

DATA HIDING IN PDF FILES AND APPLICATIONS BY IMPERCEIVABLE MODIFICATIONS OF PDF OBJECT PARAMETERS

¹*Jiun-Tsung Wang* (王竣聰) and ²*Wen-Hsiang Tsai* (蔡文祥)

¹Institute of Multimedia Eng., National Chiao Tung University, Hsinchu, Taiwan

²Dept. of Computer Science and Engineering, National Chiao Tung University, Hsinchu, Taiwan
E-mail: jtwang@cs.nctu.edu.tw, whtsai@cs.nctu.edu.tw

ABSTRACT

The PDF file is very popular nowadays, and so a good choice as cover media for covert communication. A data hiding method for this purpose is proposed, which hides secret data by slight modifications of the values of various PDF object parameters, yielding a difference of appearance very difficult to notice by human vision. Because digital files are easy to duplicate and modified, a method for authentication of PDF files by the proposed data hiding technique is also proposed. Secret keys are used to enhance the security of the embedded data. Experimental results show the feasibility of the proposed methods.

1. INTRODUCTION

Because of the rapid development of broadband networks and personal computers, transmissions of digital documents and images through the Internet become easier and easier. Some properties of the PDF file, such as high-quality printing and cross-platform applicability, promote the popularity of the PDF file. Furthermore, PDF files can be transmitted on the Internet quickly, so they become good examples of cover media to carry secret information. Because their file sizes usually are large and they are rich-text documents, we can design methods to embed data into PDF files. For this application, we also need a special decoding scheme to extract the secret information embedded in the PDF files. In this study, it is desired to design a covert communication method via PDF files. On the other hand, since PDF files may be modified illegally, it is also desired to design a method for authenticating PDF files whenever necessary.

Zhong and Chen [2] proposed an information steganographic algorithm based on the PDF file, which hides secret data by embedding them in the space between the objects in a PDF file. Zhong, et al. [3] proposed another covert communication method for PDF files, which hides secret messages by tuning the distances between the texts in a PDF file. Liu and Tsai [4] proposed an active quotation authentication method

for Microsoft Word documents by hiding the block signature in the document.

In this paper, we propose new data hiding techniques for PDF files by using different objects in the PDF. Two applications of the proposed techniques, namely, covert communication via PDF files and authentication of PDF files, are also investigated. Algorithms for them are proposed. Experimental results show the feasibility of the proposed techniques. In the remainder of this paper, we give first a general introduction to the PDF in Section 2. In Section 3, we describe the first proposed technique of hiding data in the page parameter. And in Section 4, we describe another technique of hiding data in text matrices. These two techniques may be used for covert communication directly. In Section 5, we propose a PDF file authentication method. Some experimental results are shown in Section 6, followed by a conclusion and some suggestions for future research.

2. INTRODUCTION TO PDF

The PDF was designed as a cross platform for document exchange by Adobe Systems [1]. It is a mixture of text and binary formats, and its content is described by a page description language which is modified from PostScript[®]. There are three parts generally in a PDF file, and the first part is the *magic header* which is constructed by a comment and it is optional. The second part is the set of PDF objects which are used to describe the content and the appearance of the PDF file. The final part is the *cross reference table* and the *trailer* of a document which are used to locate the entry point and the objects of a PDF file.

According to the cross reference table and the trailer, a PDF parser can find the entry point of a PDF file and build a hierarchical tree. Basic components in a PDF file are called *objects*, for example, pages, contents, images, fonts, and numbers. The most interesting object is *content object*, in which there is much syntax, such as *text matrix*, *text-showing operator*, *coordinate transformation matrix*, *image-showing operator*, and

font-specifying operator. A matrix in a content object is used to describe the scale, orientation, and translation parameters of a destination object. A text matrix is composed by six distinct numbers a, b, c, d, e, f and a “Tm” operator. The structure of a text matrix is illustrated in Figure 1. Numbers $a, b, c,$ and d are elements of a rotation matrix which are used to describe the scale and orientation of a text object, and the numbers e and f are translation parameters which are used to describe the position of a text. In this paper, we propose a method to embed secret data in such *numeric objects* of a PDF file.

$$T_m = T_{LM} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

Figure 1 The structure of a text matrix.

3. HIDING DATA IN PARAMTERS OF PAGES

3.1 Concept of Proposed Method

According to the PDF standard, there are eight basic types of objects in PDF files. One of them is the page object, which includes parameters to describe the properties of a page in a PDF file. For example, the visible area of a page, named *media box*, is described by four numbers, with the first two numbers specifying the position of the left upper corner, and the last two numbers specifying the position of the right lower corner, both in terms of coordinates. These four numbers decide the width and height of the visible area of the page. A page object which includes a media box is illustrated in Figure 2. Small changes on the page width and height are not easy to be noticed, and so we may embed secret message data in the media boxes of pages.

```

11 0 obj
<</Type/Page
/Parent 1 0 R
/Resources 14 0 R
/MediaBox[0 0 595.2 841.68]
/Contents[12 0 R ]
>>
Endobj

```

Figure 2 An example of a page object.

Another basic type of objects is *number*, including integer and real numbers. Integers and real numbers are exchangeable in PDF files under some conditions, and there is no difference between 0 and 0.0. We may so embed secret message data in a PDF file by modification of such numbers. Modification of them may be categorized into two forms:

Type 1: the original number is an integer and the result is a real number.

Type 2: the original number is a real number and the result is still a real number.

The former type can be divided further into two types:

Type 1.1: the modification makes a small change of the magnitude of the number, for example, modification of 20 to be 20.00457;

Type 1.2: the modification makes no change of the magnitude of the number, for example, modification of 20 to be 20.00000.

The later type can be divided further as well into two types:

Type 2.1: the modification makes a small change of the magnitude of the number, for example, modification of 854.7 to be 854.700457;

Type 2.2: the modification makes no change of the magnitude of the number, for example, modification of 854.7 to be 854.700000.

We may design data hiding schemes with the above four types of modifications. That is, we may embed data in the *number object*. With Types 1.1 and 2.1, small changes can be used to embed multiple digits into a digit, and there is no limitation on the length of embedded digits. In the above example for Type 1.1, we embed three digits 4, 5, and 7 into the integer 20 after appending a separator of a pre-selected digit sequence, 00, to the integer. The use of such a kind of separator is necessary in the message extraction process (described later) to distinguish the digits of the original data from the embedded ones. Of course, such a separator must be selected to be unique to cause no ambiguity in digit decoding in the extraction process. Similarly, in the example for Type 2.1 above, we embed the secret digits 457 into 854.7 as its tail digits following the separator 00.

By Types 1.2 and 2.2 of number value modifications, we can embed multiple 0’s into an integer or a real number. However, to use these 0’s as message digits, we need further an additional message encoding scheme to transform the sequence of 0’s into the message digits. Obviously, we have to use unary coding for this purpose since only a symbol, namely, 0, can be used here for data encoding. Therefore, to encode the digits 13, for instance, we have to append a sequence of 13 0’s to the end of the original data. Note that no separator is required here. Though this scheme of unary coding is feasible in our application of covert communication, it will generate too many 0’s and so increase the size of the resulting stego-PDF file. Consequently, we do not use it in our experiments.

3.2 Proposed Data Hiding Process

In the proposed data hiding process, we apply exclusive-OR operations to every byte in a secret message and every corresponding byte in a user key

before hiding the message in a PDF file. After that, the data are put into three-bit groups. If the number of bits is not divisible by 3, then we pad 1 or 2 zeros after the last group. We then map each 3-bit group to a decimal digit by a coding table, for example, Table 1, and then append it to the end of the original parameters in the media box being processed with a separator “00” between them. A corresponding detailed algorithm is described in the following.

Algorithm 1: *encoding a message and hiding it in a media box*

Input: a user key K , a secret message S , and a media box $B = (b_1, b_2, b_3, b_4)$.

Output: a media box B' with data embedded F' .

Steps:

1. For every byte in S , apply exclusive-OR operations to the i -th byte of K and the i -th byte of S to generate the i -th byte of a new sequence of bytes S' with the same length as that of S .
2. Divide S' into 3-bit groups f_1, f_2, \dots, f_k .
3. Transform f_1 through f_k into decimal numbers (digits) n_1, n_2, \dots, n_k by Table 1.
4. Concatenate n_1, n_2, \dots, n_k in order as a digit string and divide it into four parts N_1, N_2, N_3 , and N_4 .
5. Embed the data into B in the following way:
for $i = 0, 1, 2, 3$,
5.1 if the number in b_i is an integer, then append to it the number sequence of “00” (a dot and two zeros) followed by N_i to yield b_i' ;
5.2 otherwise, append to it “00” (two zeros) followed by N_i to yield b_i' .
6. Take $B' = (b_1', b_2', b_3', b_4')$ as the desired output.

Table 1 Mapping of groups of three bits to a digit.

Bit stream	digit	Bit stream	digit
000	1	100	5
001	2	101	6
010	3	110	7
011	4	111	8

3.3 Proposed Data Extraction Process

The extraction of a secret message from a media box in a PDF file is conducted in the following way. First, we scan the value of each parameter of the media box from the last digit of it to the dot, if existent. If two consecutive 0's are obtained, we cut off the digits after the two 0's. We then concatenate all the cut digits to get a string of digits, map them into 3-bits groups by Table 2, and concatenate the resulting groups into a bit stream. After that, we apply exclusive-OR operations to the bit stream and the user key, and regard the resulting bits as ASCII codes to get the hidden secret message finally. A detailed algorithm is described in the following.

Algorithm 2: *Extraction of a message from a media box*

Input: a user key K , and a media box $F = (x_1, y_1, x_2, y_2)$.

Output: a secret message S .

Steps:

1. For $i = 1$ and 2,
1.1 extract the digits after “00” in x_i , and save them in N_{2i-1} ; and
1.2 extract the digits after “00” in y_i , and save them in N_{2i} .
2. Concatenate N_1, N_2, N_3 , and N_4 into a digit string N and transform N into a bit stream by Table 2.
3. Treat the bit stream as a binary stream of ASCII codes, and transform them to a string A of ASCII codes.
4. Truncate the user key K or pad numbers to it in the following way:
4.1 if K is longer than the string A , truncate K to be a string whose length is equal to that of A ;
4.2 if K is shorter than A , pad to K the required number of bytes copied from the beginning ones of K .
5. For every byte in A , apply exclusive-OR operations to the i -th byte of A and the corresponding byte of K to generate the i -th byte of S , where S is a sequence of bytes with the same length as A .

Table 2 Mapping a digit to 3 bits.

Digit	Bit stream	Digit	Bit stream
1	000	5	100
2	001	6	101
3	010	7	110
4	011	8	111

4. HIDING DATA IN TEXT MATRICES

A content object in a PDF file contains many text objects and text matrices which are used to describe the orientation, scale, and position of the text in the PDF file. The default resolution is 72 dpi, and small changes of text matrices in a content object will not affect the appearance of it obviously. Therefore, we may embed data in text objects of PDF files as well.

4.1 Concept of Proposed Method

The contents of each PDF page are composed of some text objects. The detail information of a text object which includes the size, rotation, and position should be specified before the text showing operators are applied. See Figure 3 for an instance. Such text object information is described by a text matrix which is composed by two parts: the text-orientation and the text position. The essential encoding scheme is the same as the previously-described method for embedding data in a media box, but since there are much more text

matrices than media boxes, we change some steps of the encoding steps in the previously-described data hiding method to adapt it for the purpose of data embedding in text matrices.

```

<< /Length 59 >>
stream
/GS1 gs
BT
/TT2 1 Tf
21 0 0 21 90 150 Tm
(I am a boy)Tj
ET
endstream

```

Figure 3 An example of a content object of a PDF page.

4.2 Proposed Data Hiding Process Using Text Matrices

Considering the large number of text matrices, we embed secret digits in text matrices *uniformly*. Also, the basic structure of the media box is different from that of the text matrix, so some steps in Algorithm 1 should be refined, as described in the following.

Algorithm 3: *encoding a message and hiding it in text matrices*

Input: a user key K , a secret message S , and text matrices $F_1 = (a_1, b_1, c_1, d_1, x_1, y_1)$, $F_2 = (a_2, b_2, c_2, d_2, x_2, y_2)$, ..., $F_L = (a_L, b_L, c_L, d_L, x_L, y_L)$.

Output: text matrices F_1' , F_2' , ..., F_L' with data embedded.

Steps:

1. For every byte in S , apply exclusive-OR operations to the i -th byte of K and the i -th byte of S to generate the i -th byte of a new sequence of bytes, S' , with the same length as that of S .
2. Divide each 3 bits of S' into groups of bitstream f_1, f_2, \dots, f_k .
3. Transform f_1 through f_k into decimal numbers (digits) n_1, n_2, \dots, n_k by Table 1.
4. Concatenate n_1, n_2, \dots, n_k as a digit string N and divide it into $2L$ parts N_1, N_2, \dots, N_{2L} .
5. Embed the data into text matrices in the following way.
 - For $i = 0$ to L ,
 - 5.1 if the number in the parameter x_i of the text matrix F_i is an integer, then append to it the number sequence of “.00” (a dot and two zeros) followed by N_{2i} . Otherwise, append to it “.00” (two zeros) followed by N_{2i} .
 - 5.2 if the number in the parameter y_i of the text matrix F_i is an integer, then append to it the number sequence of “.00” (a dot and two

zeros) followed by N_{2i+1} . Otherwise, append to it “.00” (two zeros) followed by N_{2i+1} .

4.3 Proposed Data Extraction Process

Because we hide data in the text matrices uniformly, we should scan all the text matrices in a PDF file to extract the secret data. Some steps in Algorithm 2 should be tuned accordingly as described in the following.

Algorithm 4: *extracting a message from text matrices*

Input: a user key K and text matrices F_1, F_2, \dots, F_L .

Output: a secret message S .

Steps:

1. For $i = 1, 2, \dots, L$,
 - 1.1 extract the x and y coordinates from the text matrix F_i ;
 - 1.2 extract the digits after “.00” in x_i , and save them in N_{2i-1} ;
 - 1.3 extract the digits after “.00” in y_i , and save them in N_{2i} .
2. Concatenate N_1, N_2, \dots, N_{L+1} into a digit string N , and transform N into a bit stream by Table 2
3. Treat the bit stream as a binary stream of ASCII codes, and transform them into a string A of ASCII codes.
4. Truncate the user key K or pad numbers to it in the following way:
 - 4.1 if K is longer than the string A , truncate K to be a string whose length is equal to that of A ;
 - 4.2 if K is shorter than A , pad to K the required number of bytes copied from the beginning ones of K .
5. For every byte in A , apply exclusive-OR operations to the i -th byte of A and the corresponding byte of K to generate the i -th byte of S , where S is a sequence of bytes with the same length as A .

5. AUTHENTICATION OF PDF FILES FOR FIDELITY AND INTEGRITY VERIFICATION

5.1 Concept of Authentication Method

In the previous sections, we have proposed new techniques for data hiding in PDF files using several kinds of objects in the PDF. They may be used for the purpose of covert communication. In this section, the proposed authentication method which is implemented also by applying the -proposed data hiding techniques will be described.

In order to achieve the goal, we propose to embed an authentication signal in the text matrix of each text object. The authentication signal is generated from the contents of the string in each text block as well as the user key. Verifying the fidelity and integrity of a PDF file to achieve the authentication purpose then is just to extract the authentication signals from the text

matrices and match them with the signals which are generated from the strings in the currently-processed text blocks. Advantages of hiding authentication signals in text matrices are multifold. First, if any text object is moved, the authentication signal in the text matrix will be destroyed. So our method can detect illegal movements of text objects. Second, if the strings in the text objects are modified, the embedded authentication signals and the signals generated from the modified strings will not match. So, our method can also detect illegal text modification. Third, if a PDF file is regenerated by another PDF generator, the authentication signals in the PDF file will be destroyed as well, and so our method can detect regeneration of PDF files as well. Furthermore, we may protect the security of the authentication signals by exclusive-ORing the signals with a user key, as done in our method. This ensures that an illicit user, who does not have the user key, cannot create fake authentication signals to cheat other users.

5.2 Authentication Signal Embedding Process

Before embedding authentication signals, we need to scan all the text blocks and their corresponding text matrices. For each text block, we extract the string in it, sum up the values of the ASCII codes of the characters in the string, and take the modulo-256 value of the sum to get a digest of the string. Then, we apply exclusive-OR operations on the sum and the user key to generate an authentication signal, which is then embedded into the PDF file using the method proposed previously. The details are described in the following.

Algorithm 5: *embedding authentication signals in a PDF file*

Input: a user key $K = (k_1, k_2, \dots, k_N)$ with each k_i being a byte, and a PDF file P .

Output: an authentication signal-embedded PDF file.

Steps:

1. Find all the strings T_i in the PDF file P and their corresponding text matrices X_i . Let the number of text objects be M .
2. For $i = 1, 2, \dots, M$, perform the following steps.
 - 2.1 Sum up all the bytes k_1 through k_N of K and take the modulo-256 value D of the sum. That is, compute
$$D = (k_1 + k_2 + k_3 \dots + k_N) \bmod 256.$$
 - 2.2 Apply exclusive-ORing operations on all the bytes of T_i , where $i = 1, 2, \dots, M$, and take the exclusive-OR value of the result E and D as F . That is, for $T_i = (t_1, t_2, t_3, \dots, t_L)$, compute
$$E = t_1 \oplus t_2 \oplus t_3 \dots \oplus t_L;$$

$$F = D \oplus E.$$
3. Embed F in X_i by Algorithm 3.

5.3 Authentication Signal Extraction Process

After the above procedure are conducted, if the PDF is tampered with, our program can find out where the tampering occurs by verification of the authentication signals which are hidden in the text matrices. The verification, simply speaking, is a reverse version of the above process.

Before extracting the authentication signal, we have to generate the authentication verification signal first. The authentication verification signal is generated by the above procedure which is used to generate authentication signal. Next, we match the authentication signal so computed with the authentication signal extracted from the stego-PDF file. If they are the same, it is decided that the document is an unmodified one; otherwise, the document must have been tampered with. The details are described in the following.

Algorithm 6: *extracting authentication signals from a PDF file*

Input: a user key $K = (k_1, k_2, \dots, k_N)$ where each k_i is a byte, and a PDF file P .

Output: a verification report of P .

Steps:

1. Find all the strings T_i in the PDF file and their corresponding text matrices X_i . Let the number of text objects be M .
2. For $i = 1, 2, \dots, M$, perform the following steps.
 - 2.1 Sum up all the bytes k_1 through k_N of K and take the modulo-256 value D of the sum. That is, compute
$$D = (k_1 + k_2 + k_3 \dots + k_N) \bmod 256.$$
 - 2.2 Apply exclusive-ORing on all the bytes of T_i , where $i = 1, 2, \dots, M$, and take the exclusive-OR value of the result E and D as F . That is, for $T_i = (t_1, t_2, \dots, t_L)$, compute
$$E = t_1 \oplus t_2 \oplus t_3 \dots \oplus t_L;$$

$$F = D \oplus E.$$
 - 2.3 Extract embedded data from X_i by Algorithm 4 as A_i .
 - 2.4 If $A_i \neq F$, then decide that the contents of T_i have been modified and mark it so in P .

6. EXPERIMENTAL RESULTS

In our experiments, we designed a user interface written in the language of Java to implement the proposed message embedding and extracting algorithms. Some results of our experiments are shown in Figures 3 through 9. Figure 4 shows a cover PDF document. Figure 5 shows the corresponding stego-PDF document after embedding a message into the media box of the cover document, and Figure 6 shows the stego-PDF

document after embedding a message into the text matrices of the cover document through the interface as shown in Figure 7. Figure 8 shows correct extraction of the hidden message using a correct key, and Figure 9 shows erroneous extraction of the message with an incorrect key.

We also setup a server which can embed authentication signals and verify the fidelity and integrity of PDF files. Figure 10 shows the screenshot of visiting the server with Mozilla Firefox 2.0. A protected PDF file is shown in Figure 11, a modified version of it is shown in Figure 12. Finally, a verification report is shown in Figure 13.

7. CONCLUSIONS

In this paper, a covert communication method via PDF files as cover media and a method for authentication of the fidelity and integrity of PDF files by data hiding techniques have been proposed. Because we can hide a covert message in PDF files without any side effects on the visual appearance of the files, the secrets in these PDF files are not easy to observe and access illicitly. Even if an illicit user knows that there is a secret message in a 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 authentication method can detect various types of PDF file tampering. The authentication signal is randomized by a user key, so it is not easy to create fake authentication signals. The proposed methods are feasible, as proved by our experiments. Future works may be directed to investigation of using other data structures of the PDF to hide data, and other applications of the proposed data hiding techniques.

ACKNOWLEDGEMENT

This work was supported partially by National Digital Archives Program with Project No. NSC-96-2422-H-009-001 and partially by the NSC under the Project of Advanced Technologies and Applications for Next Generation Information Networks (II) – Sub-project 5: Network Security with Project No. NSC-96-2752-E-009-006-PAE.

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, Jan. 2007.
- [4]. T. Y. Liu and W. H. Tsai, "Active quotation authentication in Microsoft Word documents using

block signatures," *Proceedings of 3rd International Conference on Information Technology: Research and Education (ITRE 2005)*, Hsinchu, Taiwan, June 2005.

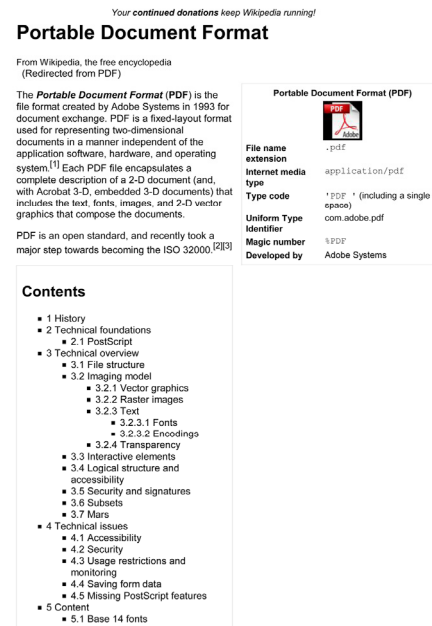


Figure 4 The view of the original PDF file in Adobe Acrobat Reader window.



Figure 5 The view of the stego-PDF file in Adobe Acrobat Reader window.

Your continued donations keep Wikipedia running!

Portable Document Format

From Wikipedia, the free encyclopedia
(Redirected from PDF)

The **Portable Document Format** (PDF) is the file format created by Adobe Systems in 1993 for document exchange. PDF is a fixed-layout format used for representing two-dimensional documents in a manner independent of the application software, hardware, and operating system.^[1] Each PDF file encapsulates a complete description of a 2-D document (and, with Acrobat 3-D, embedded 3-D documents) that includes the text, fonts, images, and 2-D vector graphics that compose the documents.

PDF is an open standard, and recently took a major step towards becoming the ISO 32000.^{[2][3]}

Portable Document Format (PDF)



File name extension: .pdf

Internet media type: application/pdf

Type code: 'PDF' (including a single space)

Uniform Type Identifier: com.adobe.pdf

Magic number: %PDF

Developed by: Adobe Systems

Contents

- 1 History
- 2 Technical foundations
 - 2.1 PostScript
- 3 Technical overview
 - 3.1 File structure
 - 3.2 Imaging model
 - 3.2.1 Vector graphics
 - 3.2.2 Raster images
 - 3.2.3 Text
 - 3.2.3.1 Fonts
 - 3.2.3.2 Encodings
 - 3.2.4 Transparency
 - 3.3 Interactive elements
 - 3.4 Logical structure and accessibility
 - 3.5 Security and signatures
 - 3.6 Subsets
 - 3.7 Mars
- 4 Technical issues
 - 4.1 Accessibility
 - 4.2 Security
 - 4.3 Usage restrictions and monitoring
 - 4.4 Saving form data
 - 4.5 Missing PostScript features
- 5 Content
 - 5.1 Base 14 fonts

Figure 6 The view of the stego-PDF file in Adobe Acrobat Reader window.

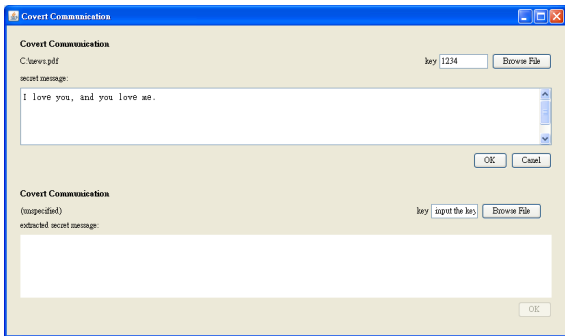


Figure 7 Window of user interface with a secret message and a user key as input.

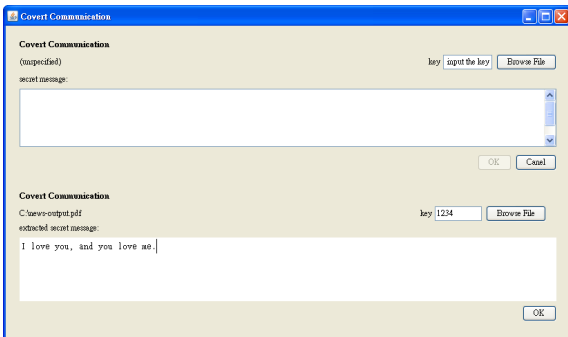


Figure 8 Window of user interface with embedded message extracted.



Figure 9 Window of user interface with a wrong key as input, resulting in erroneous extraction of embedded message.

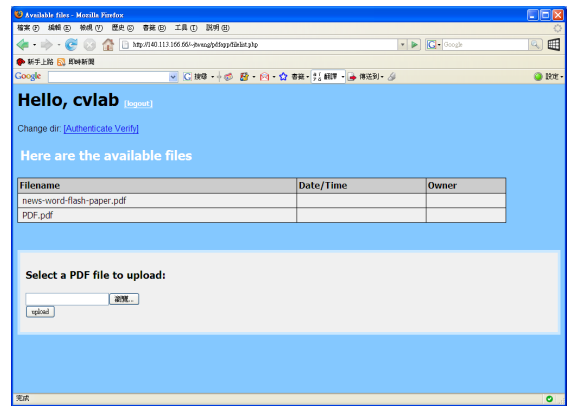


Figure 10 The window of visiting the document management server by Mozilla Firefox 2.0.

Your continued donations keep Wikipedia running!

Portable Document Format

From Wikipedia, the free encyclopedia
(Redirected from PDF)

The **Portable Document Format** (PDF) is the file format created by Adobe Systems in 1993 for document exchange. PDF is a fixed-layout format used for representing two-dimensional documents in a manner independent of the application software, hardware, and operating system.^[1] Each PDF file encapsulates a complete description of a 2-D document (and, with Acrobat 3-D, embedded 3-D documents) that includes the text, fonts, images, and 2-D vector graphics that compose the documents.

PDF is an open standard, and recently took a major step towards becoming the ISO 32000.^{[2][3]}

Portable Document Format (PDF)



File name extension: .pdf

Internet media type: application/pdf

Type code: 'PDF' (including a single space)

Uniform Type Identifier: com.adobe.pdf

Magic number: %PDF

Developed by: Adobe Systems

Contents

- 1 History
- 2 Technical foundations
 - 2.1 PostScript
- 3 Technical overview
 - 3.1 File structure
 - 3.2 Imaging model
 - 3.2.1 Vector graphics
 - 3.2.2 Raster images
 - 3.2.3 Text
 - 3.2.3.1 Fonts
 - 3.2.3.2 Encodings
 - 3.2.4 Transparency
 - 3.3 Interactive elements
 - 3.4 Logical structure and accessibility
 - 3.5 Security and signatures
 - 3.6 Subsets
 - 3.7 Mars
- 4 Technical issues
 - 4.1 Accessibility
 - 4.2 Security
 - 4.3 Usage restrictions and monitoring
 - 4.4 Saving form data
 - 4.5 Missing PostScript features
- 5 Content
 - 5.1 Base 14 fonts

Figure 11 The appearance of the original PDF document with Adobe Acrobat Reader window.


Your continued donations keep Wikipedia running!

Portable Document Format

From Wikipedia, the free encyclopedia
(Redirected from PDF)

The **Portable Document Format** (**PDF**) is the file format created by Adobe Systems in 1993 for document exchange. PDF is a fixed-layout format used for representing two-dimensional documents in a manner independent of the application software, hardware, and operating system.^[1] Each PDF file encapsulates a complete description of a 2-D document (and, with Acrobat 3-D, embedded 3-D documents) that includes the text, fonts, images, and 2-D vector graphics that compose the documents.

PDF is an open standard, and recently took a major step towards becoming the ISO 32000.^{[2][3]}

Portable Document Format (PDF)	
	
File name extension	.pdf
Internet media type	application/pdf
Type code	*PDF* (including a single space)
Uniform Type Identifier	com.adobe.pdf
Magic number	%PDF
Developed by	Adobe Systems

Contents

- 1 History
- 2 Technical foundations
 - 2.1 PostScript
- 3 Technical overview
 - 3.1 File structure
 - 3.2 Imaging model
 - 3.2.1 Vector graphics
 - 3.2.2 Raster images
 - 3.2.3 Text
 - 3.2.3.1 Fonts
 - 3.2.3.2 Encodings
 - 3.2.4 Transparency
 - 3.3 Interactive elements
 - 3.4 Logical structure and accessibility
 - 3.5 Security and signatures
 - 3.6 Subsets
 - 3.7 Mars
- 4 Technical issues
 - 4.1 Accessibility
 - 4.2 Security
 - 4.3 Usage restrictions and monitoring
 - 4.4 Saving form data
 - 4.5 Missing PostScript features
- 5 Content
 - 5.1 Base 14 fonts

Figure 12 The appearance of the tampered PDF document with Adobe Acrobat Reader window.

Your continued donations keep Wikipedia running!

Portable Document Format

From Wikipedia, the free encyclopedia
(Redirected from PDF)

The **Portable Document Format** (**PDF**) is the file format created by Adobe Systems in 1993 for document exchange. PDF is a fixed-layout format used for representing two-dimensional documents in a manner independent of the application software, hardware, and operating system.^[1] Each PDF file encapsulates a complete description of a 2-D document (and, with Acrobat 3-D, embedded 3-D documents) that includes the text, fonts, images, and 2-D vector graphics that compose the documents.

PDF is an open standard, and recently took a major step towards becoming the ISO 32000.^{[2][3]}

Portable Document Format (PDF)	
	
File name extension	.pdf
Internet media type	application/pdf
Type code	*PDF* (including a single space)
Uniform Type Identifier	com.adobe.pdf
Magic number	%PDF
Developed by	Adobe Systems

Contents

- 1 History
- 2 Technical foundations
 - 2.1 PostScript
- 3 Technical overview
 - 3.1 File structure
 - 3.2 Imaging model
 - 3.2.1 Vector graphics
 - 3.2.2 Raster images
 - 3.2.3 Text
 - 3.2.3.1 Fonts
 - 3.2.3.2 Encodings
 - 3.2.4 Transparency
 - 3.3 Interactive elements
 - 3.4 Logical structure and accessibility
 - 3.5 Security and signatures
 - 3.6 Subsets
 - 3.7 Mars
- 4 Technical issues
 - 4.1 Accessibility
 - 4.2 Security
 - 4.3 Usage restrictions and monitoring
 - 4.4 Saving form data
 - 4.5 Missing PostScript features
- 5 Content
 - 5.1 Base 14 fonts

Figure 13 The appearance of the verified PDF document with Adobe Acrobat Reader window.