Data Hiding in a Kind of PDF Texts for Secret Communication

Shangping Zhong[1,3], Xueqi Cheng[1,2], Tierui Chen[1,2]
(Corresponding author: Shangping Zhong)

Software Division, Institute of Computing Technology, Chinese Academy of Sciences[1]
P.O. Box 2704, Beijing 100080, P. R. China (Email: zhongshangping@software.ict.ac.cn)
Graduate School of the Chinese Academy of Sciences, Beijing 100039, P. R. China[2]
Department of Computer Science and Technology, Fuzhou University, Fuzhou 350002, P. R. China[3]

(Received Aug. 13, 2005; revised and accepted Oct. 1 and Nov. 5, 2005)

Abstract

In this paper, we present a novel steganographic technique for hiding data in a kind of PDF texts. We first point out the secret channels in a kind of PDF English texts, which are generated from documents that make the texts justified to occupy the full line width and position each character individually. In succession, we describe our steganographic system PDFStego in which several strategies are applied to improve security, such as making use of redundancy to complement security; constituting two chaotic maps to meet the Kerckhoffs principle and to prevent statistical attacks, and applying the secure hash algorithm to enable integrity service and blindly extracting service. Moreover, we define the embedding capacity of our system. PDFStego can be used to exchange sensitive data securely or to add copyright information to the PDF files.

Keywords: Embedding capacity[1]*, PDFStego, PDF text, secret channel, secure strategy, steganography*

1 Introduction

1.1 Background and Related Work

Communication between two parties over long distances has always been subject to interception. This led to the development of encryption schemes. Encryption schemes achieve security basically through a process of making a message unintelligible so that those who do not possess necessary keys cannot recover the message. Though encryption can hide the content of a message, the existence of an encrypted communication in progress can not be hidden from a third party. Also if the third party discovers the encrypted communication, they might be able to decipher the message. The need to avoid this led to the development of steganography schemes which compensate encryption by hiding the existence of a secret communication. Steganography provides good security in itself and when combined with encryption becomes a powerful security tool [13].

The general model of hiding data in other data can be described as follows. The embedded data is the message that one wishes to send secretly. It is usually hidden in an innocuous message referred to as a cover-text, or cover-image or cover-audio as appropriate, producing the stego-text or other stego-object. A stego-key is used to control the hiding process so as to restrict detection and/or the recovery of the embedded data to parties who know it (or who know some derived key value). As the purpose of steganography is having a covert communication between two parties whose existence is unknown to a possible attacker, a successful attack consists in detecting the existence of this communication. Copyright marking, as opposed to steganography, has the additional requirement of robustness against possible attacks [14].

All steganographic techniques share the same basic premise: they take a media and by modifying it in a subtle way, create meaning in it that can only be understood after a knowledgeable party examines it in a special way. Once the data has been embedded, it may be transferred across insecure lines or posted in public places. Three most important properties of steganography schemes are undetectability, perceptual transparency, and capacity. The challenge is to embed as much data as possible without noticeable degradation of the cover media and/or without detectable statistical changes of the cover media under possibly attempted statistical tests. In addition, applying the famous Kerckhoffs principle that security must lie only in the choice of key, we can obtain a definition of a secure stego-system: a stego-system for which an opponent who understands it, but does not know the key, can obtain no evidence (or even grounds for suspicion) that a communication has taken place.

[1]This work is supported by the National High Technology Development 863 Program of China under Grant No.2002AA119010-4.

PDF is a file format used to represent a document in a manner independent of the application software, hardware, and operating system used to create it. A PDF file contains a PDF document and other supporting data. A PDF document contains one or more pages. Each page in the document may contain any combination of text, graphics, and images in a device- and resolution- independent format. A PDF document may also contain information possible only in an electronic representation, such as hypertext links, sound, and movies, etc. [1]. Because of the merits of PDF documents, they have become important interchange information among diverse products and applications.

Now, PDF documents and applications become more and more prevalent, but steganographic algorithms using a PDF document as a cover-object draw little attention within the steganography community. We mention that steganographic algorithms using a PDF image (or other media types) as a cover-object can make use of steganographic algorithms of image or other media types. Of course, some steganographic methods using a structured-text as a cover-text may give us some ideas, for example [3, 4, 8, 9, 10, 11]. Unlike an image or audio, a structured-text has little redundancy information for secret communication. In [3, 4, 8, 9, 10, 11], these steganographic methods are proposed almost by varying the line or word or character spacing or by varying certain character features slightly. In [12], the Steganography based on postscript documents is presented, where info is embedded in spacings, font characteristics (angles, arcs). But these steganographic methods have some weaknesses, for example: A. The embedding capacity is small; B. The embedding capacity is hard to be estimated; C. The security is low, etc.

In addition, the reference [16] introduces a steganography tool wbStego4 which can hide any type of file in Adobe PDF files. Through analyzing the open source, we know that wbStego4 embeds one Byte data between two Indirect Objects in an Adobe PDF file (There are four sections: Header, Body, Cross-reference table and Trailer in an Adobe PDF file, and the Body of an Adobe PDF file is made up of Indirect Objects. Details of PDF structure can be found in [1]). When we use wbStego4, we find that the embedding capacity is very small. When we read the FAQs of wbStego, we also find that hiding data in a PDF file will increase its file size, and there are no general rules for the assessment of the amount of data a PDF file can embed.

1.2 Summary of the Approach

Although text in a PDF file is resolution-independent, there are still reasons to consider the resolution of the target device. Text positioning, in particular, may depend on the primary target device. In PDF files, the positioning model is based on 2D vector-graph positioning model. It is possible to individually position each character in a string that uses, for example, the "TJ" operator. This allows precise layout of the text. In this kind of PDF files, the positioning of each character must be specified by numbers. When we use editor software (for example MS Word) to edit our documents, we often make the text be justified to occupy the full line width, so that the right margin is not ragged. In English text, this edit format makes the position of each character random. Furthermore, when we create the above format PDF files from the edit format text using the PDF Writer, the integer numbers in the "TJ" operator string have enough randomness to supply us with a secret channel.

In this paper, we make use of the secret channel to hide data. A secure steganographic system PDFStego is described in detail in this paper. In PDFStego, applying the page description principle of the "TJ" operator with the property of perceptual transparency, we select some of the integer numerals in the "TJ" operator string to hide data. The strategy of using some redundancy to complement security is applied. The selecting of the integer numerals depends on the key. To meet the Kerckhoffs principle and to prevent statistical attacks, two Logistic chaotic maps [6] have been constituted. In addition, the secure hash algorithm (SHA) [15] is applied to supply integrity service. In the stego-system, we also use some strategies to make us able to blindly extract data from cover-texts easily. In addition, we define embedding capacity of PDFStego. PDFStego can be used to exchange sensitive data securely or to add copyright information to the PDF files (cover-texts).

1.3 Paper Outline

The rest of this paper is organized as follows. Section 2 introduces how to create PDF files as cover-texts. Section 3 presents the secret channel in a kind of PDF texts. The secure steganographic system PDFStego is described in detail in Section 4. Simulation results are given in Section 5, followed by conclusions and future work in Section 6.

2 Creating PDF Files as Cover-Texts

Currently, PDF files may be generated either directly from applications or from files containing PostScript page descriptions. Many applications can generate PDF files directly. As shown in Figure 1, the PDF Writer, available on both Apple® Macintosh® computers and computers running the Microsoft® Windows® environment, acts as a printer driver.

The resulting PDF files are platform-independent. Regardless of whether they were generated on a Macintosh or Windows computer, they may be viewed by a PDF viewing application on any supported platform.

Although text in a PDF file is resolution-independent, there are still reasons to consider the resolution of the target device. Text positioning, in particular, may depend on the primary target device. It is possible to in-

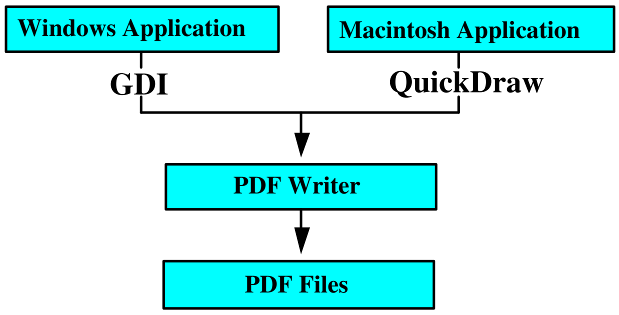


Figure 1: Creating PDF files using PDF Writer

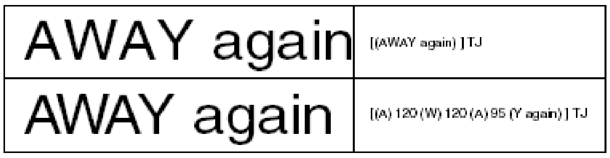
dividually position each character in PDF files which are created from Microsoft Word documents by many widely-used PDF Writers (for example, Jaws PDF Creator [7]). In the next section, we will use this kind of PDF files as cover-texts.

3 Secret Channel in a Kind of PDF Texts

PDF represents text and graphics using the Adobe 2D vector-graph positioning model which is the same model as the one used by PostScript language. Like a PostScript language program, a PDF page description draws a page by placing text and graphics on selected areas.

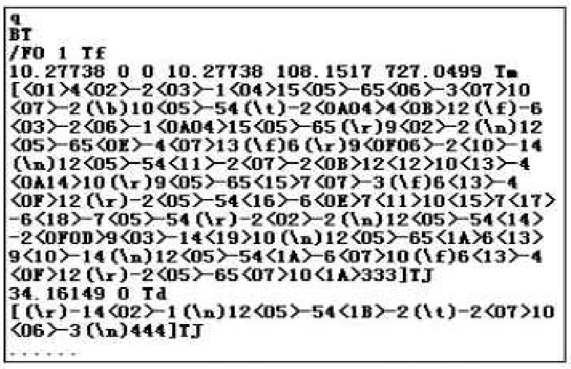
To "draw" text, a number of text objects are encapsulated in a PDF files. A PDF text object consists of operators that specify text state (for example Tf operator), text positioning (for example Tm, Td, TD operator), and text rendering (for example Tj, TJ operator)[1]. For each element of the "TJ" operator strings of the kind of PDF English text, which individually position each character and are created from Microsoft Word documents that make the text justified to occupy the full line width. If the element is a string, "TJ" shows the string; if it is an integer numeral, it is expressed in thousandths of an em. (An em is a typographic unit of measurement equal to the size of a font. For example, in a 12-point font, an em is 12 points.) "TJ" subtracts this amount from the current x coordinate in horizontal writing mode, or from the current y coordinate in vertical writing mode. In the normal case of horizontal writing in the default coordinate system, this has the effect of moving the current point to the left by the given amount. An example of the use of "TJ" is shown in Figure 2 [1].

In the kind of PDF texts which individually position each character, the "Tc" and "Tw" operators are not applied to justify characters and words spacing settings. A page-description example of a part of a PDF text is shown in Figure 3. The strings in the angle brackets are multi-byte encodings of characters, and the integer numerals between the two angle brackets are used to position each



AWAY again	[(AWAY again)] TJ
AWAY again	[(A) 120 (W) 120 (A) 95 (Y again)] TJ

Figure 2: Operation of "TJ" operator



```
q
BT
/F0 1 Tf
10.27738 0 0 10.27738 108.1517 727.0499 Tm
[<01>4<02>-2<03>-1<04>15<05>-65<06>-3<07>10
<07>-2(\b)10<05>-54(\t)-2<0A04>4<0B>12(\f)-6
<03>-2<06>-1<0A04>15<05>-65(\r)9<02>-2(\n)12
<05>-65<0E>-4<07>13(\f)6(\r)9<0F06>-2<10>-14
(\n)12<05>-54<11>-2<07>-2<0B>12<12>10<13>-4
<0A14>10(\r)9<05>-65<15>7<07>-3(\f)6<13>-4
<0F>12(\r)-2<05>-54<16>-6<0E>7<11>10<15>7<17>
-6<18>-7<05>-54(\r)-2<02>-2(\n)12<05>-54<14>
-2<0F0D>9<03>-14<19>10(\n)12<05>-65<1A>6<13>
9<10>-14(\n)12<05>-54<1A>-6<07>10(\f)6<13>-4
<0F>12(\r)-2<05>-65<07>10<1A>333]TJ
34.16149 0 Td
[(\r)-14<02>-1(\n)12<05>-54<1B>-2(\t)-2<07>10
<06>-3(\n)444]TJ
......
```

Figure 3: A page-description example of a part of a PDF text

character or word. Intuitively, the integer numerals which are used to position words (for example 65,54,333) are bigger than those which are used to position characters (for example 4, 2, 1, K).

When making an English text justified to occupy the full line width, the x coordinates (in horizontal writing mode) of characters are random, especially after being changed into integer numerals in PDF texts. Therefore, we can make use of the integer numerals to hide data.

On the other hand, from the page description principle of the "TJ" operator, we know that a PDF text will not be changed its perceptual transparency property if we only change a part of small integer numerals which are used to position characters in the PDF text. Because if it is a small integer numeral, let i be the integer numeral, for example: $1 \le i \le 6$, and let i' be the changed integer numeral, $1 \le i \le 16$, then the difference of perception is as follows:

$\frac{abs(i-i')\times em}{1000}$, "em" is a typographic unit of measurement as defined above, and "abs" is the absolute-value function. Intuitively, when the em (or font size) has a general size, the perceptual difference is very small. We define that the small integer numerals is the embedding units.

In the next section, we will discuss how we select the small integer numerals to hide data, and how we design the secure steganographic system.

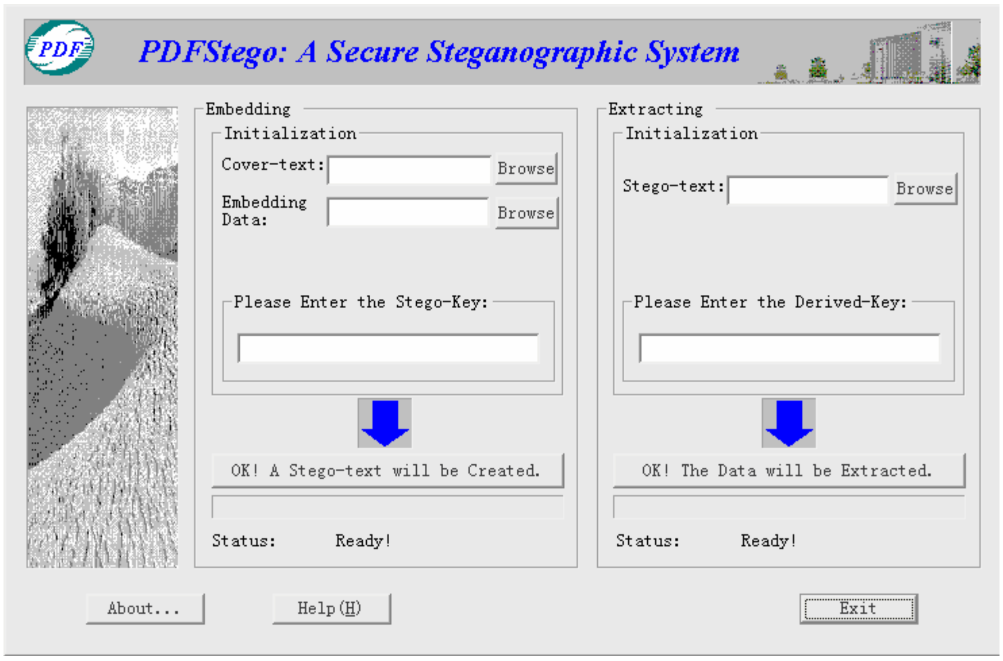

PDFStego: A Secure Steganographic System
Embedding
Initialization
Cover-text:
Browse
Embedding Data:
Browse
Please Enter the Stego-Key:
OK! A Stego-text will be Created.
Status: Ready!
Extracting
Initialization
Stego-text:
Browse
Please Enter the Derived-Key:
OK! The Data will be Extracted.
Status: Ready!
About...
Help(H)
Exit

Figure 4: The user interface of PDFStego

4 PDFStego: A Secure Steganographic System

4.1 The User Interface of PDFStego

The user interface of PDFStego is shown in Figure 4. From Figure 4, the kind of PDF files which are presented by the above section are regarded as cover-texts, and various media data can be embedded in one PDF file. PDFStego is a symmetric steganographic system, thus, the stego-key and derived-key should be exchanged by other secret channel.

4.2 Approach of Randomly Selecting the Embedding Units Based on Logistic Chaotic Map

The famous Logistic chaotic map[6] is defined as follow: $X_{n+1} = f(x_n, /mu) = /mux_n(1 - x_n)),\ 0 \leq \mu \leq 4,\ x \in [0, 1]$. When $3.57 < /mu < 4$, the iteration values $(x_0, x_1, \ldots, x_n, \ldots \in [0, 1])$ are random.

We define the parameter of redundancy is $\phi \in [0, 1]$, for example, $\phi = 10\%$. According to Equation (1), if $y_n = 1$, we use the n th embedding unit to embed data.

$$y_n = \begin{cases} 1, x_n \leq (1 - \phi) \\ 0, x_n > (1 - \phi) \end{cases} \qquad (1)$$

4.3 Hiding any Type of Media File in One PDF File

In the embedding algorithm of PDFStego, we regard a variety of media file as a binary stream, and change every 4 bits data into a "015" integer numerals by the following mapping: $0000 \rightarrow 0$; $0001 \rightarrow 1$; ......; $1111 \rightarrow 15$ (In the extracting algorithm, the anti-mapping is applied). Then, we hide these "015" integer numerals into the embedding units defined in the Section 3 by using the following embedding algorithm and extracting algorithm. Obviously, we can also change every 3 bits(or 2 bits, or 5 bits, ...) data into a "07"(or "03", or "031", ...) integer numerals by the following mapping: $000 \rightarrow 0$; $001 \rightarrow 1$; ......; $111 \rightarrow 7$ (or ...), and select "18"(or "14", or "132", ...) integer numerals in a PDF text as the embedding units.

As we know, "015"(or "07", ...) decimal integer numerals can denote any 4-bit(or 3-bit, ...) binary digit. Thus, using the below embedding algorithm and extracting algorithm, various media data can be embedded and extracted in one PDF file, and as the analysis of the Section 3, the PDF file can remains the property of perceptual transparency.

4.4 Embedding Algorithm

Figure 5 shows the overview of the data embedding steps, and the following is the text description of the embedding algorithm.

Step 1. Create a "015" integer-numeral database (Let it be IND.)which will be embedded in the PDF text. IND includes:

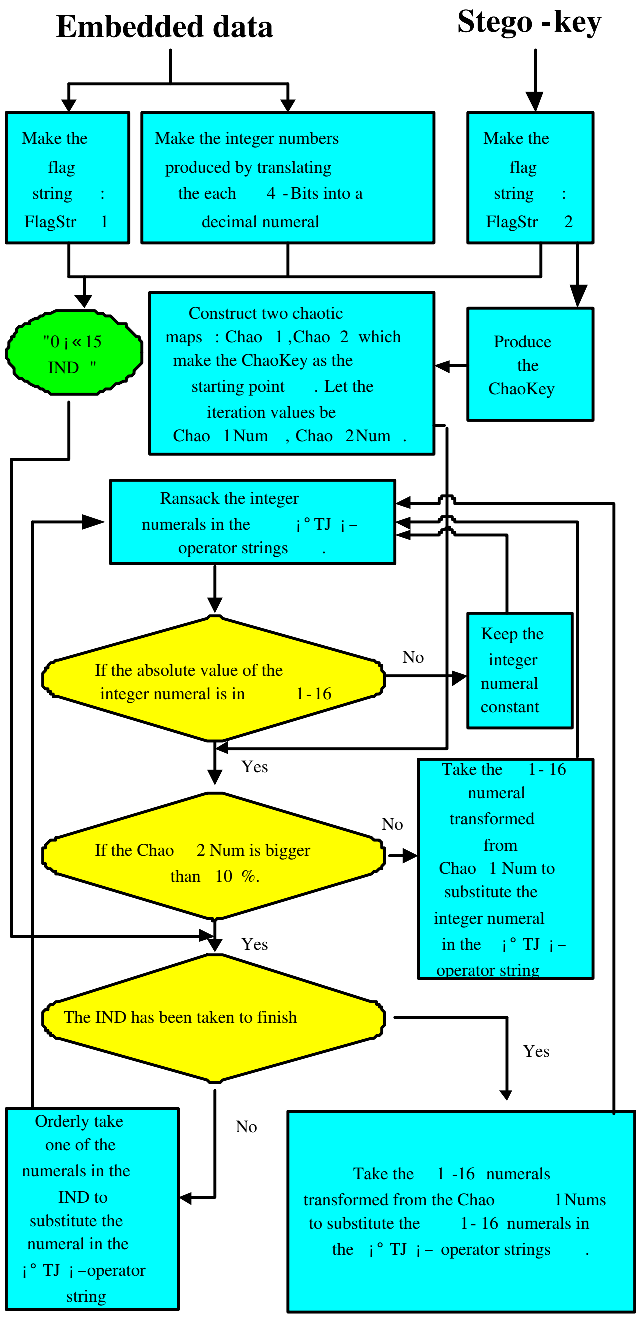

Embedded data
Stego -key
Make the flag string : FlagStr 1
Make the integer numbers produced by translating the each 4 -Bits into a decimal numeral
Make the flag string : FlagStr 2
"0 ¡« 15 IND "
Construct two chaotic maps : Chao 1 ,Chao 2 which make the ChaoKey as the starting point . Let the iteration values be Chao 1Num , Chao 2Num .
Produce the ChaoKey
Ransack the integer numerals in the ¡° TJ ¡– operator strings .
If the absolute value of the integer numeral is in 1 - 16
No
Keep the integer numeral constant
Yes
If the Chao 2 Num is bigger than 10 %.
No
Take the 1 - 16 numeral transformed from Chao 1 Num to substitute the integer numeral in the ¡° TJ ¡– operator string
Yes
The IND has been taken to finish
Yes
No
Orderly take one of the numerals in the IND to substitute the numeral in the ¡° TJ ¡–operator string
Take the 1 -16 numerals transformed from the Chao 1Nums to substitute the 1 - 16 numerals in the ¡° TJ ¡– operator strings .

Figure 5: An overview of the data embedding steps

1) Twenty "0∼15" integer numerals (Formed to be a flag string, and denoted as FlagStr1), which are produced by the following steps:
   - Produce a 160-bit message digest by SHA-1 algorithm with the input of the embedded data.
   - Transform each 8-Bit data from the 160-bit message digest into a ASCII code, and take the ASCII code as the Mod(16) function's input.
2) The "0∼15" integer numerals produced by transforming each 4-Bit embedded datum into a decimal numeral;
3) Twenty "0∼15" integer numerals (Formed to be a flag string, and denoted as FlagStr2), which are produced by the following steps:
   - Produce a 160-bit message digest by SHA-1 algorithm with the input of the stego-key.
   - Transform each 8-Bit data from the 160-bit message digest into a ASCII code, and take the ASCII code as the Mod(16) function's input.

Step 2. Generate a real number by regarding zero as the whole number and FlagStr2 as the decimal fraction. Let the real number be ChaoKey.

Step 3. Construct two Logistic chaotic maps: Chao1 and Chao2 which make ChaoKey as the starting point. Let the iteration values of the two chaotic maps be Chao1Num and Chao2Num.

Step 4. Scan PDF text and pick out the integer numerals in the "TJ" operator strings one by one. If the absolute value of the current integer numeral, denoted as i, is in the span of [1, 16], and the current Chao2Num is bigger than the parameter of redundancy ϕ (for example, $\phi = 10\%$), take out one integer numerals of the integer-numeral database (IND) in sequence, denoted as j ($j \in [0, 15]$), increase j by 1 and get a numeral $j+1(j+1 \in [1, 16])$, and substitute i by j+1. Otherwise, if the Chao2Num is smaller than the parameter of redundancy (for example, 10%), then take one of the "0∼15" numerals transformed from Chao1Num, denoted as h ($h \in [0, 15]$), increase it by 1 and get a numeral $h + 1(h + 1 \in [1, 16])$, and substitute i by $h + 1$; If the absolute value of the current integer numeral i is bigger than 16, then keep i unchanged. If IND is not empty but we have finished scanning the PDF text, then we come to the conclusion that the PDF text is not big enough to embed the embedding data, so return false.

Step 5. Create the iteration values of Chao1Num and Chao2Num. If the IND is empty now, then substitute the "1∼16" integer numerals in the "TJ" operator strings by the "1∼16" numerals transformed from

Chao1Nums and unchanged the integer numerals bigger than 16 until finishing scanning the PDF text. Otherwise, the IND is not empty, then goto **Step 4**.

Step 6. Obtain the stego-texts (PDF files) which is transformed from the cover-texts (PDF files) that have already been embedded the embedding data.

4.5 Extracting Algorithm

Figure 6 shows the overview of the embedded data extracting steps, and the following is the text description of the extracting algorithm.

Step 1. Twenty "015" integer numerals, which produced by the following steps:

- Produce a 160-bit message digest by SHA-1 algorithm with the input of the derived-key.
- Transform each 8-Bit data from the 160-bit message digest into a ASCII code, and take the ASCII code as the Mod(16) function's input.

In succession, increasing the "015" integer numerals by 1, and get the "116" numerals. Let the "116" integer numerals be the flag string, denoted as FlagStr.

Step 2. Generate a real number through regarding zero as the whole number and FlagStr as the decimal fraction. Let the real number be ChaoKey.

Step 3. Construct a Logistic chaotic map: Chao2, which make ChaoKey as the starting point. Let the iteration values of the chaotic map be Chao2Num.

Step 4. Scan PDF text and pick out the integer numerals in the "TJ" operator strings one by one. If the absolute value of the integer numeral picked out is in $[1, 16]$, and the Chao2Num is bigger than the parameter of redundancy (for example, 10%), then take the "116" integer numeral.

Step 5.1. Regard the beginning twenty "116" integer numerals as the integrity flag string, and let it be CheckStr.

Step 5.2. Regard the "116" integer numerals between the twenty-first numeral and the reciprocal twenty-first numeral (included) as the "116" integer numerals figure of the embedded data, turn the extracted "116" integer numerals into general figure, and write to a file. Let the file be EmbeddedFile.

Step 6. Compare the twenty numerals in CheckStr with the twenty "116" integer numerals generated by SHA-1 algorithm and Mod(16) function using the EmbeddedFile as the input(Using the same steps as the **Step 1.**). If they are completely the same, then the embedded data have already been extracted successfully. Otherwise, report error that the extracted embedded data may have been tampered and it cannot be used.

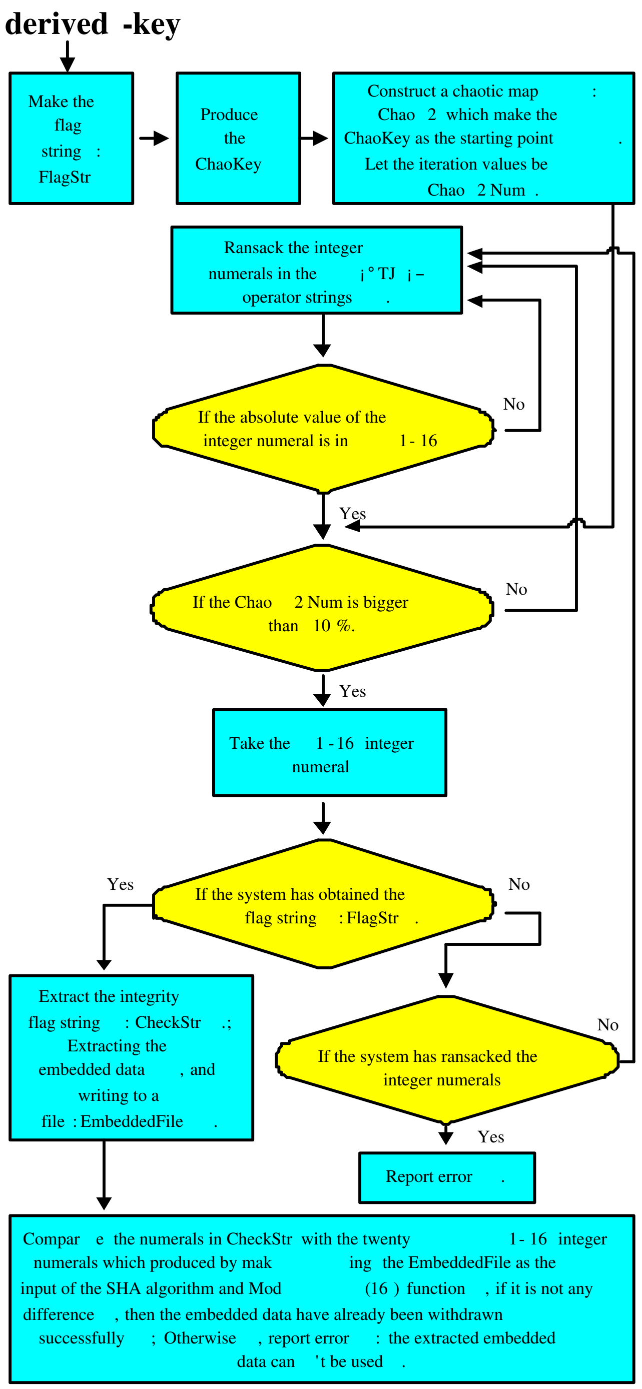

derived -key
Make the flag string : FlagStr
Produce the ChaoKey
Construct a chaotic map : Chao 2 which make the ChaoKey as the starting point . Let the iteration values be Chao 2 Num .
Ransack the integer numerals in the ¡°TJ ¡- operator strings .
If the absolute value of the integer numeral is in 1 - 16
No
Yes
If the Chao 2 Num is bigger than 10 %.
No
Yes
Take the 1 - 16 integer numeral
If the system has obtained the flag string : FlagStr .
Yes
No
Extract the integrity flag string : CheckStr .; Extracting the embedded data , and writing to a file : EmbeddedFile .
If the system has ransacked the integer numerals
No
Yes
Report error .
Compare the numerals in CheckStr with the twenty 1 - 16 integer numerals which produced by making the EmbeddedFile as the input of the SHA algorithm and Mod (16) function , if it is not any difference , then the embedded data have already been withdrawn successfully ; Otherwise , report error : the extracted embedded data can 't be used .

Figure 6: An overview of the embedded data extracting steps

4.6 Estimation of Embedding Capacity

Let *cm* denote the amount of character in a PDF text. We assume that there are *sk*% spacing and kerning pairs which have been defined for the font in the PDF text (we assume that all the characters in the PDF text are in the same font). We also assume there are *se*% integer numerals whose absolute values are in the span of $[1, 16]$. In addition, we define that the parameter of redundancy is *pr*%. Then, we estimate the embedded-data capacity as follows:

$$Capacity = ((cm - cm \times sk\%) \times se\%) \times (1 - pr\%) \qquad (2)$$

Let the embedded data's size be *ed*. If $Capacity > (\frac{eq \times 8}{4} + 40)$, then the embedded data can be embedded into the channel. On the other hand, if we assume there are *se*% integer numerals whose absolute values are in "18", and if $Capacity > (fraceq \times 83{+}40)$, the embedded data can be embedded into the channel. We define the rate of capacity as follows:

$$\frac{ed}{em} \times 100\% \qquad (3)$$

From Figure 2, we can easily find the spacing and kerning pairs (for instance, ¡0A04¿, ¡0F06¿, ¡0A14¿ and ¡0F0D¿, etc.) which have been defined for the font in the PDF text. Anywhere from 50 to 1000 or more kerning pairs may be defined for any one font. A handful of the thousands of possible kerning pairs: Ay, AW, KO, wa [2]. Here, *cm*, *sk*% and *se*% are different in different PDF texts, and *pr*% is defined by us according to the required embedding capacity, undetectability and perceptual transparency.

Furthermore, we can simply regard all the integer values in secret channel (described in Section 3) as the embedding units rather than just those in the 1-16 range, and use the low 4 bits (or 2 bits, or 3 bits) of the "selected" integer values to embed data according to the above approach. This would increase embedding capacity, or make the changes of stego-text even less visible.

4.7 Security Analysis

4.7.1 Obeying the Kerckhoffs Principle

In this paper,we only select some of the integer numerals in the "TJ" operator string to hide data. The selecting of integer numerals depends on the key, and the security of the embedding and extracting algorithms obeys the Kerckhoffs principle.

4.7.2 The Strategy of Using Some Redundancy to Complement Security

The strategy of using some redundancy to complement security is applied in this paper. Figure 7 shows the relation of the three kinds of numerals. The parameter of redundancy is defined by us. Of course, the strategy can be used to enhance the security of the Steganographic System.

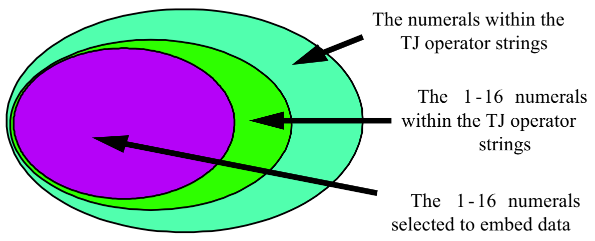

The numerals within the TJ operator strings
The 1-16 numerals within the TJ operator strings
The 1-16 numerals selected to embed data

Figure 7: The relation of the three kinds of numerals

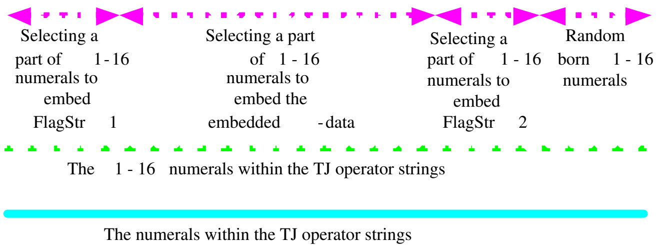

Selecting a part of 1-16 numerals to embed FlagStr 1
Selecting a part of 1-16 numerals to embed the embedded -data
Selecting a part of 1-16 numerals to embed FlagStr 2
Random born 1-16 numerals
The 1-16 numerals within the TJ operator strings
The numerals within the TJ operator strings

Figure 8: The order of embedding data

4.7.3 The Assurance Mechanism of the Data Integrity and Blind Extracting

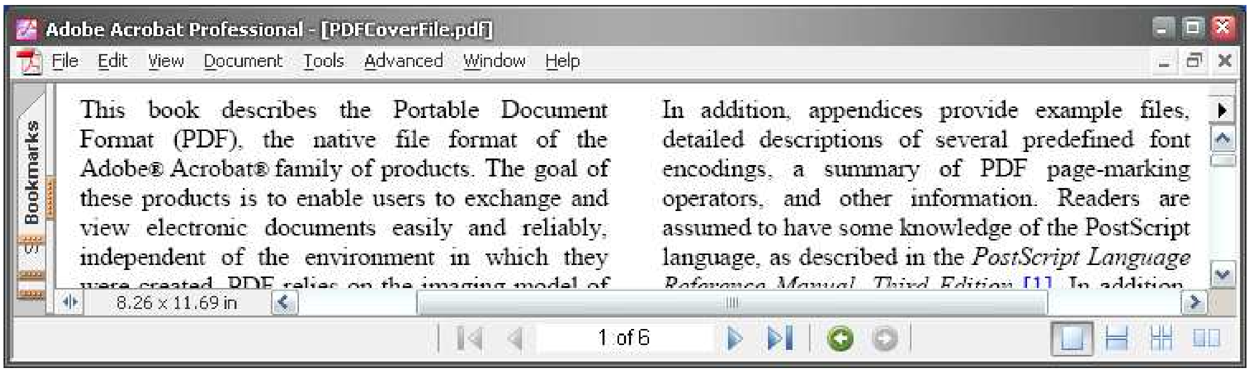
From the above embedding and extracting algorithms, we know that the "015" integer-numeral database (IND) includes FlagStr1, the integer numerals translated from the embedded data and FlagStr2. The order of embedding also depends on the constituting sequence of the IND.

Figure 8 shows the order of embedding data. Through this way, it not only makes the system support blind extracting, but also guarantees the embedded-data's integrity, thus it raises the system's practicability.

4.7.4 Possible Attacks

Steganography mainly considers passive attacks, and the discussion of active attacks is most common for watermarking systems [5]. A passive attacker is only able to analyze the data he could intercept. An active attacker is allowed to modify the data. In this paper, we only discuss the passive attacks.

The steganographic system can prevent stego-only-attack and emb-stego-attack (Emb means the embedded data) through the above security strategies. Because the integer numbers in the kind of PDF files have enough randomness to supply us with a secret channel, and hiding data in a PDF file will not increase its size, the PDFStego can prevent general statistical attacks. For cover-stego-attack and cover-emb-stego-attack, we can use a set of possible cover-texts (PDF files) to make the attackers un-


Adobe Acrobat Professional - [PDFCoverFile.pdf]
File Edit View Document Tools Advanced Window Help
Bookmarks
8.26 x 11.69 in
1 of 6

This book describes the Portable Document Format (PDF), the native file format of the Adobe® Acrobat® family of products. The goal of these products is to enable users to exchange and view electronic documents easily and reliably, independent of the environment in which they were created. PDF relies on the imaging model of

In addition, appendices provide example files, detailed descriptions of several predefined font encodings, a summary of PDF page-marking operators, and other information. Readers are assumed to have some knowledge of the PostScript language, as described in the *PostScript Language Reference Manual, Third Edition* [1]. In addition,

Figure 9: An overview of a cover-text

able to decide which of the possible cover-texts were really used for hiding.

5 Simulation Results

5.1 Perceptual Transparency Property of PDFStego

Figure 9 is an English cover-text (a PDF file) created from an MS Word document which makes the text justified to occupy the full line width by Jaws PDF Creator [7]. The cover-text has six pages with 24026 characters. Figure 10 is a Lena picture with the size of 4668 Bytes. Figure 11 is the stego-text which has embedded the Lena picture.

Intuitively, the stego-text remains the property of perceptual transparency.

5.2 Embedding Capacity Evaluation

In the example of Subection 5.1, the cover-text has the following properties: $cm = 24026$, $sk \approx 10\%$, and $se \approx 60\%$. We define that the pr is equal to 10%, and we know ed is equal to 4668. Then, according to Equation (1) and (**??**), we have:

$$\begin{aligned} Capacity &= ((cm - cm \times sk\%) \times se\%) \times (1 - pr\%) \\ &= ((24026 - 24026 \times 10\%) \times 60\%) \\ &\times (1 - 10\%) \\ &\approx 11676\ (Units) \end{aligned}$$

Of course,

$$\begin{aligned} Capacity &\approx 11676(Units) = 46704(Bits) \\ &> (\frac{ed \times 8}{4} + 40) = 9376(Units) \\ &= 37504(Bits) \end{aligned}$$

the Lena picture with the size of 4668 Bytes can be embedded into the cover-text, and the rate of capacity is higher than

$$\frac{ed}{cm} \times 100\% = \frac{4668}{24026} \times 100\% \approx 19.43\%$$

Figure 10: The Lena picture with the size of 4668 bytes

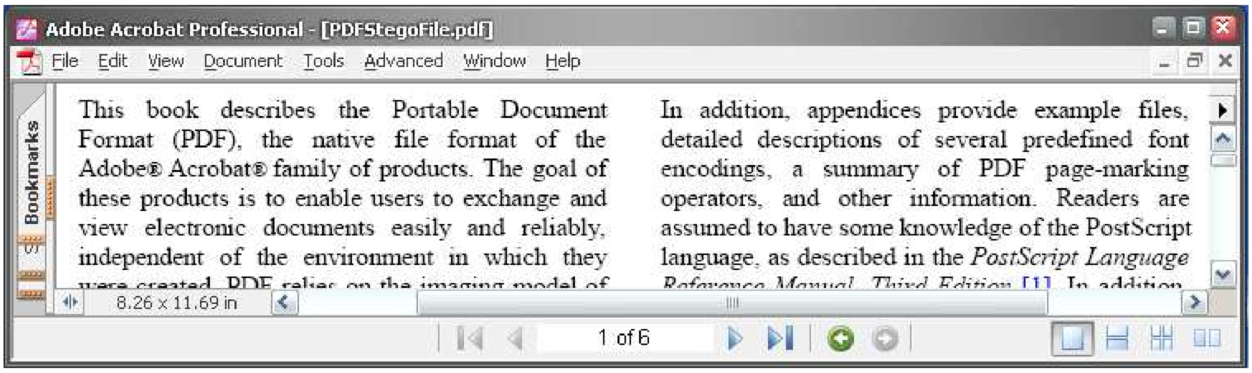
5.3 Performance Analysis and Comparison

Taking the cover-text (with 24026 characters, provided by Subection 5.1) for example, when we use the approach of wbStego4 [16], we find that the embedding capacity is very small, only 107 Bytes, and hiding data in a PDF file will increase its filesize. Moreover, when reading the FAQs of wbStego, we find that there are no general rules for the assessment of the amount of data (that) a PDF file can embed, and the size of a PDF file will increase when hiding data in it.

On the other hand, Comparing with the previous steganographic algorithms of structure-texts in which an embedding unit represents a single bit, the embedding units of PDF texts are integer numerals. Thus, we can gain much greater capacity to embed data. The previous steganographic methods which are proposed by varying characters spacing can embed data with the size of 24026 Bits at most. If we use the methods, the Lena picture with the size of 4668 Bytes(37344 Bits) can not be embedded into a structure-text. Of course, the previous steganographic methods [3, 4, 8, 9, 10, 11, 12] which are proposed by varying lines or words spacing or by varying certain character features slightly have smaller embedding capacities than the methods which are proposed by varying characters' spacing.

As we know, embedding capacity and the general rules for the assessment of the amount of data that a cover-text can embed are the two important factors of practicability. If the size of a cover-text will increase when hiding data in it, the steganographic system can not prevent general statistical attacks.

Table 1 shows the results of comparative performance


Adobe Acrobat Professional - [PDFStegoFile.pdf]
File Edit View Document Tools Advanced Window Help
Bookmarks
8.26 x 11.69 in
1 of 6

This book describes the Portable Document Format (PDF), the native file format of the Adobe® Acrobat® family of products. The goal of these products is to enable users to exchange and view electronic documents easily and reliably, independent of the environment in which they were created. PDF relies on the imaging model of

In addition, appendices provide example files, detailed descriptions of several predefined font encodings, a summary of PDF page-marking operators, and other information. Readers are assumed to have some knowledge of the PostScript language, as described in the *PostScript Language Reference Manual, Third Edition* [1]. In addition,

Figure 11: An overview of the stego-text which has embedded the Lena picture

analysis. Because the property of perceptual transparency can not be measured well by actual methods, in Table 1, we use "No Optically Changed" to denote perceptual transparency property. The embedding capacity of PDFStego and wbStego4 are gained by taking the cover-text (with 24026 characters, provided by Subection 5.1) for example. From Table 1, we can easily find the good performances of PDFStego.

In addition, for all this kind of PDF files, no matter they are edited differently, e.g. single/double column, or contain any combination of text, graphics, and images, PDFStego has good performances.

6 Conclusions and Future Work

We have presented a novel steganographic technique for hiding data in a kind of PDF texts. The steganographic system PDFStego has been designed, analyzed and implemented. Theoretic analysis and the computing results show that the steganographic system is secure, the embedding capacity is high, and the algorithm is practical.

We are currently doing more research on the steganographic technique's resistance against various attacks. How to evaluate the security of this steganographic system is also a challenge in our future work.

References

[1] Adobe Systems Incorporated, *Portable document format reference manual*, Version 1.3, http://www.adobe.com. March,1999.

[2] J. H. Bear, *Desktop Publishing-kerning and Tracking*, http://desktoppub.about.com/, 2005.

[3] J. T. Brassil, et al., "Electronic marking and identification techniques to discourage document copying," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1495-1504, 1995.

[4] J. Brassil, N. F. Maxemchuk, and L. O. Gorman, "Electronic marking and identification techniques to discourage document coping," in *Proceedings of INFORCOM'94*, pp. 1278-1287, 1994.

[5] E. Franz and A. Pfitzmann, "Steganography secure against Cover-Stego-Attacks," *3th nternational Workshop, Information Hiding 1999*, LNCS 1768, pp. 29-46, 2000.

[6] B. L. Hao, *Starting with Parabolas- An Introduction to Chaotic Dynamics*, Shanghai Scientific and Technological Education Publishing House, pp. 10-12, Shanghai, China, 1993.

[7] *Global graphics software ltd..jaws PDF creator*, http://www.jawspdf.com/pdf_creator/releases.html, 2005.

[8] S. H. Low and N. F. Maxemchuk, "Performance comparision of two text marking methods," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 4, pp. 561-572, 1998.

[9] S. H. Low, et al. "Document marking an identification using both line and word shifting," in *Proceedings INFOCOM'95*, pp. 853-860, Boston, MA, Apr. 1995.

[10] S. H. Low , N. F. Maxemchuk, and A. M. Lapone, "Document identification for copyright protection using centroid detection," *IEEE Transactions on Communications*, vol. 46, no. 3, pp. 372-383, 1998.

[11] N. F. Maxemchuk and S. H. Low, "Marking text documents," in *Proceedings International Conference Image Processing*, pp. 13-17, Santa Barbara, CA., Oct. 1997.

[12] T. May, *Cyphernomicon 14.7*, http://www.cyphernet.org/cyphernomicon /chapter14/14.7.html, 2005.

[13] H. Noda, et al, "Bit-plane decomposition steganograohy combined with JPEG2000 compression," *5th International Workshop, Information Hiding 2002*, Noordwijkerhout, The Netherlands, LNCS 2578 , Springer-Verlag, pp. 295-309, Oct. 2002.

[14] F. A. P. Petitcolas, R. J. Anderson and M. G. Kuhn, "Information hiding-A survey," in *Proceedings of the IEEE, special issue on protection of multimedia content*, vol. 87, no. 7, pp. 1062-1078, July 1999.

[15] W. Stallings, *Cryptography and network security: Principles and practice*, second edition, Prentice-Hall, Inc. 2002.

Table 1: The results of comparative performances analysis

Performances	PDFStego	wbStego4	The methods which are proposed by varying characters spacing
Perceptual Transparency	No Optically Changed	No Optically Changed	No Optically Changed
Embedding Capacity	About 46704 Bits	856 Bits	24026 Bits at most
Whether the Embedding Capacity can be Defined?	Yes	No	Yes
Whether the size of a cover-text will increase when hiding data in it?	No	Yes	No

[16] wbStego Studio, *The steganography tool wbStego4* http://www.wbailer.com/wbstego. 2005.

Shangping zhong received his B.S. in Mathematics Science from Fuzhou University, Fuzhou, China, in 1991, and his M.S. in Computer Science and Technology from Fuzhou University, Fuzhou, China, in 1997, and his Ph.D. in Computer Science and Technology from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2005. His current research interests include information security and mobile communications.

Xueqi Cheng received his M.S. in Computer Science and Technology from Northeastern University, Shenyang, China, in 1996. Currently, he is a researcher in Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His current research interests include information security, Information Retrieval and Service Grid.

Tierui Chen received her B.S. in Computer Science and Technology from Northeastern University in 2003. Currently, she is currently pursuing master degree in the Institute of Computing Technology, Chinese Academy of Sciences. Her research interests are information security and digital rights management.