a wrong key, the system will return a random string as the message extraction result.

Table III. An Example of a Chosen Set with the New Word Sequence "such as"

Original word sequence	Usage frequency	Huffman code	Original word sequence	Usage frequency	Huffman code
like	773	1	specifically	12	011001
including	143	00	namely	10	011000
for example	39	0100	particularly	10	0111111
of	29	01110	like the	10	010100
notably	23	01101	most notably	10	010101
especially	20	01011	include	9	0111110
and	16	011110			

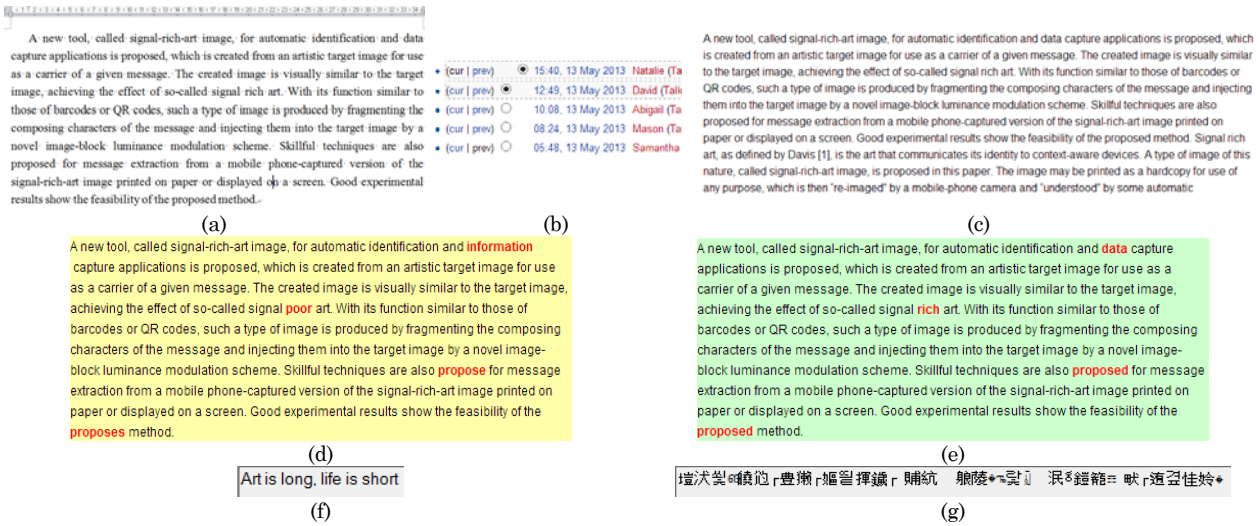

Fig. 9. An example of generated stego-documents on constructed Wiki site with input secret message "Art is long, life is short." (a) Cover document. (b) Revision history (c) Stego-document. (d) Previous revision of revision of (e) with words in red being those corrected to be new words in revision of (e) in red. (e) Newest revision of created stego-document. (f) Correct secret message extracted with the right key "1234." (g) Wrong extracted secret message with a wrong key "123."

A series of experiments with different parameters have also been conducted to quantitatively measure the data embedding capacity of the proposed method using a lot of cover documents as inputs. Since the data embedding capacity is dependent on the secret message content which influences the selections of authors and changed word sequences for each revision, we have run experiments for each document ten times using different messages as inputs, and recorded the average of the resulting data embedding capacities. The parameters of six different cover documents are shown in Table IV. For example, document 1 has 2,419 characters, 641 words, and 80 sentences; document 3 has 10,128 characters, 2,211 words, and so on.

In these experiments, firstly we selected the replacing word sequences for a revision to be the top n most frequently used ones in the database DB_{cw}, where n = 2, 4, 8, 16, 32. Figure 10(a) shows the resulting data embedding capacities from which we can see that the more the selected replacing word sequences, the more the embedded message bits. This result comes from the fact that when more replacing word sequences are available, the constructed Huffman codes will become longer.

We have also conducted experiments on using different numbers of revisions (1, 2, 4, 8) in the generated stego-documents to see the resulting data embedding capacities. Figure 10(b) shows the results which indicate that when the number of revisions in the stego-document is larger, more message bits can be embedded, as expected. This means that if we want to embed a larger secret message, more revisions should be generated. Yet, on a Wiki site, each revision will be stored as its original text without any compression. Thus, a larger storage space is required to store more generated revisions when the secret message is longer. However, one can solve this issue by simply comparing the difference between two adjacent revisions and only storing the difference between them where this comparison function may be provided by other collaborative writing platforms if desired. Furthermore, we can see also from Figures 10(a) and 10(b) that when a cover document has a larger size, the resulting data embedding capacity will be larger as well. Thus, if we want to embed more data, we have to choose a larger cover document.

Table IV. The Information of Experimental Documents

Document	Character	Word	Sentence	Document	Character	Word	Sentence
Document 1	2,419	641	80	Document 4	11,215	2,617	86
Document 2	4,762	956	45	Document 5	26,591	6,180	631
Document 3	10,128	2,211	121	Document 6	60,349	14,306	1,603

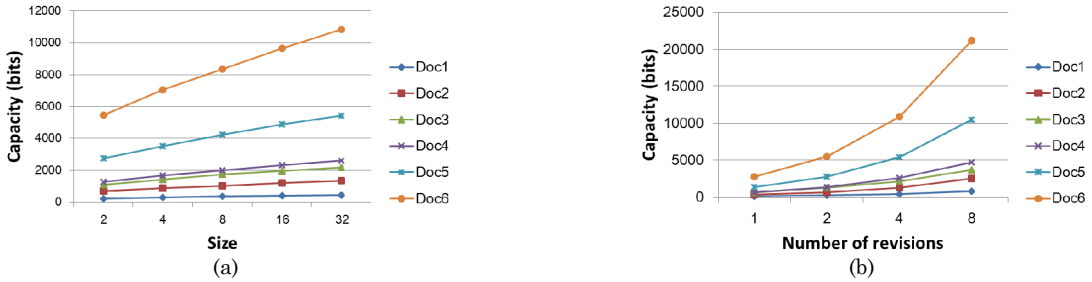

Fig. 10. The embedding capacities. (a) Embedding capacities of documents with chosen sets of different sizes. (b) Embedding capacities of documents with different number of revisions.

Figure 11 shows a comparison of the resulting embedding capacities yielded by the proposed method with those yielded by Liu and Tsai's method [2007]. We can see from Figure 11 that when the number of revisions of the proposed method is equal to one, the embedding capacity of the proposed method is very close to that yielded by Liu and Tsai [2007]. Note that not every word sequence in the current revision D_{i-1} can be utilized for data embedding in the proposed method, because we limit the maximum number of corrected word sequences in a revision. Thus, when the number of revisions is just *one*, the embedding capacity of the proposed method may not be better than that of Liu and Tsai [2007] which allows the use of *every* word for message embedding. However, when the number of revisions is equal to or greater than two, the embedding capacities of the proposed method are instead much larger.

Like the methods proposed by [Bronner and Monz 2012 Bronner et al. 2012] which can be utilized for multiple languages, we have tried to apply Algorithm 1 to two adjacent revisions of a Chinese document and obtain the correction pairs for them successfully, where the two revisions are shown in Figure 12. Note that since Chinese has no explicit word segmentation mark, we cannot use *spaces* to split an article in Chinese into words. Therefore, each character in Chinese was treated as a word directly to solve the issue. Figure 12 shows the found correction pairs between the two revisions, in which, e.g., one of the found correction pairs is <做到, 達成>, where both word sequences in the pair mean the same as "achieve" in English.

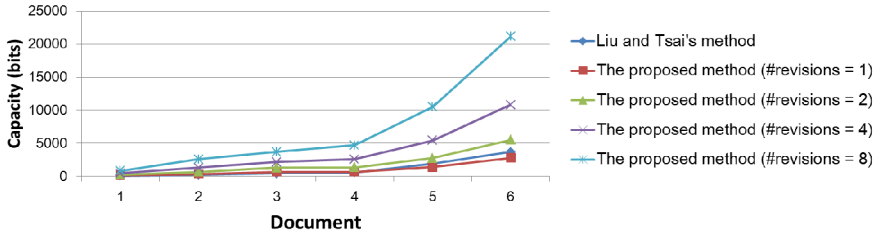

Fig. 11. Comparison of embedding capacities yielded by Liu and Tsai [2007] and proposed method using different numbers of revisions.

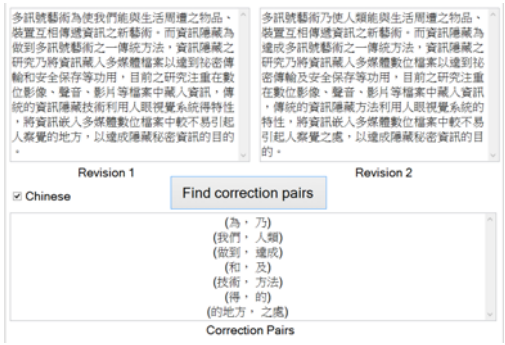

Fig. 12. An example to show the interoperability of the proposed method which can be applied on Chinese articles.

Moreover, for the purpose of presenting the contributions made by the proposed method, we have compared it with several other methods for data hiding via texts [Bolshakov 2005; Chapman et al. 2001; Shirali-Shahreza and Shirali-Shahreza 2008; Liu and Tsai 2007] as shown in Table V. Firstly, the synonym replacement methods [Bolshakov 2005; Chapman et al. 2001; Shirali-Shahreza and Shirali-Shahreza 2008] utilize synonym dictionaries to embed messages, where the synonym dictionaries were usually *manually* built by language experts. And the embedding capacities of these methods are limited, since only those word sequences in the cover document which exist in the synonym dictionary can be utilized for data embedding. Also, since they replace the word sequences in a cover document into their synonyms, the resulting stego-document is usually a worse version of the original cover document due to the possible losses of the original meanings in the replacements. Furthermore, the usage frequencies of the corresponding synonyms of a word sequence are not analyzed in these methods. Secondly, the change tracking method proposed by Liu and Tsai [2007] utilizes synonym dictionaries and a small collaborative writing database with only 7,581 chosen sets to embed messages, where the synonym dictionaries were built *manually* as well. Also, the embedding capacities of this method is limited, since only two revisions are generated by two authors and only the word sequences in the cover document are degenerated for data embedding. Moreover, the usage frequencies of word sequences of this method are just a simulated one created by using the Google SOAP Search API.

As a summary, several merits of the proposed method can now be pointed out, which include: (1) the database of the proposed method is constructed *automatically* from Wikipedia, which is the largest collaborative writing platform on the Internet; therefore, the resulting stego-document generated by the proposed method is more realistic than that generated by the other four methods [Bolshakov 2005; Chapman et al. 2001; Shirali-Shahreza and Shirali-Shahreza 2008; Liu and Tsai 2007]; (2) the dababase constructed by the proposed method is much larger than that by Liu and Tsai [2007], with 1,688,732 chosen sets in the former and only 7,581 in the latter; (3) the usage frequency of each correction pair used in the proposed method is a real parameter obtained by mining the collaborative writing data found on Wikipedia, but that of Liu and Tsai [2007] is just a simulated one created by using the Google SOAP Search API; and (4) the proposed method can simulate the collaborative

writing process conducted by multiple authors and revisions, but Liu and Tsai [2007] can only generate one pre-draft version of a cover text, simulating the work of two authors. Thus, to the best of our knowledge, this is the first work that can simulate the real collaborative writing process with multiple authors and revisions by mining the revision histories on Wikipedia or similar platforms and using the characteristics in the collaborative writing process effectively for message embedding.

Table V. Comparison of Methods for Data Hiding via Texts

Method	Utilized database	Database construction	Embedding capacity	# of revisions	# of authors	Usage frequencies of word sequences
Chapman et al. [2001]	Synonym dictionary	Manually	Limited	1	1	–
Bolshakov [2005]	Synonym dictionary	Manually	Limited	1	1	–
Shirali-Shahreza and Shirali-Shahreza [2008]	Synonym dictionary	Manually	Limited	1	1	–
Liu and Tsai [2007]	Synonym dictionary + Small collaborative writing database	Mainly manually	Limited	2	2	Simulated
Proposed method	Large collaborative writing database	Automatically	Unlimited	Unlimited	Many	Real data

Furthermore, to illustrate the *usability* of the proposed method in the real world, it is pointed out that one can build a collaborative writing platform, such as a Wiki site, for uses by a school, company, or government and then implement the proposed method on this platform. For example, for a school, especially with a large size, the teachers may establish a big wiki site with many documents for general teaching, administration, and communication uses, which are accessible by teachers, staff members, students, parents, etc. Sometimes, a teacher might want to communicate with a student's parents in a secret way. Then, the wiki site may be used as a platform for such covert communication of messages. In addition, the teacher may keep secret records of the students on the wiki site using the data embedding schemes provided by the proposed method. That is, a collaborative writing platform can not only let people work collaboratively but also can let people hide message into the documents existing on this platform for applications of covert communication and secret data keeping.

5. SECURITY CONSIDERATION

5.1 Camouflage

In the proposed method, we collected collaborative writing data in Wikipedia written by real people to construct the database DB_{cw} for use in message embedding. Therefore, the stego-document created using DB_{cw} is more robust to attacks by malicious users since the stego-document looks like a realistic work completed by multiple virtual authors on a collaborative writing platform. These authors do not *actually* edit these revisions and so are regarded as *virtual authors*. These virtual authors are created to simulate the real-world authors and used to embed messages to avoid the problem of involving real authors who might leak the secret. Also, to increase the realisticness of the created stego-document, the content of the corrected word sequences in a revision and the word sequences replacing the corrected ones are selected according to the *real* usage frequencies mined from the collaborative writing data in Wikipedia. Thus, the *statistical property* of simulated corrections in the generated stego-document is close to that of a real one.

Moreover, in order to increase the camouflage effect of the stego-document created by the proposed method, two additional ways can be adopted. The first is to change the time of editing for each generated revision in a stego-document to make it fit the model of revision time in reality, such as the analyzed patterns of revision history mentioned in [Viégas et al. 2004]. This can be achieved by using

a key to select randomly a time for each revision in a possible time duration between the related pair of adjacent revisions. The second way is about the selection of authors for data embedding. If an author makes more realistic corrections in his/her revision history of creating a stego-document, then inclusion of him/her as one of the collaborative authors will cause less conspicuousness to adversaries. This idea can be implemented simply by pre-generating some revision data of virtual authors who looks like owning the real collaborative writing work from the collaborative writing platform, as conducted in the study.

In addition, since we assume that only word sequence corrections will occur in the collaborative writing process, the stego-document created by the proposed method contain only such a type of correction. We can remedy this by manipulating additionally the unused portions of the stego-document to include more types of corrections, such as paraphrases and factual edits, to mislead the adversary, where these extra corrections will be ignored during message extraction.

5.2 Randomness

According to Kerckhoffs' principle [Kerckhoffs 1883], it may be assumed that an adversary, who understands the system but does not have the secret key, can obtain no information about the embedded message. By using the key to enhance the security of the proposed technique, some randomness measures in the phases of secret message embedding and secret message extraction are adopted in the proposed method: (1) randomization of the bits of the secret message to be embedded by encryption; (2) randomization of the parameters and author encoding, including the number of authors, the maximum allowed number of word sequences changed in the revision, the author list, and the author codes; and (3) randomization of the selections of the splitting points for each revision.

More specifically, in the first measure, the secret message is randomized though encryption by using the key, where the encryption method we adopted is AES-256. The Advanced Encryption Standard (AES) is one of the most popular ciphers and provides very high security — the public known attacks up to now have all been shown to be computational infeasible [Bogdanov et al. 2011; Biryukov and Khovratovich 2009]. In the second measure, the parameters (the number of authors and the maximum allowed number of word sequences changed in the revision) and the author encoding (the author list and the author code for each author) are decided by the key and some pseudo-random number generators. In the third measure, for each revision, $N_g - 1$ lists of the set I for use as the splitting points are selected randomly by the key and a pseudo-random number generator. Let the resulting stego-document D' include revision history $\{D_0, D_1, D_2, \ldots, D_n\}$ with N_a authors, the size of the set I of word sequences for selection in each revision D_k be I_k, and the number of word sequences changed in each revision D_k be N_{g_k}. Then, for an adversary who does not have the key, he/she needs to execute Algorithm 3 for all possible combinations of word splitting points of the revisions and the author codes, and observe the result to check the correctness of the encrypted secret message. The time complexity for this work is of the order of $(N_a!)\times\left[\prod_{k=0}^{n-1} C(I_k, N_{g_k} - 1)\right]$ which is a very big number, where $C(a,b)$ means the combination of a things taken b at a time without repetition. Moreover, it is very hard for an adversary to decide which result yielded by the algorithm is correct because the secret message is encrypted by AES-256 and looks like random noise. Therefore, the proposed method is expected to be secure for secret message hiding.

Additionally, the collaborative writing database may be available to adversaries since they can re-construct the collaborative writing database by using the same Wikipedia data and the same algorithms as those proposed in this study. To increase the security against this type of attack, one additional way to increase the robustness of the proposed method is to use the key to decide the subset of a chosen set and select a word sequence from the subset. Therefore, only authorized users with the key can know the correct subset of the chosen set, and an adversary cannot.

5.3 Possible Extensions for the Proposed Method Using Natural Language Processing Methods

For the ability of constructing the collaborative writing database *automatically* and generalizing the proposed method for *multi-language uses*, four characteristics of collaborative writing as mentioned previously have been analyzed based on the *assumption* that only word sequence corrections will be made in a revision. However, the real collaborative writing process is much more complicated and language-dependent, so data hiding via collaborative writing is still worth intensive researches.

Many possible methods in natural language processing [Bronner and Monz 2012; Bronner et al. 2012; Dutrey et al. 2010] may be applied to extend the proposed method. For example, some original word sequences in an input cover document may be *polysemous*. Therefore, selecting appropriate word sequences from DB_{cw} by the proposed method to replace such polysemous word sequences might constitute a meaningless context. One possible way out is to analyze the *distributional similarity* of word sequences [Madnani and Dorr 2010] to find appropriate replacing word sequences that do not cause this problem, where distributional similarity means the similarity in the meanings of those words that have the same contexts in documents. Moreover, we can also build language models [Bronner and Monz 2012; Bronner et al. 2012; Dutrey et al. 2010], such as dependency trees used in grammatical analysis, to embed messages during revision generations based on the model.

6. CONCLUSIONS

A new data hiding method via creations of fake collaboratively-written documents on collaborative writing platforms has been proposed. An input secret message is embedded in the revision history of the resulting stego-document through a simulated collaborative writing process with multiple virtual authors. With this camouflage, people will take the stego-document as a normal collaborative writing work and will not be expected to realize the existence of the hidden message. To generate simulated revisions more realistically, a collaborative writing database was mined from Wikipedia, and the Huffman coding technique was used to encode the mined word sequences in the database according to the statistics of the words. Four characteristics of article revisions were identified, including the author of each revision, the number of corrected word sequences, the content of the corrected word sequences, and the word sequences replacing the corrected ones. Related problems arising in utilizing these characteristics for data hiding have been solved skillfully, resulting in an effective multi-way method for hiding secret messages into the revision history. Moreover, because the word sequences used in the revisions were collected from a great many of real people's writings on Wikipedia, and because Huffman coding based on usage frequencies is applied to encode the word sequences, the resulting stego-document is more realistic than other text steganography methods, such as word-shift methods [Kim et al. 2003], non-displayed characters based methods [Lee and Tsai 2010]], synonym replacement methods [Bolshakov 2005; Chapman et al. 2001; Shirali-Shahreza and Shirali-Shahreza 2008], etc. The experimental results have shown the feasibility of the proposed method. Future works may be directed to analyzing more characteristics of collaborative writing works or establishing appropriate language models [Bronner and Monz 2012; Bronner et al. 2012; Dutrey et al. 2010] for more effective data hiding or other applications.

ACKNOWLEDGEMENT

The authors would like to thank the associate editor, Prof. Pradeep Atrey, and the anonymous reviewers of this paper for their helpful comments as well as suggestions for improving the organization of the paper and the possible ways for improving and extending the proposed method.

REFERENCES

Alattar, A. M. and Alattar, O. M. 2004. Watermarking electronic text documents containing justified paragraphs and irregular

line spacing. In *Proceedings of the SPIE 5306*, *Security, Steganography, and Watermarking of Multimedia Contents VI*, E. J. Delp III and P. W. Wong, Editors, 685–695.
Bennett, K. 2004. Linguistic steganography: Survey, analysis, and robustness concerns for hiding information in text, *CERIAS Tech. Rep. 2004-13*, Purdue Univ., West Lafayette, IN.
Bergroth, L., Hakonen, H. and Raita, T. 2000. A survey of longest common subsequence algorithms. In *Proceedings of the 7th Int. Symp. on String Process. Inf. Retrieval (SPIRE)*, A Curuna, Spain, 39–48.
Biryukov, A. and Khovratovich, D. 2009. Related-key cryptanalysis of the full AES-192 and AES-256. In *Proceedings of the 15th Int. Conf. on The Theory and Application of Cryptology and Information Security (ASIACRYPT)*, Tokyo, Japan, 1–18.
Bogdanov, A., Khovratovich, D. and Rechberger, C. 2011. Biclique cryptanalysis of the full AES. In *Proceedings of the 17th Int. Conf. on The Theory and Application of Cryptology and Information Security (ASIACRYPT)*, Seoul, Korea, 344–371.
Bolshakov, I. 2004. A method of linguistic steganography based on collocationally-verified synonymy. In *Proceedings of the 6th Int. Workshop on Information Hiding Workshop,* Toronto, Canada, 180–191.
Bronner, A. and Monz, C. 2012. User edits classification using document revision histories. In *Proceedings of the 13th Conf. of the European Chapter of the Association for Computational Linguistics*, Avignon, France, 356–366.
Bronner, A., Negri, M., Mehdad, Y., Fahrni, A. and Monz, C. 2012. CoSyne: synchronizing multilingual wiki content. In *Proceedings of the 8th Annual Int. Symp. on Wikis and Open Collaboration (WikiSym)*, Linz, Austria, Article 33, 1–4.
Chapman, M., Davida, G. and Rennhard, M. 2001. A practical and effective approach to large-scale automated linguistic steganography. In *Proceedings of the 4th Information Security Conf.*, Malaga, Spain, 156–165.
Cheddad, A., Condell, J., Curran, K. and McKevitt, P. 2010. Digital image steganography: Survey and analysis of current methods. *Signal Processing 90*, 3, 727–752.
Doerr, G. and Dugelay, J. L. 2003. A guide tour of video watermarking. *Signal Processing: Image Commun. 18*, 4, 263–282.
Dutrey, C., Bernhard, D., Bouamor, H. and Max, A. 2010. Local modifications and paraphrases in Wikipedia's revision history. *Procesamiento de Lenguaje Natural* 46, 51–58.
Erdmann, M., Nakayama, K., Hara, T. and Nishio, S. 2009. Improving the extraction of bilingual terminology from Wikipedia. *ACM Trans. Multimedia Comput. Commun. Appl. 5*, 4, Article 31, 17 pages.
Kerckhoffs, A. 1883. La cryptographie militaire. *J. Sciences Militaires 9*, 5–38.
Kim, Y. W., Moon, K. A. and Oh, S. I. 2003. A text watermarking algorithm based on word classification and inter-word space statistics. In *Proceedings of the 7th Int. Conf. on Document Analysis & Recognition*, Edinburgh, Scotland, UK, 775–779.
Lee, I. S. and Tsai, W. H. 2010. A new approach to covert communication via pdf files. *Signal Processing 90*, 2, 557–565.
Lie, W. N. and Chang, L. C. 2006. Robust and high-quality time-domain audio watermarking based on low-frequency amplitude modification. *IEEE Trans. on Multimedia 8*, 1, 46–59.
Lin, P. Y., Lee, J. S. and Chang, C. C. 2011. Protecting the content integrity of digital imagery with fidelity preservation. *ACM Trans. Multimedia Comput. Commun. Appl. 7*, 3, Article 15, 20 pages.
Liu, T. Y. and Tsai, W. H. 2007. A new steganographic method for data hiding in microsoft word documents by a change tracking technique. *IEEE Trans. on Information Forensics and Security 2*, 1, 24–30.
Madnani, N., and Dorr, B. J. 2010. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics 3*, 3, 341–387.
Max, A. and Wisniewski, G. 2010. Mining naturally-occurring corrections and paraphrases from Wikipedia's revision history. In *Proceedings of the LREC 2010*, Valletta, Malta, 3143–3148.
Mohanty, S. P. and Bhargava, B. K. 2008. Invisible watermarking based on creation and robust insertion-extraction of image adaptive watermarks. *ACM Trans. Multimedia Comput. Commun. Appl. 5*, 2, Article 12, 22 pages.
Nelken, R. and Yamangil, E. 2008. Mining Wikipedia's article revision history for training computational linguistics algorithms. In *Proceedings of the AAAI Workshop on Wikipedia & Artificial Intell.: An Evolving Synergy*, Chicago, Illinois, USA, 31–36.
Shirali-Shahreza, M. H. and Shirali-Shahreza, M. 2008. A new synonym text steganography. In *Proceedings of the Int. Conf. on Intelligent Information Hiding and Multimedia Signal Processing*, Harbin, China, 1524–1526.
Spammimic.com 2010. Spam mimic. http://www.spammimic.com
Stutsman, R., Grothoff, C., Atallah, M. and Grothoff, K. 2006. Lost in just the translation. In *Proceedings of the ACM Symp. Applied Computing*, Dijon, France, 338–345.
Tai, W. L., Yeh, C. M. and Chang, C. C. 2009. Reversible data hiding based on histogram modification of pixel differences. *IEEE Trans. on Circuits and Systems for Video Technology 19*, 6, 904–908.
Viégas, F. B., Wattenberg, M. AND Dave, K. 2004. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the SIGCHI Conf. on Human Factors in Computing Systems*, Vienna, Austria, 575–582.
Wayner, P. 1992. Mimic functions. *Crypt. XVI*, 3, 193–214.
Wayner, P. 2002. Disappearing cryptography: Information hiding: Steganography and watermarking. Morgan Kaufmann Publishers Inc., San Francisco, CA.

Received February 2013, revised May 2013 & September 2013, accepted September 2013.

Online Appendix to: A New Data Hiding Method via Revision History Records on Collaborative Writing Platforms

YA-LIN LEE[1] AND WEN-HSIANG TSAI[1, 2], [1]National Chiao Tung University, [2]Asia University, Taiwan

A. SUPPLEMENTAL ALGORITHMS

The algorithms for message embedding and revision generation which are utilized in the phases of secret message embedding and secret message extraction are described in the following as Algorithms 4 and 5, respectively.

A.1. Message embedding and revision generation

Algorithm 4. Message embedding and revision generation

Input: the current revision D_{i-1}, a binary message M'', a secret key K, an author list I_a, and the collaborative writing database DB_{cw} constructed by Algorithm 1.

Output: a previous revision D_i for D_{i-1}.

Stage 1 — embedding data via author encoding.

1) (*Embedding bits by an author code*) Choose an author a_j for D_i from I_a with its author code being identical to the leading n_a bits of M''; and remove these n_a bits from M''.

Stage 2 — embedding data using the number of changed word sequences in the current revision.

2) (*Finding candidate word sequences for changes in* D_{i-1}) Create a candidate set Q_r of word sequences for changes in D_{i-1} by the *following* steps.
   a) Take in order an unprocessed word w in D_{i-1}, and set q as w initially.
   b) If q is identical to any new word sequence in DB_{cw}, then add q to Q_r and continue; if q matches some leading words in any new word sequence in DB_{cw}, then let q be the concatenation of the old q and the right word of q in D_{i-1} and go to Step 2.b; else, continue.
   c) If there still exists any unprocessed word in D_{i-1}, go to Step 2.a; otherwise, continue.
3) (*Finding independent word sequence lists in* Q_r) Decompose Q_r to form a set $I = \{I_1, I_2, \ldots, I_u\}$ by the following steps.
   d) Take each word sequence in Q_r as a list initially.
   e) Check the *ordered* word sequences in Q_r one by one sequentially:
      if the currently-checked word sequence q_s and its previous one q_t include some common consecutive words, then regard q_s as dependent on q_t, add q_s into the list of q_t in I, and eliminate the list of q_s itself in I; else, keep the list of q_s in I.
4) (*Deciding the number of word sequences to be changed*) Decide the number N_g of groups into which Q_r is to be divided by the following steps.
   f) Compute N_m by Eq. (3) and compute n_m as $\log_2 N_m$.

DOI:http://dx.doi.org/10.1145/0000000.0000000

g) Decide N_g as the decimal value of the first n_m bits of message M'' plus one, and remove these n_m bits from M''.

Stage 3 — embedding data via the word sequences changed in the current revision.

5) (*Choosing splitting points*) Choose randomly $N_g - 1$ elements of I as splitting points using K.
6) (*Classifying the word sequences into independent sets*) Divide the elements of I into N_g groups, G_{I1} *through* G_{INg}, by the splitting points, and take out all the word sequences in the lists in each G_{Ik} to form a new group, denoted as G_k, resulting in N_g groups of word sequences, G_1 through G_{Ng}.
7) (*Choosing word sequences to change*) For each group G_k with k = 1 initially, encode its word sequences by *Huffman* coding according to their usage frequencies, choose and mark the word sequence s_j' in G_k with its code matching the leading message bits of M'' *as* the word sequence to be changed, and remove the matched leading bits from M''.

Stage 2.4 — embedding message data via replacing the word sequences in the previous revision.

8) (*Finding the chosen set*) Find the chosen set of the previously marked s_j' from the correction pairs kept in DB_{cw}, and collect all the original word sequences in the chosen set as a set Q_c'.
9) (*Choosing the original word sequence for use in replacement*) Encode the word sequences s_j in Q_c' by Huffman coding according to their usage frequencies, choose s_j in Q_c' with its code matching the leading message bits of M'', and remove the matched leading bits from M''.
10) (*Conducting word sequence correction*) Replace the word sequence s_j' in D_{i-1} with s_j.
11) (*End of looping*) Increase k by one and go to Step 7 until $k > N_g$.
12) (*Revision generation*) Take the final revised content of D_{i-1} as the desired previous revision D_i.

A.2. Message extraction

Algorithm 5. Message extraction

Input: the current revision D_{i-1}, the previous revision D_i, a secret key K, the author list I_a, and the collaborative writing database DB_{cw} constructed by Algorithm 1.

Output: a binary message m.

Stage 1 — extracting message data from the author.

1) (*Extracting message bits from the author code*) Find the author a_j of the previous revision D_i and let m be the n_a-bit code of this author.

Stage 2 — extracting message bits carried by numbers of changed word sequences in current revisions.

2) (*Finding Q_r and I*) Find Q_r in D_{i-1} and I from Q_r by performing Steps 2 and 3 of Algorithm 4.
3) (*Finding the correction pairs between D_i and D_{i-1}*) Perform Algorithm 1 with D_i and D_{i-1} as the input to get the correction pairs between D_i and D_{i-1}.
4) (*Collecting the information of correction pairs*) Collect all the correction pairs yielded in the last step as a set CP, where each element $cp_o = <w_o, w_o'>$ in CP includes an original word sequence w_o in D_i and a new word sequence w_o' in D_{i-1}.
5) (*Extracting the code for the number of changed word sequences*) Conduct the following steps to extract the code for the number of changed word sequences in D_{i-1}:
   a) compute N_m in D_{i-1} by Eq. (3) and n_m as $\log_2 N_m$;
   b) express the number of elements in set CP, which is also the total number N_g of the changed word sequences in D_{i-1}, as an n_m-bit binary number N_g';
   c) decrement N_g' by one and append the n_m bits of N_g' to m.

Stage 3 — extracting data via changed word sequences in current revisions.

6) (*Choosing splitting points*) Choose splitting points in the same ways as Step 5 of Algorithm 4.
7) (*Classifying word sequences into independent sets*) Perform Step 6 of Algorithm 4 to classify the elements of I into N_g groups.

8) (*Choosing changed word sequences for message extraction*) For each group G_k created in Step 7 with k = 1 initially, encode its word sequences by Huffman coding according to their usage frequencies, and for each s_j' in G_k, check whether s_j' is identical to a new word sequence w_o' in CP: if so, append the code of s_j' to m; else, check the next word sequence in G_k repeatedly.

Stage 4 — extracting message data via replacing word sequences in previous revisions.

9) (*Finding the chosen set*) Find the chosen set of word sequence s_j' from DB_{cw}, and collect all the original word sequences in the correction pairs of the chosen set as a set Q_c'.

10) (*Extracting the code of replacing word sequences*) Encode the word sequences in Q_c' by Huffman coding according to their usage frequencies, and for each s_j in Q_c', check whether s_j is identical to an original word sequence w_o in CP — if so, then append the Huffman code of s_j to m and go to Step 8 with k increased by one until $k > N_g$; else, check the next word sequence in Q_c' repeatedly.