International Journal of u- and e- Service, Science and Technology
Vol.7, No.2 (2014), pp.27-40
http://dx.doi.org/10.14257/ijunesst.2014.7.2.03

Protection of PDF Files: a Sharing Approach

Suiang-Shyan Lee, Shuo-Fang Hsu and Ja-Chen Lin

Department of Computer Science, National Chiao Tung University,
Hsinchu, Taiwan 30050
{sslee, sfhsu, jclin}@cs.nctu.edu.tw

Abstract

Portable Document Format (PDF) is a popular file format to store documents. Protection of PDF files is thus important. In this paper, we present a (t, n) PDF-sharing scheme which shares the given secret PDF file F among n stego-PDF files. Each stego-PDF file looks completely identical to its original PDF version. When people collect sufficient number (t) of stego-PDF files among the n created stego-PDF files, the secret PDF file F can be revealed error-freely, while less than t stego-PDF files gets nothing but noise. Therefore, our protection not only tolerates missing, but also resists thief's peeping. As a result, the scheme provides the readers a different approach to the protection of important PDF files, in a manner quite distinct from the watermark or encryption approaches used by commercial PDF makers.

Keywords: *Portable Document Format (PDF), protection, sharing, missing-tolerant, peeping-resistant*

1. Introduction

Nowadays, people frequently send electronic files by Internet and store the files in digital devices. How to protect these files against peeping or stealing has been an important issue. Among the existing formats of electronic files, Portable Document Format (PDF) [1-2] is a popular one for documents. PDF format uses an open standard, and users can easily exchange and view PDF files. As stated in the manual of Adobe Portable Document Format Version 1.7 [2], the exchanging and viewing can be independent of the environment in which the PDF files were created. Our paper here discusses the protection of a given important or secret PDF file *F*.

Several PDF makers (for example, Adobe Acrobat Pro) already provided in their PDF file creators certain protection services such as encryption, watermarking, or digital signatures. Some reported papers [3-9] also made certain progresses about the security issue related to PDF files. In the above, the output is usually a single PDF file. However, using a single output PDF file implies that there is no way to recover if the created PDF file is lost (no matter it is due to hackers' attack or natural crash of disk). On the other hand, duplicating the created PDF file and storing in many places will increase the chance of being pirated. To overcome this dilemma, we suggest the use of sharing decomposition, as introduced below.

In this study, the given secret PDF file *F* is decomposed by sharing to get *n* random-noise messages called shadows. Then each message is embedded in an ordinary non-sensitive host-PDF file, and this creates *n* stego-PDF files. Notably, the secret PDF file *F* is revealed in a lossless manner if people collect sufficient number (*t*) of stego-PDF files among the *n* created ones. However, less than *t* stego-PDF files reveal random-noise only.

Some people might wonder why not just use a key to encrypt the secret PDF file to get an encrypted version, then share the key. Although the *n* shadows of the key will be smaller in size than the *n* shadows of the secret PDF; this kind of product is extremely weak. The disk crash or

ISSN: 2005-4246 IJUNESST

hacker attack of the computer, which stores the encrypted version of PDF, will delete the secret PDF forever.

The rest of the article is organized as follows: Section 2 provides a brief introduction to the PDF structure, along with some basic techniques in our previous work. The proposed scheme is illustrated in Section 3. The experimental results and discussions are in Sections 4 and 5, followed by a conclusion in Section 6.

2. A Review of the Structure of PDF File and Sharing

2.1. Structure of PDF File

To help the readers become familiar with the terminology of the PDF structure, a simple PDF file is shown in the appendix. In general, the structure of a PDF file consists of four main sections: a header, a body, a cross-reference table, and a trailer. More specifically, 1) the header identifies the PDF version; 2) the body includes several objects which are the visual components of the file, such as text, images, and pages; 3) the cross-reference table (the xref table) contains several single-line entries each specifies the location of an object within the file so that the objects can be randomly accessed; and 4) the trailer points to the location of the cross-reference table or some special objects.

A PDF document can be regarded as a hierarchy of objects contained in the body section of the PDF file, and the objects are accessed through a tree structure. The root of the hierarchy is the document's catalog dictionary which is pre-defined in the PDF specification [2]. Some categories of objects in a PDF file can be referred to by name rather than by object reference. These objects are established using the document's name dictionary, and the name dictionary can reference them directly without having to use object numbers; for example, some entries in the name dictionary might be called as JavaScript or EmbeddedFiles. In addition, the visible content of the PDF is contained in some objects (*e.g.*, the stream object) of the body section. Moreover, the visible content is often compressed by PDF makers to save space. Therefore, to reproduce the original text of a PDF file which was produced by Microsoft Word software, people have to decompress the file using the zlib/deflate decompression method.

2.2. MOD-based (*t, n*) Secret Sharing [10]

Thien and Lin [10] proposed a (*t*, *n*) threshold scheme to share a secret image *D* among *n* shadows. Then, *D* can be revealed using any *t* of the *n* shadows, whereas less than *t* shadows reveal nothing. To create these *n* shadows, *D* is divided into several non-overlapped sectors of *t* pixels each. Next, grab the first sector to evaluate

$$f(x_i) = (a_0 + a_1 x_i + a_2 x_i^{\,2} + \ldots + a_{t-1} x_i^{\,t-1})_{\mathrm{Mod}}, \quad (1)$$

where $a_{0, \ldots, } a_{t-1}$ are the *t* pixel values of the sector, and x_i is the user-specified Shadow-ID for i^{th} shadow. Finally, the values $f(x_1)$, $f(x_2)$, ... , $f(x_n)$ are appended to Shadows 1, 2, ..., *n*, respectively. This finishes the sharing of the current sector. Then grab next sector to repeat the above sharing process. When all sectors are shared, the sharing process is done. Notably, when sharing phase is ended, each shadow size is 1/*t* of the original secret image size. In the revealing phase, the secret image is revealed if and only if at least *t* of the *n* shadows are gathered. Using Lagrange's interpolation can reconstruct the coefficients $a_{0, \ldots, } a_{t-1}$.

For example, if (*t*, *n*) = (2, 3), then each sector has *t*=2 pixels. Assume the Mod in Eq. (1) is Mod 251. Now, for a sector whose two pixel values are (16; 149), Eq. (1) can be used to create *n*=3 shadow values for this sector; for instance, $f(1)=(16+149\times1)_{\mathrm{Mod251}}=165$,

$f(2)=(16+149\times 2)_{\text{Mod251}}=63$, and $f(3)=(16+149\times 3)_{\text{Mod251}}=212$. So we attach 165 to Shadow 1, and attach 63 to Shadow 2, and attach 212 to Shadow 3. In the revealing phase, whenever a person collects t=2 shadows (*e.g.*, $f(1)$=165 and $f(3)$=212), then the coefficients a_0=16 and a_1=149 of the above-mentioned sector can be unveiled by

$$\begin{aligned} f(z) &\equiv [\, f(1)\times((1-3)^{-1}\times(z-3)) + f(3)\times((3-1)^{-1}\times(z-1))]_{\text{Mod251}} \\ &\equiv [165\times -1\times 2^{-1}\times(z-3) + 212\times 2^{-1}\times(z-1)]_{\text{Mod251}} \\ &\equiv [-165\times 126\times(z-3) + 212\times 126\times(z-1)]_{\text{Mod251}} \\ &\equiv [126\times(165\times 3-212)+126\times(-165+212)\times z]_{\text{Mod 251}} \\ &\equiv [126\times(283)+126\times(47)\times z]_{\text{Mod 251}} \\ &\equiv [16+149\times z]_{\text{Mod251}} \ . \end{aligned} \tag{2}$$

In the above, 2^{-1} = 126 is because $(2\times 126)_{\text{Mod251}}=(252)_{\text{Mod251}}=1$, *i.e.*, 2 and 126 are the multiplication inverse element of each other in the Mod 251 field. Note that Mod 251 can also be replaced by the manipulation in a finite filed called Galois Field (GF), but the computation of +–×÷ should be in terms of the +–×÷ operation in Galois' Field; it is a little more complicated than in Mod 251. However, GF(256) deals with 0-255, whereas Mod 251 deals with 0-250. If the input data are binary, using GF(256) is more natural and also slightly cut the output size of sharing process because 256 is a whole power of 2, but 251 is not. Hereinafter, we will use GF(256). Any reader who is not familiar with the +–×÷ in Galois' Field can consult the example in [11].

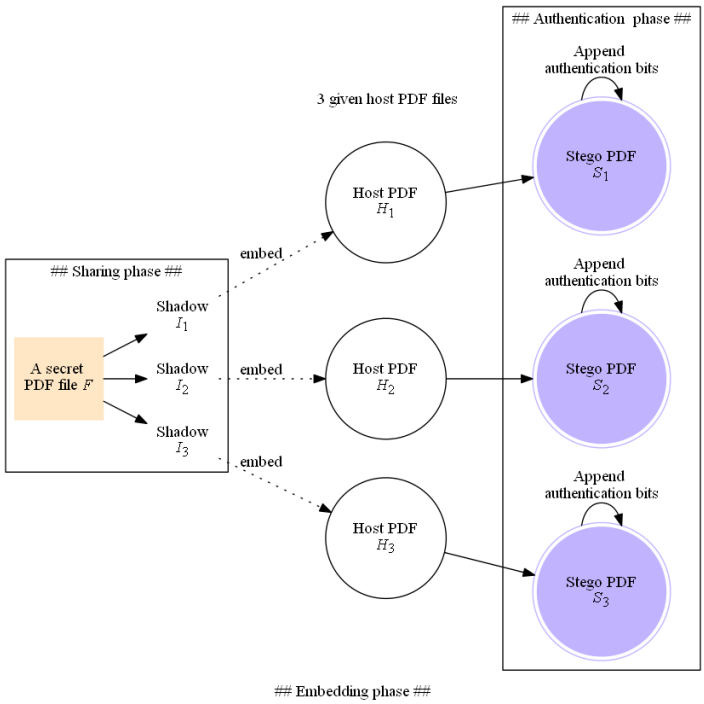


Figure 1. Flowchart of the Encoding (the (*t*, *n*) PDF-Sharing) with *n*=3

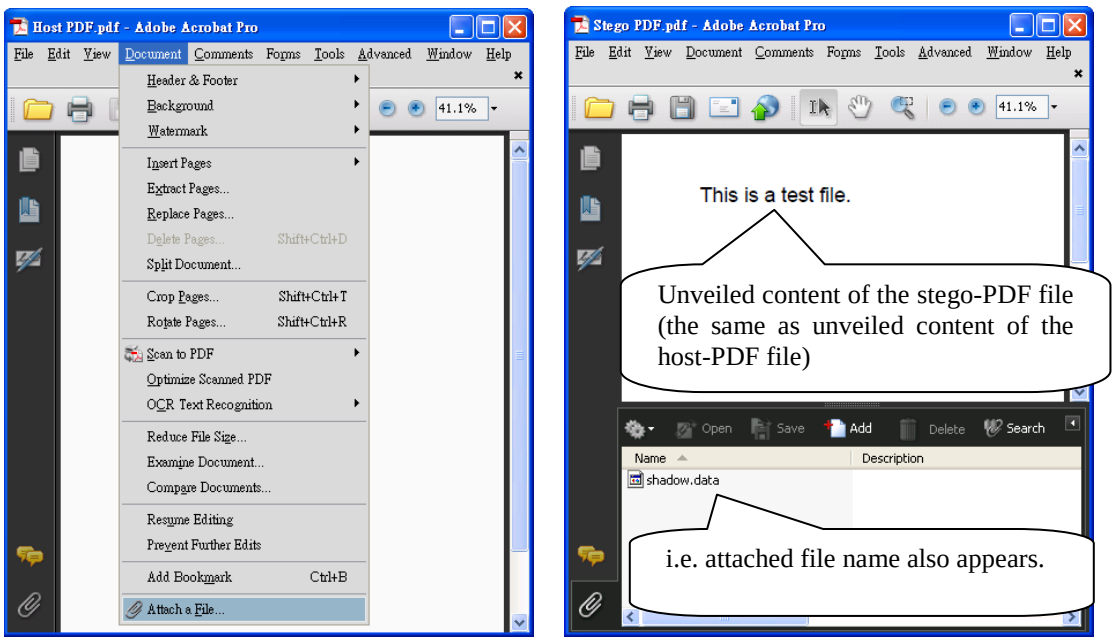
3. The Proposed (*t*, *n*) PDF-sharing Scheme

3.1. Overview

We propose a threshold scheme to share a secret PDF file F among n shadows. Then, to avoid attracting attacker's attention, each shadow is embedded in an ordinary host-PDF file to form a stego-PDF file. Notably, i) each of the n stego-PDF files can display the content and format exactly the same as that of the original host-PDF file; ii) no matter which t of the n stego-PDF files is gathered, the secret PDF file F can be recovered without any error.

The encoding procedure of our design, called (t, n) PDF-sharing, has three phases {sharing; embedding; authentication}. In the sharing phase, we produce n shadows which share the given secret PDF file. In the embedding phase, each of the n host-PDF files embeds a shadow, and the Shadow-ID of that shadow is also embedded in the corresponding host-PDF file using a covert communication method [3]. In the authentication phase, a hash function is used to authenticate the stego-PDF file against fake one.

Figure 1 gives an overview of the (t, n) PDF-sharing for the case n=3. Section 3.2 is the encoding procedure which creates n stego-PDF files. Section 3.3 is the reconstruction procedure which reveals the secret PDF file F from any t of the n stego-PDF files.



(a) (b)

Figure 2. To Attach a File in an Ordinary PDF using Adobe Acrobat Pro. (a) To Attach a File via the Software Adobe Acrobat Pro. (b) The Attached File Name (in this Case, the File Name "shadow.data" in the Figure) will be Seen in certain Area when the PDF is Opened Again. (Hence, we have to Conceal the File Name "shadow.data" Later in our Design)

3.2. The Encoding (the (*t*, *n*) PDF-sharing)

3.2.1. Sharing Phase: The sharing phase uses the technique described in Section 2.2, and then creates the n output shadows $I_1, I_2,..., I_n$. The main steps of sharing phase are as follows:

Input: A secret PDF file F, and a threshold value t.
Step 1: Specify the n user-specified Shadow-ID integers $\{x_1, x_2,..., x_n\}$.
Step 2: Grab next t not-yet-shared bytes of F to form a unit, and then apply to this unit the (t, n) sharing using the operations in Galois' Field. In other words, if the t bytes are $\{a_0, a_1, a_2,..., a_{t-1}\}$, then evaluate

$$f(x_i) = a_0 + a_1x_i + a_2x_i^2 + ... + a_{t-1}x_i^{t-1} \text{ GF(256)} \quad (3)$$

and append the value $f(x_i)$ to shadow I_i.
Step 3: Repeat Step 2 until reaching the end of the file F. (If the finial grabbed unit is fewer than t bytes, zeros are appended so that a complete unit is generated.)
Output: n shadows $\{I_1, I_2,..., I_n\}$ where each shadow is a byte-stream consisting of $\lceil |F|/t \rceil$ shared values, if F has $|F|$ bytes. Since each shared value is a byte, each shadow is t times smaller than the input secret PDF file F.

3.2.2. Embedding Phase: To protect the shadows, each shadow I_i is then embedded in a host-PDF file H_i. In general, if a person wants to attach a file to the PDF file when that PDF file is being created, then most PDF commercial software allow him to embed the attached file directly in the PDF file. So, the external files can be stored or transmitted along with the PDF file. We can use this embedding feature (available for PDF versions not older than Version 1.4) to embed each shadow I_i in its host-PDF file H_i; this simple action is just like the action of attaching a file to an email. When a stego-PDF file is re-opened, it shows the identical content of the host-PDF file, together with I_i. For example, in Figure 2, the file name "shadow.data" also appeared in Figure 2(b) because a shadow file, whose name was "shadow.data", had been embedded earlier in the host-PDF file to get the stego-PDF file. In order to conceal the name "shadow.data" of the attached file when the stego-PDF file is opened, we may use an "anonymous" name dictionary [12]. The skill is explained below. For each pair (I_i, H_i), the steps of embedding I_i in H_i are as follows:
Input: n shadows $I_1, I_2,..., I_n$ and n host-PDF files $H_1, H_2,..., H_n$.
Step 1: Attach each I_i to the corresponding host-PDF file H_i. Because a new object is inserted in H_i, an entry showing the words "*/EmbeddedFiles*" will appear in the name dictionary. (The name dictionary is discussed briefly in the ending paragraph of Sec. 2.1.)
Step 2: Use any text editor to open the file H_i (with attached I_i); for example, open H_i by the commercial software such as Notepad++, UltraEdit, or 010 Editor. When we open PDF file by a text editor, the new object mentioned in Step 1 may look like:

<< /EmbeddedFiles ..... >>.

In the file H_i, find the entry (in this case, "EmbeddedFiles") and locate its byte-position μ_i within H_i. Then, in H_i, the user randomly specifies a word to replace the word "EmbeddedFiles" which is a name already in document's name dictionary. The n randomly-specified words can be different among the n host-PDF files $\{H_i\}$. Just notice that each randomly-specified word must avoid any keyword used in the default PDF entry's names.
Step 3: Update the cross-reference table of H_i in order to keep the position of each object in the body section coincide with its entry in the cross-reference table. Note that the cross-reference table records the byte offsets of each object in the file. Because we modified one of the objects at the previous step, we have to ensure whether any object's position is changed; if a position shifts, we update the number of byte offset in the cross-reference table.

Step 4: Apply the Alternative-Space-Coding method [3] introduced by Lee and Tsai to embed the Shadow-ID x_i in H_i. This guarantees that, in the future, any authorized person who has a stego-PDF file can then locate and grab the Shadow-ID of the shadow hidden in each PDF file. After embedding x_i in H_i, the product is named as stego-PDF file S_i; true for each i in 1,…, n.

Output: The n stego-PDF files S_1, S_2,...., S_n. In addition, the μ_i in Step 2 is also regarded as output bits. Hence, the outputs are the n pairs (S_1, μ_1), (S_2, μ_2),...., (S_n, μ_n).

Below is a remark for Step 1. In Figure 2, we use the software "Adobe Acrobat Pro" to demonstrate how to attach a file; however, the attaching operation can also be done by programming directly, or by some other toolkits. Notably, after Step 1 of the embedding phase, the entry to be concealed (whose original name was the "EmbeddedFiles" in our example) cannot be found/located (even by the reader himself) if the reader's PDF maker compresses the reader's PDF code further. This problem can be solved either by decompressing the PDF code; or, by preventing the file from being compressed when saving the file S_i (*e.g.*, by using "File→Save" instead of "File→Save As..." in Adobe Acrobat Pro software). Anyway, how to get the original syntax plain code is a parsing job of the programmer, and this issue is too far away from our main topic.

Below is a remark for Step 4. The idea of Alternative-Space-Coding is to use two types of white space (*i.e.*, ASCII codes A0 and 20), where A0 and 20 are used in a PDF text alternatively as a between-word space, to encode a secret bit b. The rules are: 4a). If b=1, then use "A0" to express the white space between the two words there. 4b). On the other hand, if b=0, then use "20" to express the white space between the two words there. Besides Alternative-Space-Coding, Lee and Tsai [3] elaborately proposed some other hiding approaches too. Due to the limitation of page, the discussion is omitted here.

3.2.3 Authentication Phase: In order to authenticate each stego-PDF file S_i and avoid using a fake stego-PDF file in the reconstruction, an SHA-256 cryptographic hash function is included in the authentication phase to end the encoding process. For each stego-PDF file S_i, do these steps:

Step 1: Calculate the SHA-256 signature for S_i. This produces a 256-bits hash value ω_i which can identify the S_i.

Step 2: Use the Shadow-ID x_i to encrypt μ_i and ω_i. After that, ω_i and μ_i become $\hat{\omega}_i$ and $\hat{\mu}_i$, respectively.

Step 3: After the keyword syntax "EOF" of the trailer section of S_i, append both the already-further-encrypted signature $\hat{\omega}_i$ and byte-position $\hat{\mu}_i$ to S_i.

3.3. Reconstruction Procedure, i.e., the Inverse Procedure of (*t*, *n*) PDF-sharing

Using I-III below, any user who gets at least t (non-fake) stego-PDF files can reconstruct the secret PDF file F without any loss. The reconstruction can be done independently by that user without asking any other people or system manager to send him any key.

I. (Validation). The user verifies the authentication of a stego-PDF file S_i as follows:

Step 1: Extract the user key x_i from S_i by using the covert approach introduced in Lee and Tsai [3].

Step 2: Extract the two values $(\hat{\omega}_i, \hat{\mu}_i)$ from the region right below the trailer section of S_i. Then, use the key x_i to decrypt $(\hat{\omega}_i, \hat{\mu}_i)$ back to (ω_i, μ_i).

Step 3: Remove the two values $(\hat{\omega}_i, \hat{\mu}_i)$ from the stego-PDF file S_i. Calculate again the SHA-256 signature of the remaining stego file. The calculated signature must equal to the decrypted ω_i; otherwise, the stego file is a faked one and cannot be used in the reconstruction.

II. (Extraction of shadows). For each validated S_i, do the following steps:

Step 1: In the body section of the stego-PDF file S_i, at the position μ_i, store back the 13-letters word "EmbeddedFiles".

Step 2: Update the cross-reference table of S_i in order to keep the position of each object in the body section coincide with its entry in cross-reference table.

Step 3: Extract the shadow file I_i from S_i.

III. (De-sharing). The user can use any t of the shadows $\{I_i\}$ obtained in II to reconstruct the secret PDF file F without any loss. This revealing phase was already described in the example at the end of Sec. 2.2 except that Mod 251 should be replaced by GF(256), for our sharing process (Step 2 of Sec. 3.2.1) used GF(256) rather than Mod 251. Notably, the user reconstructs t bytes of F each time, and it is repeated $|F|/t$ times until the whole file F is revealed. A more thorough description can be found in [10, 13-14].

4. Experiments

This section shows three experiments. Experiment 1 is a (2, 4) PDF-sharing instance. Figure 3(a) is the secret PDF file F. Since each shadow file of F can be treated as a stream of bytes, so each shadow file can be expressed as an 8-bits gray-valued image. The four shadows are shown in Figure 3(b), respectively. Figure 3(c) is the host-PDF file. In Experiment 1, the host-PDF file is duplicated to get four identical copies; the four copies can embed, respectively, the four shadows of the secret PDF file F. The four stego-PDF files are shown in Figure 3(d-g). Obviously, the appearances of all stego-PDF files are exactly the same as the original host-PDF file 3(c). The secret PDF file F is revealed error-freely in Figure 3(h) after running our reconstruction procedure, using "*any*" two of the four stego-PDF files in Figure 3(d-g). Experiment 2 is a (2, 2) PDF-sharing. In Experiment 2, Figure 4(a) is the secret PDF file F, and Figuer 4(b) is the two shadows. The two host-PDF files are different (as shown in Figure 4(c-d)). Finally, a (4, 4) PDF-sharing was conducted in Experiments 3. The file size of all experiments is in Table 1.

Table 1. The Files' Size in our Experiments (unit: byte)

(*t*, *n*)	Secret-PDF *F*	*n* Shadows	*n* Host-PDFs	*n* Stego-PDFs
Experiment 1 (*t*=2, *n*=4)	2,598	1,300	75,159	76,748
		1,300	75,159	76,748
		1,300	75,159	76,748
		1,300	75,159	76,748
Experiment 2 (*t*=2, *n*=2)	56,693	28,348	50,468	70,889
		28,348	49,000	67,670
Experiment 3 (*t*=4, *n*=4)	56,693	14,175	50,468	62,015
		14,175	49,000	63,175
		14,175	68,934	80,697
		14,175	57,521	70,970

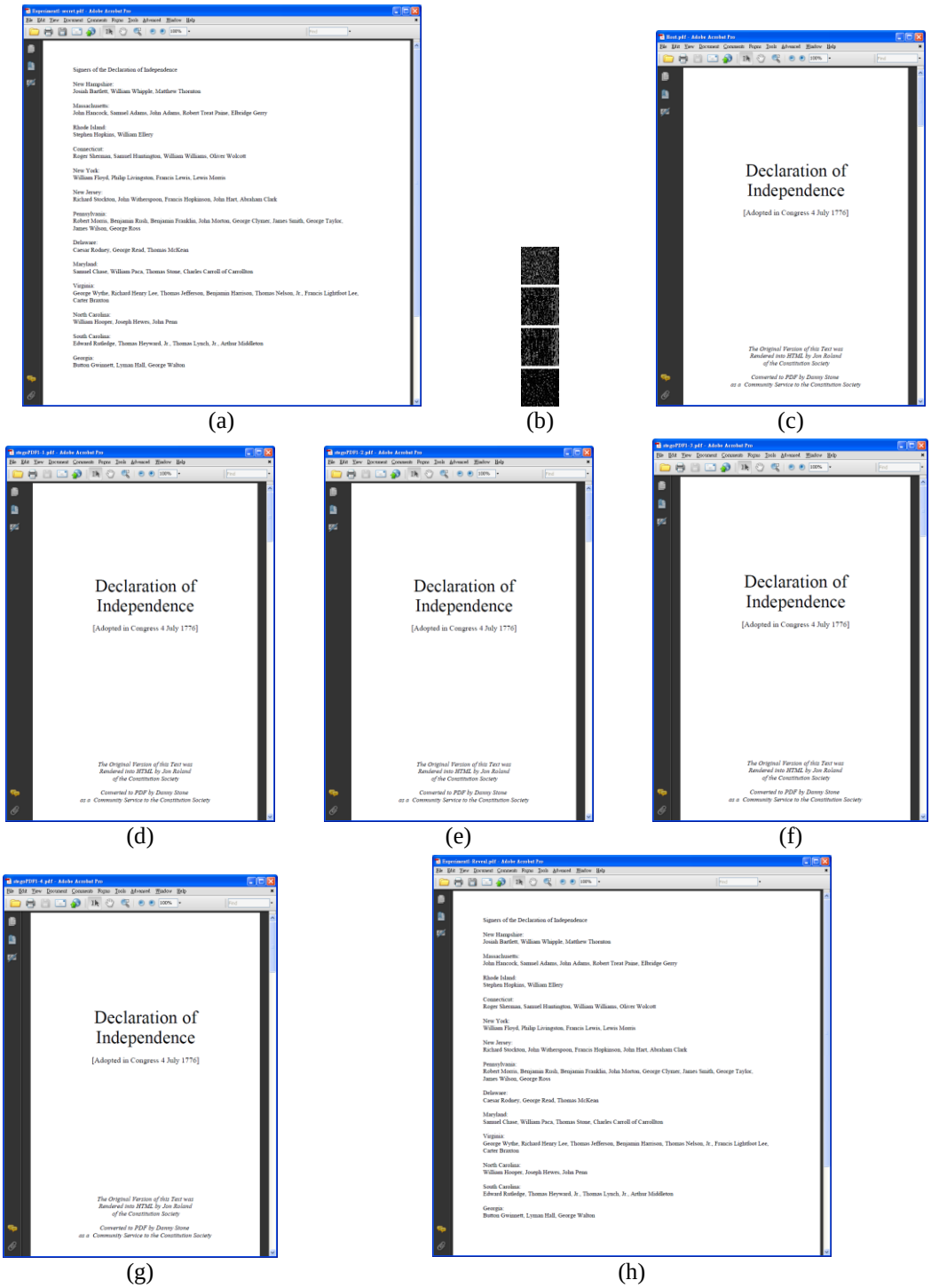


Figure 3. Experiment 1: a (2, 4) PDF-sharing. (a) The Secret PDF File *F*; (b) the Four Shadows {I_1, I_2, I_3, I_4} of *F*; (c) the Host-PDF file that will be Repeatedly used (therefore, H_1=H_2=H_3=H_4); (d-g) the Four Stego-PDF Files {S_1, S_2, S_3, S_4}; (h) the Secret PDF file *F* Reconstructed using "*any*" Two of the Four Stego-PDF Files in (d-g). Notably, (h) is Completely Identical to (a)

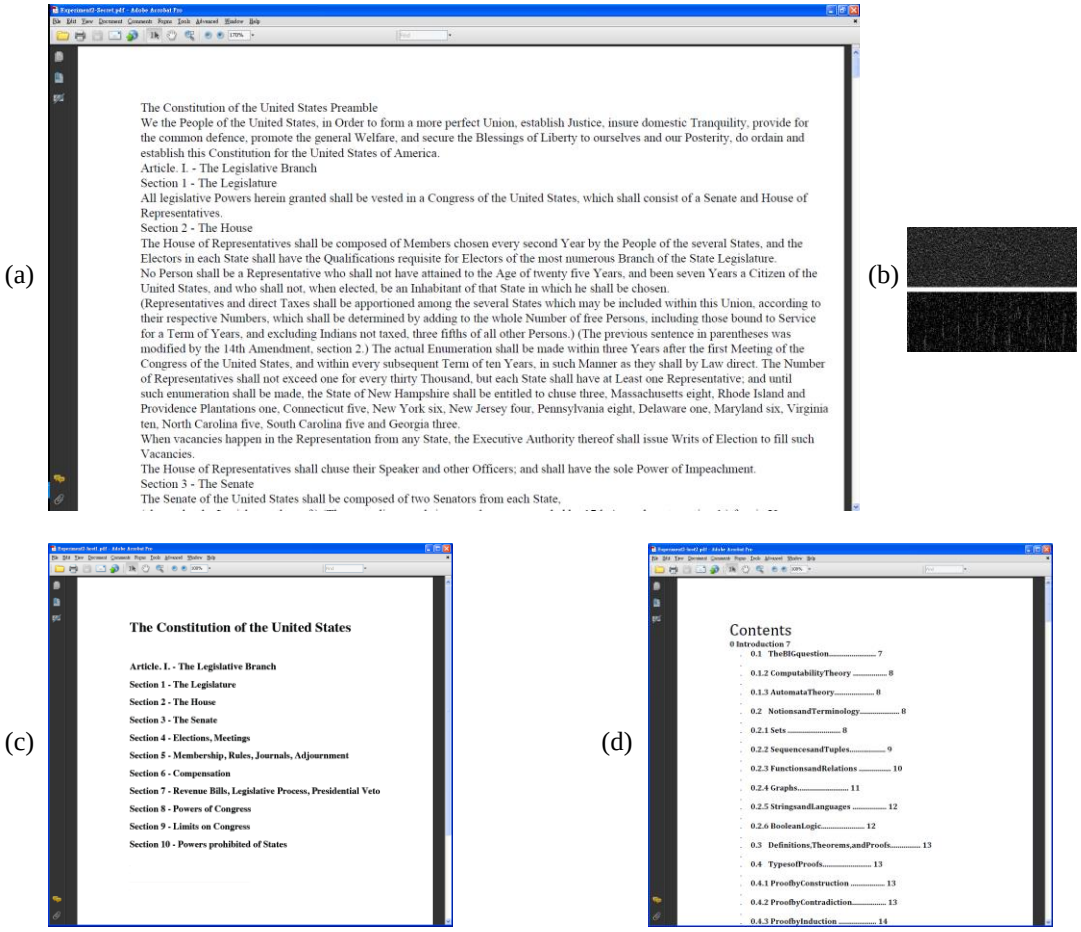


Figure 4. Experiment 2: a (2, 2) PDF-sharing. (a) A Small Area (of Page 1) of the Secret PDF File *F*; (b) the Two Shadows $\{I_1, I_2\}$ of *F*; (c-d) Page 1 of the Two Host-PDF Files $\{H_1, H_2\}$

To save paper length, not all of the figures in Experiments 2 and 3 were shown. Just note that all of the secret PDF files in Experiments 1-3 are reconstructed in a lossless manner, and each stego-PDF file looks completely identical to its host-PDF file, when both are opened by commercial PDF software.

Notably, some readers might want all stego-PDF files look the same for easier management, whereas some other readers might prefer different stego-PDF files for security reason. This gives no difficulty for our design, as shown in Figures 3 and 4.

In our design, some host-PDF files can be even smaller than the secret *F* in size. In Table 1, which describes the files' size in our experiments, the secret file *F* in Experiment 2 is much bigger than the secret file in Experiment 1; in fact, it is even bigger than each host file in Experiment 2, for 56693>Max{50468; 49000}. Since each shadow size is big, the size of stego files also increases much. If a user does not like a big jump of size-ratios between the host and stego files, then he should either use the host-PDF files whose sizes are much bigger than the size of the secret *F* (as in Experiment 1); or, use a larger value of *t*, *i.e.*, requiring more shadows in the unveiling meeting. For example, when the (*t*, *n*) sharing is (t=4, $n\geq$4), and the secret PDF file *F* is still the same *F* in Experiment 2, then the jump of size-ratio between host and stego can be alleviated. One such example is the Experiment 3 (t=4, n=4) whose data size is shown in

final lines of Table 1. In general, a bigger value of t (t=4) here causes a smaller shadow, and hence, a smaller stego-PDF file.

Table 2. Comparing the Embedding Phase of our Method with Reported Hiding Methods that also used PDF File as Host Media. Here, "Type 1" means that "the Hiding is based on Imperceptible Property", and "Type 2" Indicates that "the Hiding is based on the Structure of PDF Document"

Methods	Types	Embedding capacity	Is the capacity influenced much by the "content" of host-PDF file?
Null-Space-Coding [3]	Type 1	*24,026 characters can hide 23,930 characters.	Yes
Alternative-Space-Coding [3]	Type 1	*The embedding rate is about 1.08%.	Yes
Method of Zhong et al. [4]	Type 1	*24,026 characters can hide 5,838 characters (i.e. 46,704 bits).	Yes
Method of Liu et al. [5]	Type 2	*A user-defined ratio (e.g. 5%) according to the host file's size.	a little
Our method	Type 1 Type 2	No limitation in our algorithm, except that the capacity is still constrained by the physical constraint of operating system (e.g. four Giga-bytes in FAT32)	No

Remark: "*" means "quoted directly from the reported paper".

5. Discussions

5.1. The Embedding Phase

A few reported hiding methods are for PDF hiding [3-5]. These PDF hiding methods can at least be classified as two types. Both types of methods yield transparency, *i.e.*, there is no noticeable optical changing when the stego-PDF is displayed by, saying, Adobe. Table 2 gives a brief comparison among the reported methods and the embedding phase of our method. For methods of Type 1 (see Table 2), the hiding utilizes some properties about the imperceptible displayed items of the host-PDF file, such as blank space, character feature, and line width. Both methods [3] and [4] are stunning representatives of Type 1. To achieve the goal of hiding, Lee and Tsai skillfully proposed in [3] the Null-Space-Coding and Alternative-Space-Coding techniques; whereas Zhong et al. gorgeously used in [4] a secret channel to hide data in PDF text. However, as shown in Table 2, our embedding capacity can still compete with that of [3-4]. As for the methods of Type 2, the hiding is based on the structure of PDF document. Method [5] is a typical and attractive method of type 2. In [5], Liu et al. elegantly used several redundant objects by operating the PDF document flows to embed secret data. However, as shown in Table 2, our embedding capacity is still a little larger than theirs.

Besides using the Alternative-Space-Coding, which is a Type-1 hiding method, we also use in our (t, n) PDF-sharing a Type-2 embedding method which was inspired by a security professional report [12]. The Type-2 embedding method won't cause the loss of generality because PDF is a flexible format: general PDF reader is fairly tolerant to deformed files. Our Type-2 embedding algorithm is based on altering the syntax after embedding a file, and there is

almost no limitation about the size of the attached file, except that the capacity is still constrained by the physical constraint of operating system (*e.g.*, 4 Giga-bytes in FAT32). As a result, our embedding capacity can be extremely large.

We embedded our Shadow-ID by a method of Type-1, because the ID of Shadow is usually small in size. In contrast, the Type-2 method that we used in Steps 1-2 of Sec. 3.2.2 provides us a larger hiding capacity. Most of all, as indicated in the final column of Table 2, our shadow can be embedded in any kind of host-PDF file, regardless the content of the host-PDF. Hence, the embedding is very convenient.

5.2. The Sharing Phase and Authentication Phase

Our scheme is based on sharing. The purpose is that a thief cannot reveal the secret PDF file if he collects less than t copies of the host-PDF file. On the other hand, the secret PDF file can be revealed by the union of any t copies of the host-PDF file, even if some of the host-PDF files disappear. So the product is both peeping-resistant and missing-tolerant. We use the PDF-sharing scheme instead of hiding because the secret PDF file can be stolen when attackers are familiar with the hiding methods. The functionality comparisons are shown in Table 3.

Table 3. Functionality Comparisons between some Reported Hiding Methods and our Sharing Scheme

Status / Methods	When attackers have known the method and got a stego-PDF file	When a stego-PDF file is lost
Null-Space-Coding [3]	Attackers can decrypt sensitive messages.	The secret PDF file disappears.
Alternative-Space-Coding [3]	Attackers can decrypt sensitive messages.	The secret PDF file disappears.
Zhong et. al.'s method [4]	Attackers can decrypt sensitive messages.	The secret PDF file disappears.
Liu et. al.'s method [5]	Attackers can decrypt sensitive messages.	The secret PDF file disappears.
Our sharing scheme	**Peeping-resistant**: attackers only derive some noise (shadow).	**Missing-tolerant**: users can retrieve the secret PDF file from the remaining stego-PDF files.

We give an example to illustrate how the sharing scheme can be used. Suppose a company has eight employees, and these workers can access a public folder by login (however, after login, the worker's password will be recorded). Assume our (7, 9) PDF-sharing scheme is utilized; each employee *i* gets stego-PDF file *i*, and the remaining 9-8=1 stego-PDF file is kept in the company's public folder. If an employee wants to decrypt the secret file, he must either cooperate with six other employees, or cooperate with five other employees and then login in the company's public folder to access the 9^{th} stego file. We can even design in such a manner so that the 9^{th} shadow is necessary for secret unveiling, and the 9^{th} shadow is kept by the company owner in his own home or in an unknown place (rather than kept in company), just to avoid betray from a major/minor group of employees. The detail of the design of this necessary share is too tedious and hence omitted here.

The encrypted hash value $\hat{\omega}_i$ provides us a tool to check the stego-PDF file against tampering. We explain below why the hash value ω_i should be transformed to $\hat{\omega}_i$ in the encoding process. Without transforming $\hat{\omega}_i$ to ω_i, a thief can replace our stego-PDF file by his

own faked PDF file G, and attach to it a hash value ω_i(G) of that G. Then we cannot see that the file has been hacked, for the faked file G can pass the hash-value authentication test by using the hash value ω_i(G). To the contrary, because we use $\hat{\omega}_i$, rather than ω_i when we do the authentication test, the transformed hash-value for file G should be $\hat{\omega}_i$ (G), rather than ω_i(G). This enables us to point out that the file G is a faked one.

Notably, the Shadow-ID is included in the stego-PDF file, and this avoids the need of an extra communication request to get Shadow-ID, when the secret is to be unveiled. Our intention is to provide a convenient way for users to obtain the Shadow-ID. However, a reader can exclude the Shadow-ID, too; if he wants to do so. In this case, the Shadow-ID is obtained from extra transmission or accessing a networking portal.

5.3. Control of Size

When we embed a shadow whose size is much bigger than that of the host-PDF file, the size of the stego-PDF file will be much bigger than that of the host-PDF files too. The unusual large size may attract the attackers. In order to reduce the attacker's attention, it is recommended to have a reasonable ratio about the size of stego-PDF file over the size of host-PDF file (the SH ratio). To control, after certain derivation and combination of formulas, we obtain

$$\begin{aligned} \text{SH ratio} &\le 1 + Max\left\{\frac{\alpha_1}{\text{size of } H_1}, \frac{\alpha_2}{\text{size of } H_2}, \ldots, \frac{\alpha_n}{\text{size of } H_n}\right\} \times (\text{share size}) \\ &= 1 + Max\left\{\frac{\alpha_1}{\text{size of } H_1}, \frac{\alpha_2}{\text{size of } H_2}, \ldots, \frac{\alpha_n}{\text{size of } H_n}\right\} \times (\text{secret size}) / t, \end{aligned} \tag{4}$$

where each α_i is the compression rate when an attachment is converted to the PDF syntax. For example, assuming that a user wishes that the target SH ratio should not exceed 120%. Now, if (0.95/1000) is the largest [α_i/(size of H_i)] in Eq. (4); then, by Eq. (4), SH ratio $\le$ 1+(0.95/1000)×(secret size)/t=120%. Hence, the secret size should be less than 0.2×(1000t/0.95). We can conclude that the size |F| of the secret PDF file F should be less than t×200/0.95=(211)t bytes.

6. Conclusion

In this paper, we present a (t, n) PDF-sharing scheme. Our scheme provides the users a different approach to the protection of their secret PDF files. The protection is both peeping-resistant (*i.e.*, no message-leaking if a thief steals less than t shadows) and missing-tolerant (*i.e.*, some shadows can be absent due to disk crash or network disconnection); and hence, non-traditional. Besides these advantages, we also compared the embedding capacity between ours and other kinds of PDF hiding methods. The control to get suitable stego size is also included in this paper. Finally, by adjusting the parameter values t and n in the proposed (t, n) scheme, each person can easily distribute his/her secret in a way customized to meet his/her own need.

Acknowledgements

The work was supported by National Sci. Council, R.O.C., NSC 100-2221-E-009-141-MY3.

Appendix

For easier understanding of the structure of PDF files, we show here a PDF file using a simple text-string example. This single-page PDF file will show only a single line, namely, "*This is a simple PDF file*." The file is opened by NotePad++. As shown in Table 4, the four main sections (i.e. header, body, cross-reference table and trailer) are listed, respectively, in the first, second, third and fourth blocks. To save space, the body and trailer sections are shown in a two-column form. The body section contains five objects; each object has its own number (1 0 obj, 2 0 obj, 3 0 obj, *etc*.). The displayed text ("*This is a simple PDF file*.") belongs to the fourth object (4 0 obj). In addition, there are several already-defined entries (*i.e.*, *keys*) such as "/Type" and "/Kids". Finally, at the end of file, the PDF format shows the syntax "*EOF*".

Table 4. A simple Example to Illustrate Roughly the Syntax of the PDF Format

<table>
<tr><td colspan="2">%PDF-1.4</td></tr>
<tr><td>1 0 obj
<<
/Kids [2 0 R]
/Type /Pages
/Count 1
>>
endobj
2 0 obj
<<
/Rotate 0
/Parent 1 0 R
/MediaBox [0 0 612 792]
/Resources 3 0 R
/Type /Page
/Contents [4 0 R]
>>
endobj
3 0 obj
<<
/Font
<<
/F0
<<
/BaseFont /Times-Italic</td><td>/Subtype /Type1
/Type /Font
>>
>>
>>
endobj
4 0 obj
<<
/Length 78
>>
stream
1. 0. 0. 1. 50. 700. cm
BT
/F0 36. Tf
(This is a simple PDF file.) Tj
ET
endstream
endobj
5 0 obj
<<
/Pages 1 0 R
/Type /Catalog
>>
endobj</td></tr>
<tr><td colspan="2">xref
0 6
0000000000 65535 f
0000000015 00000 n
0000000074 00000 n
0000000192 00000 n
0000000291 00000 n
0000000422 00000 n</td></tr>
<tr><td>trailer
<<
/Root 5 0 R
/Size 6
>></td><td>startxref
472
%%EOF</td></tr>
</table>

References

[1] J. Whitington, "PDF Explained: The ISO standard for document exchange", O'Reilly Media, California, **(2011)**.
[2] Adobe Systems Incorporated, PDF Reference, 6th edition, version 1.7, http://www.adobe.com/devnet/pdf/pdf_reference_archive.html, **(2006)**.
[3] I. S. Lee and W. H. Tsai, "A new approach to covert communication via PDF files", Signal Processing, vol. 90, no. 2, **(2010)**, pp. 557-565.
[4] 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, **(2007)**, pp. 17-26.
[5] Y. Liu, X. Sun and G. Luo, "A novel information hiding algorithm based on structure of PDF document", Computer Engineering, vol. 32, no. 17, **(2006)**, pp. 230-232.
[6] S. Steward, "PDF Hacks: 100 Industrial-Strength Tips Tools", O'Reilly Media, California, **(2010)**.
[7] G. S. Bindra, Editors, "Invisible communication through Portable Document File (PDF) format", Proceedings of the 7th International Conference on IIH-MSP, Dalian, China, **(2011)** October 14-16.
[8] G. S. Bindra, "Masquerading as a trustworthy entity through Portable Document File (PDF) format", Proceedings of the 2011 IEEE International Conference on PASSAT, and IEEE International Conference on Social Com., Boston, USA, **(2011)** October 9-11.
[9] K. F. Rafat and M. Sher, "Secure digital steganography for ASCII text documents", Arabian J. for Science and Engineering, vol. 38, no. 8, **(2013)**, pp. 2079-2094.
[10] C. C. Thien and J. C. Lin, "Secret image sharing", Computers and Graphics, vol. 26, no. 5, **(2002)**, pp. 765-770.
[11] S. J. Lin, L. S. T. Chen and J. C. Lin, "Fast-weighted secret image sharing", Optical Engineering, vol. 48, no. 7, 077008 **(2009)**.
[12] D. Stevens, PDF tools, http://blog.didierstevens.com/programs/pdf-tools/, **(2008)**.
[13] C. C. Lin and W. H. Tsai, "Secret image sharing with capability of share data reduction", Optical Engineering, vol. 42, no. 8, **(2003)**, pp. 2340-2345.
[14] R. Z. Wang and C. H. Su, "Secret image sharing with smaller shadow images", vol. 27, no. 6, **(2006)**, pp. 551-555.

Authors

Suiang-Shyan Lee received his B.S. degree in Computer Science and Information Engineering; and his M.S. degree in Computer Science and Engineering. He is currently a Ph.D. candidate in the Department of Computer Science at National Chiao Tung University, Taiwan. His research interests include image processing and pattern recognition.

Shuo-Fang Hsu received his B.S. degree in Computer and Communication Engineering in 2009, and M.S. degree in Computer Science in 2011. He has been in the Ph.D. program since 2011 in the Department of Computer Science at National Chiao Tung University, Taiwan. His current research interests include data hiding and image sharing.

Ja-Chen Lin received his B.S. and M.S. degrees from National Chiao Tung University, Taiwan. He received his Ph.D. degree in mathematics from Purdue University, U.S.A. In 1984-1988, he was a graduate instructor at Purdue University. He joined the Department of Computer Science at National Chiao Tung University in 1988, and is currently a professor there. Dr. Lin is a member of the Phi-Tau-Phi Scholastic Honor Society. In image processing and pattern recognition, he published about one hundred papers.

