COVERT COMMUNICATION VIA PDF FILES BY NEW DATA HIDING TECHNIQUES*

[1]*Yin-Cheng Lai* (賴以晟) and [2]*Wen-Hsiang Tsai* (蔡文祥)
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
E-mail: [1]troy0420.cs96g@g2.nctu.edu.tw, [2]whtsai@cs.nctu.edu.tw

ABSTRACT

PDF files become very popular nowadays. People use them to exchange information on the Internet, and so it is also a good choice to use PDF files as cover media for covert communication. A method for this purpose by two new data hiding techniques is proposed. The first is a technique based on a white space coding using space-equivalent codes in the PDF file. And the second is a technique of inserting invisible texts into PDF files via the use of artificially created text matrices outside the visible area of the displayed PDF document. A secret key is used to enhance the security of the embedded data. Experimental results show the feasibility of the proposed method.

1. INTRODUCTION

Many kinds of file formats can be used in information communication. The PDF (portable document format) is one of the most common formats and has become more and more popular nowadays because of its advantages in various applications, such as high printing quality and cross-platform applicability. In addition to using PDF files directly to exchange information, it is also advantageous to use them as *cover media* to hide data for secret message transmission and other uses. Although PDF files are popular now, there are yet not many researches on data hiding in PDF files. It is desirable to have more kinds of data hiding techniques for various application purposes.

Several techniques for data hiding in PDF files have been proposed in recent years [2-5]. Zhong and Chen [2] proposed an information steganography algorithm on PDF documents by hiding data between indirect objects of PDF documents. The algorithm can embed data of unlimited lengths into PDF documents and the embedded PDF documents are kept transparent when being displayed in PDF readers. Zhong, Cheng and Chen [3] proposed a steganographic technique for hiding data in a kind of PDF English texts. They modified integer numerals which are used to position characters in the PDF text. Because the perceptual difference is very small, people cannot be aware of the hidden data in the PDF document. Liu et al. [4] proposed an algorithm based on equivalent transformations in PDF files. They discovered that the effect of the page display of a PDF file is extraneous to the seriation of the dictionary's entries so that data hiding can be achieved by arranging special arrays of entries, instead of by operations of adding any other data to the cover PDF. Wang and Tsai [5] proposed a data hiding method by slight modifications of the values of various PDF object parameters, yielding a difference of appearance difficult to notice by human vision.

In this paper, we propose two data hiding techniques via PDF files for secret transmission. The first is based on a *white space coding* scheme. When opening a PDF file with a certain text editor, we can see the initial code of the PDF file, in which there are many *white-space characters* used to separate syntactic constructs from one another. According to the PDF standard [1], many distinct characters are treated as *white-space characters*. We can use this property to embed secret messages. The second technique is based on a scheme of inserting *invisible* texts into a cover PDF. In a PDF file, a *text matrix* decides where and how to display the corresponding text in a PDF document. We may insert some specially-specified text matrices with their coordinates being *outside* the visible area of the PDF so that the corresponding text, where secret messages are embedded, will not show on the displayed PDF documents, thus achieving the effect of data hiding or steganography.

In the remainder of this paper, we give first a general introduction to the PDF in Section 2. In Section 3, we describe the proposed secret transmission method via PDF files by two different kinds of data hiding techniques. And in Section 4, some experimental results are shown. A conclusion is made in Section 5.

2. INTRODUCTION TO PDF

The Adobe portable document format (PDF) is one of the Adobe® Acrobat® family of products [1] and is described by a page description language which is modified from PostScript®. Each PDF *file* represents a

* This work was supported by NSC project No. 97-2631-H-009-001.

document in a manner independent of the application software, hardware, and operating systems used to create the document and of the output device on which the document is to be displayed or printed. The basic elements in a PDF file are objects that together describe the appearance of one or more pages of the PDF document. A PDF document's pages can contain any combination of text, graphics, and images, and a PDF page's appearance is determined by a content stream which contains a sequence of graphics objects to be painted on the page.

According to the PDF standard [1], a canonical PDF file initially consists of four parts (see Figure 1) as follows.

1. A one-line *header* identifying the version of the PDF specification to which the file conforms.
2. A *body* consists of a sequence of indirect objects representing the contents of the PDF document.
3. A *cross-reference table* containing information about the indirect objects in the PDF file, such as the offset of each object which give the object's location in the displayed document.
4. A *trailer* giving the location of the cross-reference table and those of certain special objects within the body of the file.

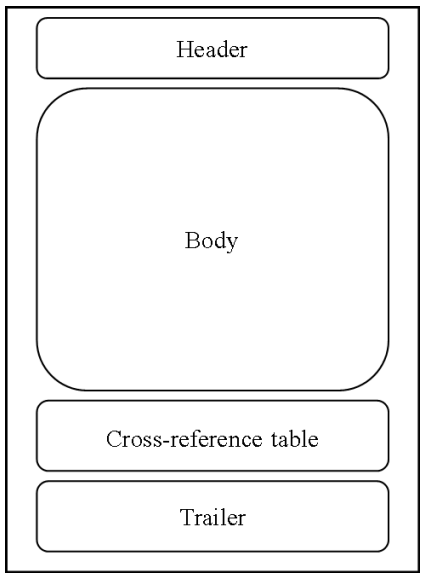


Fig. 1. Structure of a PDF file.

In a PDF file, *white-space characters* are used to separate syntactic constructs from one another. According to the PDF standard, many distinct characters are treated as white spaces. Table 1 shows this property of the PDF. All white-space characters are equivalent, except in comments, strings, and streams. We can use this property to embed secret messages into PDF files, as done in this study. More details will be described in Section 3.1.

A *text matrix* in an indirect object of a PDF file is used to set the state of the corresponding text and locate it. The structure of a text matrix is as follows:

$$a \quad b \quad c \quad d \quad e \quad f \quad \text{Tm}$$

where a through f are all numbers and "Tm" is the operator of the text matrix. The first four numbers a, b, c and d are used for describing text scaling, rotation, and skew. The initial values are 1, 0, 0 and 1, respectively. The other two numbers e and f are the distances to translate the origin of the coordinate system used for PDF document display in the horizontal and vertical dimensions, respectively. In this paper, we use the numbers e and f to embed secret messages. More details will be described in Section 3.2.

Table 1. White-space characters in PDF files.

decimal	hexadecimal	name
0	00	null (NUL)
9	09	tab (HT)
10	0A	line feed (LF)
12	0C	form feed (FF)
13	0D	carriage return (CR)
32	20	Space (SP)

3. DATA HIDING TECHNIQUES FOR SECRET TRANSSMISSION VIA PDF FILES

3.1 Data Hiding in PDF Files by White Space Coding

According to Table 1, a white-space character may described by one of six different codes, namely, the hexadecimal number 00, 09, 0A, 0C, 0D, or 20. After some experiments, we found out in this study that in some text editors, 0C will show as a line and 0A will cause line feeding, in the displayed document of a PDF file containing these two codes. If we use these two codes to embed secret messages, people will be aware of the existence of the hidden data easily by inspecting the displayed PDF document. So we only use 00, 09, 0D and 20 to embed the secret message in this study.

Furthermore, we know that all white-space characters are equivalent except in comments, strings, and streams, so if 00, 09, 0D and 20 are not in comments, strings, and streams, they all *usable* to embed secret messages.

Accordingly, we can embed two bits of message data using a single usable white-space character. That is, we can use 00, 09, 0D and 20 to represent the 2-bit message data 00, 01, 10, and 11, respectively. Table 2 shows this scheme, which we call *white-space coding*.

Table 2. Proposed data embedding scheme.

data to embed	used hexadecimal code
00	00
01	09
10	0D
11	20

For example, if the secret message is 0011011110, we use five white space characters to embed the message and the codes of them are 00, 20, 09, 20, and 0D, representing the in-order bit pairs 00, 11, 01, 11, and 10 in the secret, respectively.

Besides, in order to tell how many white-space characters have been modified for data hiding, we use two bytes to represent the length of the secret message and embed them into the PDF file before embedding the secret message data. These two bytes will be used in the message data recovery process.

Since we do not insert any other data into the cover PDF in the above-described message data embedding process, the size of the PDF file will not change. And so it is difficult for a reader of the displayed stego-PDF document to be aware of the existence of the hidden data in the PDF file. But obviously, the capacity of the data which are embed, called *data embedding capacity* in the sequel, is limited by the number of the usable white-space characters that the cover PDF have.

3.2 Data Hiding in PDF Files by Insertion of Invisible Texts

In this section, the other proposed data hiding technique based on a scheme of inserting invisible texts into a cover PDF is described. The basic idea is that we embed into the PDF file some text matrices whose coordinates are outside the visible area of the PDF so that the corresponding texts will not be shown on the displayed PDF document.

In more detail, given a secret message in the form of a bit string, we transform it, three bits a time, into decimal numbers and process them to define the values of e and f in a number of text matrices. The transformation scheme is specified by Table 3. After the transformation, the secret data become a long string of decimal digits, which we regard as a big integer number and denote as N. Then we process N to define e and f for some text matrices according to the following steps.

1. Create a text matrix M_1 and set e of it as N if N does not cause an overflow, and set f of it as 0.
2. If N causes an overflow, first separate N into two numbers N_1 and N_2, where N_1 is taken to be the leading decimal digits in N which together as an integer do not cause an overflow, and N_2 to be the remaining decimal digits in N. And then take e of M_1 as N_1 and f of M_2 as N_2 if N_2 does not cause an overflow.
3. If N_2 causes an overflow, then separate N_2 into two numbers N_{21} and N_{22} in a similar way so that N_{21} are the leading decimal digits in N_2 which together as an integer do not cause an overflow, and then take N_{21} to be f of M_1, leaving N_{22} to be processed in the next step.
4. Create a second text matrix M_2, and repeat the above three steps *recursively* using N_{22} as input, until all the remaining decimal digits in N_{22} are exhausted.

Note that the range and precision of numbers are limited by the internal representations used in the computer on which the PDF consumer application is running. So when we define the values e and f of the created text matrices using decimal digits in the original N in the above process, overflows might occur. The result of the above recursive process is several text matrices which include all the decimal digits of N as their values of e and f.

In a PDF document, each page of the document is represented by a *page object*—a dictionary that includes references to the page's contents and other attributes. Each page object has a parameter named *MediaBox* which defines the boundaries of the physical medium on which the page is intended to be displayed or printed. In short, the MediaBox decides the visible area of the page. A common visible area of a PDF page is 595×842. In order to guarantee that the position of the text in a text matrix is outside the visible area to create invisibility to an observer of the displayed PDF document, we concatenate "999" before the decimal numbers of e and f of each created text matrix. An example is given as follows.

Suppose the secret message is "010001110." According to Table 3, we transform it into a string of decimal digits "327." Then, we prefix "999" to it to yield N = 999327. Finally, we create a text matrix and put N in it according to the previously-described secret message embedding process. The final text matrix is as shown below:

1 0 0 1 999327 0 Tm.

Table 3. Transformation between binary message string and decimal numbers.

bit segment	decimal number	bit segment	decimal number
000	1	100	5
001	2	101	6
010	3	110	7
011	4	111	8

After we insert message data into the cover PDF file in the above way, the offset of each indirect object and the offset of the cross-reference table in the file may change. Such changes may cause a wrong display of the resulting PDF document. So we have to update the cross-reference table and the trailer of the PDF file to get a correct stego-PDF file. More specifically, suppose that we embed the secret message in an indirect object B to get B'. Since we insert additional data in B, the size of B' is bigger than B. Therefore, the offsets of the indirect objects whose locations are behind B need to be updated by increasing them for a value D which is the difference of the size between B and B'. And if the cross-reference table is also behind B, the trailer needs to be updated by the same way, too.

Since the embedded data will not appear in the displayed PDF document, the *data embedding capacity* is unlimited, but at the cost of increasing the size of the resulting stego-PDF file.

3.3 Proposed Data Hiding Algorithm

We have proposed two data hiding techniques via PDF files. We may use only either of them or both for secret transmission. For the latter case, if the secret message is short enough, then the first technique suffices to embed all the message data into the cover PDF; or if we cannot embed data just by the first technique, then the second technique is used further to embed the remaining data. The detail is described below as an algorithm.

Algorithm 1: *embedding a message in a PDF file.*
Input: a user key K, a secret message S, and a cover PDF file P.
Output: a stego-PDF file P'.
Steps:

1. Encrypt S by a certain method with K (for example, by the DES algorithm), to get encrypted secret data S' and let l be the length of it in bytes.
2. Count the number m of all usable white-space characters in P and compute the embedding capacity n in bytes as $n = m/4$.
3. Separate S' into two parts S_1 and S_2 by the following way.
   3.1 If $n \geq l+2$, take S_1 to be a string composed of S' and prefixed by its length l (expressed as two bytes), and take S_2 to be null.
   3.2 Else, take S_1 to be a string composed of the first $n - 2$ bytes of S' and prefixed by its length $n - 2$ (expressed as two bytes), and take S_2 to be the remaining bytes of S'.
4. Embed S_2 into P in the following way if S_2 is not null.
   4.1 Transform S_2 into a bit sequence and pad S_2 with one or two zeros, if necessary, to make the number of bits in the resulting string S_2' a triple of an integer.
   4.2 Divide S_2' into 3-bit segments $g_1, g_2, \ldots, g_k$, and map $g_1, g_2, \ldots, g_k$ into decimal numbers $g_1', g_2', \ldots, g_k'$, respectively, according to Table 3.
   4.3 Embed g_1' through g_k' into P one by one by the scheme of inserting invisible texts, as described in Section 3.2.
5. Embed S_1 into P in the following way.
   5.1 Transform S_1 into a bit sequence S_1'.
   5.2 Divide S_1' into 2-bit segments $f_1, f_2 \ldots, f_{4w}$, where w denotes the length of original S_1 in bytes.
   5.3 Embed $f_1, f_2, \ldots, f_{4w}$ into P one by one by the scheme of white space coding, as described in Section 3.1.
6. Update the cross-reference table and trailer, if needed, to get a stego-PDF file P' as output.

3.4 Proposed Data Recovery Algorithm

The proposed data recovery process is described as follows, which corresponds to Algorithm 1.

Algorithm 2: *recovering a secret message from a stego-PDF file.*
Input: a user key K and a stego-PDF file P'.
Output: the secret message S hidden in P'.
Steps:

1. Find the first eight usable white-space characters in P' and decode them according to Table 2 to get two bytes of data which contains the number n of the characters embedded in P' by the scheme of white space coding.
2. Find the subsequent n usable white space characters in P', $h_1, h_2 \ldots, h_n$.
3. Transform h_1 through h_n into binary pairs according to Table 2, and concatenate them to get secret data S_1.
4. Scan P' and find out all the text matrices embedded in P' by the scheme of inserting invisible texts by detecting the existence of the prefix "999" in the value e of each checked matrix.
5. Extract sequentially the decimal numbers $g_1, g_2 \ldots, g_k$ from the values of e and f in the text matrices found in the last step.
6. Transform g_1 through g_k into 3-bit segments according to Table 3, and concatenate them to get secret data S_2.
7. Concatenate S_1 and S_2 to get secret data S'.
8. Decrypt S' with the input key K to recover the desired secret message S.

4. EXPERIMENTAL RESULTS

In order to implement the proposed method for covert communication, we designed a user interface written in the language of Java. It supports the following three ways to hide and recover secret messages:

1. using the proposed white space coding technique only;
2. using the proposed technique of inserting invisible texts only;
3. using the proposed method of combining the two techniques together.

In our experiments, we used the combined method of the third way above. Some results of our experiments are shown in Figures 2 through 6. Figure 2 shows the window of the user interface with a secret message and a user key as input, which mean that the user embeds the secret message in a cover PDF by the proposed combined method. The result of the secret recovery with a correct user key is shown in Figure 3. The cover PDF is shown in Figure 4 and the stego-PDF is shown in Figure 5. Comparing the two figures, no change can be seen on the displayed PDF documents. In addition, Figure 6 shows the result of the secret recovery with an incorrect user key.

5. CONCLUSIONS

In this paper, two different data hiding techniques via PDF files and a method for covert communication by combining the two techniques have been proposed. The first technique is based on a white space coding using codes in the PDF file which appear to be white spaces. The second a technique inserts invisible texts into PDF files via the use of artificially created text matrices which appear outside the visible area of the displayed PDF document and so are invisible to the observer of the document. Secret keys are used to enhance the security of the embedded data.

The data embedding capacity of the first data hiding technique is limited by the number of usable white-space characters in the cover PDF while the size of the cover PDF will not change after data embedding because we do not insert any other data in it. The data embedding capacity of the second technique is unlimited so that we can embed a large amount of secret data. If the secret message is short enough, we may just use the first technique; else, the second technique is used subsequently to embed the remaining secret data.

No matter which technique or both are uses to implement covert communication, it has no influence on the display of the PDF file so that people cannot be aware of the existence of the hidden data. Even if an illicit user knows that there is a secret message in the PDF file, the covert message can be protected by a user key, and the illicit user still cannot extract the original secret message. The proposed methods are feasible, as proved by the good experimental results.

Future researches may be directed to applying the proposed data embedding techniques to other information hiding applications, like copyright protection by watermarking, secret authentication, information sharing, etc.

REFERENCES

[1] Adobe Systems Incorporated, *PDF Reference*, Sixth Edition, Addison-Wesley, California, USA, Nov. 2006.

[2] S. Zhong and T. Chen, "Information steganography algorithm based on PDF documents," *Computer Engineering*, vol. 32, no. 3, pp. 161-163, Feb. 2006.

[3] S. Zhong, X. Cheng and T. Chen, "Data hiding in a kind of PDF texts for secret communication," *International Journal of Network Security*, vol. 4, no. 1, pp. 17-26, January 2007.

[4] X. Liu et al., "A steganographic algorithm for hiding data in PDF files based on equivalent transformation," *Proc. of 2008 International Symposiums on Information Processing,* pp. 417-421, Moscow, Russia, May 23-25, 2008.

[5] J. T. Wang and W. H. Tsai, "Data hiding in PDF files and applications by imperceptible modifications of PDF object parameters," *Proc. of 2008 Conf. on Computer Vision, Graphics & Image Proc.*, Yilan, Taiwan, Aug. 24-26, 2008.

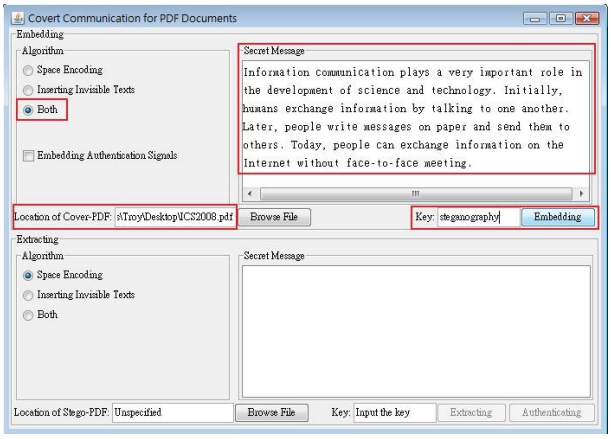


Figure 2. Window of user interface.

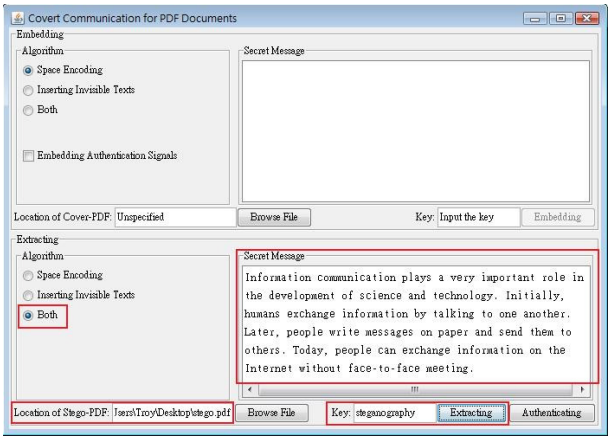
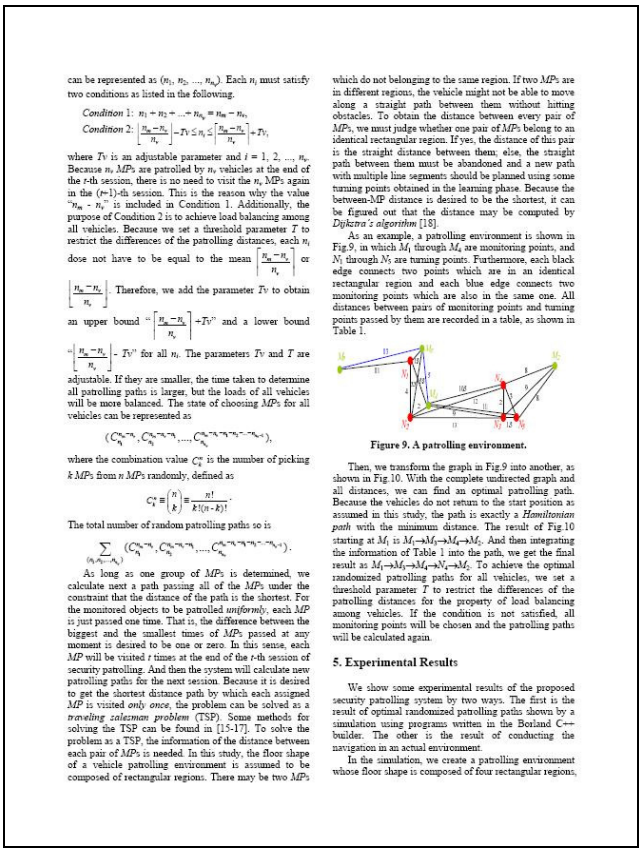


Figure 3. The result of secret recovery with correct user key.



can be represented as $(n_1, n_2, ..., n_{n_p})$. Each n_i must satisfy two conditions as listed in the following.

Condition 1: $n_1 + n_2 + ... + n_{n_p} = n_m - n_v$,

Condition 2: $\left\lfloor \frac{n_m - n_v}{n_p} \right\rfloor - Tv \le n_i \le \left\lceil \frac{n_m - n_v}{n_p} \right\rceil + Tv$,

where Tv is an adjustable parameter and $i = 1, 2, ..., n_p$. Because n_v *MPs* are patrolled by n_v vehicles at the end of the t-th session, there is no need to visit the n_v MPs again in the $(t+1)$-th session. This is the reason why the value "$n_m - n_v$" is included in Condition 1. Additionally, the purpose of Condition 2 is to achieve load balancing among all vehicles. Because we set a threshold parameter T to restrict the differences of the patrolling distances, each n_i dose not have to be equal to the mean $\left\lfloor \frac{n_m - n_v}{n_p} \right\rfloor$ or $\left\lceil \frac{n_m - n_v}{n_p} \right\rceil$. Therefore, we add the parameter Tv to obtain an upper bound "$\left\lceil \frac{n_m - n_v}{n_p} \right\rceil + Tv$" and a lower bound "$\left\lfloor \frac{n_m - n_v}{n_p} \right\rfloor - Tv$" for all n_i. The parameters Tv and T are adjustable. If they are smaller, the time taken to determine all patrolling paths is larger, but the loads of all vehicles will be more balanced. The state of choosing *MPs* for all vehicles can be represented as

$$(C^{n_m - n_v}_{n_1}, C^{n_m - n_v - n_1}_{n_2}, ..., C^{n_m - n_v - n_1 - n_2 - ... - n_{n_p - 1}}_{n_{n_p}}),$$

where the combination value C^n_k is the number of picking k *MPs* from n *MPs* randomly, defined as

$$C^n_k \equiv \binom{n}{k} \equiv \frac{n!}{k!(n-k)!}.$$

The total number of random patrolling paths so is

$$\sum_{(n_1, n_2, ..., n_{n_p})} (C^{n_m - n_v}_{n_1}, C^{n_m - n_v - n_1}_{n_2}, ..., C^{n_m - n_v - n_1 - n_2 - ... - n_{n_p - 1}}_{n_{n_p}}).$$

As long as one group of *MPs* is determined, we calculate next a path passing all of the *MPs* under the constraint that the distance of the path is the shortest. For the monitored objects to be patrolled *uniformly*, each *MP* is just passed one time. That is, the difference between the biggest and the smallest times of *MPs* passed at any moment is desired to be one or zero. In this sense, each *MP* will be visited t times at the end of the t-th session of security patrolling. And then the system will calculate new patrolling paths for the next session. Because it is desired to get the shortest distance path by which each assigned *MP* is visited *only once*, the problem can be solved as a *traveling salesman problem* (TSP). Some methods for solving the TSP can be found in [15-17]. To solve the problem as a TSP, the information of the distance between each pair of *MPs* is needed. In this study, the floor shape of a vehicle patrolling environment is assumed to be composed of rectangular regions. There may be two *MPs* which do not belonging to the same region. If two *MPs* are in different regions, the vehicle might not be able to move along a straight path between them without hitting obstacles. To obtain the distance between every pair of *MPs*, we must judge whether one pair of *MPs* belong to an identical rectangular region. If yes, the distance of this pair is the straight distance between them; else, the straight path between them must be abandoned and a new path with multiple line segments should be planned using some turning points obtained in the learning phase. Because the between-MP distance is desired to be the shortest, it can be figured out that the distance may be computed by *Dijkstra's algorithm* [18].

As an example, a patrolling environment is shown in Fig.9, in which M_1 through M_4 are monitoring points, and N_1 through N_5 are turning points. Furthermore, each black edge connects two points which are in an identical rectangular region and each blue edge connects two monitoring points which are also in the same one. All distances between pairs of monitoring points and turning points passed by them are recorded in a table, as shown in Table 1.

Figure 9. A patrolling environment.

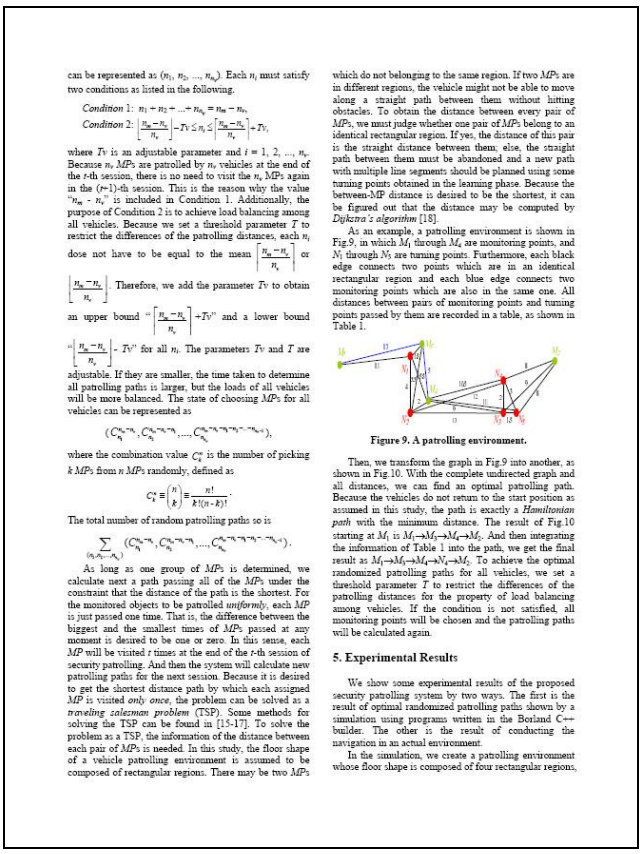
Then, we transform the graph in Fig.9 into another, as shown in Fig.10. With the complete undirected graph and all distances, we can find an optimal patrolling path. Because the vehicles do not return to the start position as assumed in this study, the path is exactly a *Hamiltonian path* with the minimum distance. The result of Fig.10 starting at M_1 is $M_1 \rightarrow M_3 \rightarrow M_4 \rightarrow M_2$. And then integrating the information of Table 1 into the path, we get the final result as $M_1 \rightarrow N_3 \rightarrow M_4 \rightarrow N_4 \rightarrow M_2$. To achieve the optimal randomized patrolling paths for all vehicles, we set a threshold parameter T to restrict the differences of the patrolling distances for the property of load balancing among vehicles. If the condition is not satisfied, all monitoring points will be chosen and the patrolling paths will be calculated again.

5. Experimental Results

We show some experimental results of the proposed security patrolling system by two ways. The first is the result of optimal randomized patrolling paths shown by a simulation using programs written in the Borland C++ builder. The other is the result of conducting the navigation in an actual environment.

In the simulation, we create a patrolling environment whose floor shape is composed of four rectangular regions,

Figure 4. A cover PDF.



can be represented as $(n_1, n_2, ..., n_{n_p})$. Each n_i must satisfy two conditions as listed in the following.

Condition 1: $n_1 + n_2 + ... + n_{n_p} = n_m - n_v$,

Condition 2: $\left\lfloor \frac{n_m - n_v}{n_p} \right\rfloor - Tv \le n_i \le \left\lceil \frac{n_m - n_v}{n_p} \right\rceil + Tv$,

where Tv is an adjustable parameter and $i = 1, 2, ..., n_p$. Because n_v *MPs* are patrolled by n_v vehicles at the end of the t-th session, there is no need to visit the n_v MPs again in the $(t+1)$-th session. This is the reason why the value "$n_m - n_v$" is included in Condition 1. Additionally, the purpose of Condition 2 is to achieve load balancing among all vehicles. Because we set a threshold parameter T to restrict the differences of the patrolling distances, each n_i dose not have to be equal to the mean $\left\lfloor \frac{n_m - n_v}{n_p} \right\rfloor$ or $\left\lceil \frac{n_m - n_v}{n_p} \right\rceil$. Therefore, we add the parameter Tv to obtain an upper bound "$\left\lceil \frac{n_m - n_v}{n_p} \right\rceil + Tv$" and a lower bound "$\left\lfloor \frac{n_m - n_v}{n_p} \right\rfloor - Tv$" for all n_i. The parameters Tv and T are adjustable. If they are smaller, the time taken to determine all patrolling paths is larger, but the loads of all vehicles will be more balanced. The state of choosing *MPs* for all vehicles can be represented as

$$(C^{n_m - n_v}_{n_1}, C^{n_m - n_v - n_1}_{n_2}, ..., C^{n_m - n_v - n_1 - n_2 - ... - n_{n_p - 1}}_{n_{n_p}}),$$

where the combination value C^n_k is the number of picking k *MPs* from n *MPs* randomly, defined as

$$C^n_k \equiv \binom{n}{k} \equiv \frac{n!}{k!(n-k)!}.$$

The total number of random patrolling paths so is

$$\sum_{(n_1, n_2, ..., n_{n_p})} (C^{n_m - n_v}_{n_1}, C^{n_m - n_v - n_1}_{n_2}, ..., C^{n_m - n_v - n_1 - n_2 - ... - n_{n_p - 1}}_{n_{n_p}}).$$

As long as one group of *MPs* is determined, we calculate next a path passing all of the *MPs* under the constraint that the distance of the path is the shortest. For the monitored objects to be patrolled *uniformly*, each *MP* is just passed one time. That is, the difference between the biggest and the smallest times of *MPs* passed at any moment is desired to be one or zero. In this sense, each *MP* will be visited t times at the end of the t-th session of security patrolling. And then the system will calculate new patrolling paths for the next session. Because it is desired to get the shortest distance path by which each assigned *MP* is visited *only once*, the problem can be solved as a *traveling salesman problem* (TSP). Some methods for solving the TSP can be found in [15-17]. To solve the problem as a TSP, the information of the distance between each pair of *MPs* is needed. In this study, the floor shape of a vehicle patrolling environment is assumed to be composed of rectangular regions. There may be two *MPs* which do not belonging to the same region. If two *MPs* are in different regions, the vehicle might not be able to move along a straight path between them without hitting obstacles. To obtain the distance between every pair of *MPs*, we must judge whether one pair of *MPs* belong to an identical rectangular region. If yes, the distance of this pair is the straight distance between them; else, the straight path between them must be abandoned and a new path with multiple line segments should be planned using some turning points obtained in the learning phase. Because the between-MP distance is desired to be the shortest, it can be figured out that the distance may be computed by *Dijkstra's algorithm* [18].

As an example, a patrolling environment is shown in Fig.9, in which M_1 through M_4 are monitoring points, and N_1 through N_5 are turning points. Furthermore, each black edge connects two points which are in an identical rectangular region and each blue edge connects two monitoring points which are also in the same one. All distances between pairs of monitoring points and turning points passed by them are recorded in a table, as shown in Table 1.

Figure 9. A patrolling environment.

Then, we transform the graph in Fig.9 into another, as shown in Fig.10. With the complete undirected graph and all distances, we can find an optimal patrolling path. Because the vehicles do not return to the start position as assumed in this study, the path is exactly a *Hamiltonian path* with the minimum distance. The result of Fig.10 starting at M_1 is $M_1 \rightarrow M_3 \rightarrow M_4 \rightarrow M_2$. And then integrating the information of Table 1 into the path, we get the final result as $M_1 \rightarrow N_3 \rightarrow M_4 \rightarrow N_4 \rightarrow M_2$. To achieve the optimal randomized patrolling paths for all vehicles, we set a threshold parameter T to restrict the differences of the patrolling distances for the property of load balancing among vehicles. If the condition is not satisfied, all monitoring points will be chosen and the patrolling paths will be calculated again.

5. Experimental Results

We show some experimental results of the proposed security patrolling system by two ways. The first is the result of optimal randomized patrolling paths shown by a simulation using programs written in the Borland C++ builder. The other is the result of conducting the navigation in an actual environment.

In the simulation, we create a patrolling environment whose floor shape is composed of four rectangular regions,

Figure 5. The stego-PDF of the cover PDF of Fig. 4.

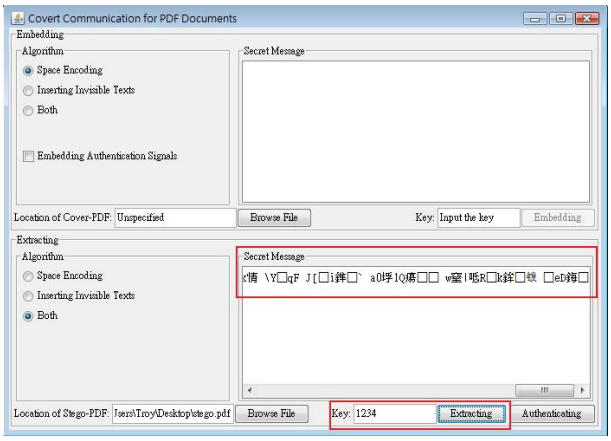


Figure 6. Result of secret recovery with incorrect key.