Signal Processing 90 (2010) 557–565

Contents lists available at ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

A new approach to covert communication via PDF files

I-Shi Lee [a,c], Wen-Hsiang Tsai [a,b,*]

[a] Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan
[b] Department of Computer Science and Information Engineering, Asia University, Taichung 41354, Taiwan
[c] Department of Management Information, Technology and Science Institute of Northern Taiwan, Taipei, Taiwan

ARTICLE INFO

Article history:
Received 23 May 2008
Received in revised form
18 July 2009
Accepted 23 July 2009
Available online 30 July 2009

Keywords:
ASCII codes
Covert communication
Data hiding
Invisible codes
PDF files and readers

ABSTRACT

A new covert communication method via PDF files is proposed. A secret message, after being encoded by a special ASCII code and embedded at between-word and between-character locations in the text of a PDF file, becomes invisible in the window of a common PDF reader, creating a steganographic effect for secret transmission through the PDF file. Experimental results show the feasibility of the proposed method.

1. Introduction

Hiding data in various media with steganographic effects is a good way to covert communication. Many data hiding methods have been proposed with various types of multimedia as cover media, like images, audios, videos, etc. [1,2]. Security of data hiding techniques also has been studied [3,4]. Recently, there are more and more investigations on uses of various types of documents for data hiding, including HTML, Word, and PowerPoint files [5–7].

Portable Document Format (PDF) files [8] are popular nowadays, and so using them as carriers of secret messages for covert communication is convenient. Though there are some techniques of embedding data in text files [9–11], studies of using PDF files as cover media are very few, except Zhong et al. [12] in which integer numerals specifying the positions of the text characters in a PDF file are used to embed secret data.

In this paper, a new covert communication method by embedding secret messages in PDF files is proposed. A message is regarded as a string of bits or characters, and encoded with a special ASCII code by binary or unitary coding. The encoding results are then embedded at the *between-word* or *between-character* locations in the text part of a cover PDF file. The embedding results in the resulting stego-PDF document are found to be *invisible* in this study in the windows of common PDF readers, creating a steganographic effect and achieving the purpose of secret communication. Experimental results showing the feasibility of the proposed method are also included.

In the sequel, we describe how secret messages are encoded in Section 2, and how they are hidden and recovered in Section 3. Some experimental results are presented in Section 4, followed by some discussions in Section 5 and a conclusion in Section 6.

2. Secret message encoding using a special ASCII code

The PDF, created by Adobe Systems for document exchange [8], is a fixed-layout format for representing

* Corresponding author at: Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan. Tel.: +886 3 5715900; fax: +886 3 5721490.
E-mail addresses: gis87809@gmail.com (I.-S. Lee), whtsai@cis.nctu.edu.tw (W.-H. Tsai).

0165-1684/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.sigpro.2009.07.022

documents in a manner independent of the application software, hardware, and operating system. Each PDF file contains a complete description of a 2-D document, which includes text, fonts, images, and vector graphics described by a context-free grammar modified from PostScript. Many PDF readers are available for use to read PDF files; each PDF file appears in the window of a PDF reader as an image-like document.

On the other hand, ASCII codes were designed to represent 8-bit text characters for information interchange [13]. There are totally 256 of them among which 95 are numbered from 20 to 7E (hexadecimal) are printable. These 95 codes together with the control code 0A (for *line feeding*) are used for encoding secret messages in this study. They are listed in Table 1. The *width* of a text character represented by an ASCII code, as seen in a PDF reader's window, may be specified by a value in an array, called "widths," in the *type*-1 *font dictionary* in a PDF file [8].

It is found in this study that the ASCII code A0 (used as a *non-breaking space*), when embedded in a string of text characters, become *invisible* in the windows of several versions of the popular PDF reader, Adobe Reader, including the versions of 6.0 Professional, 7.0 Professional, 8.1.2, and 9.0 under the Windows OS environment. This phenomenon may be utilized for data hiding, as done in this study.

More specifically, two types of invisibility may be created from the ASCII code A0. One type is created by specifying the *width* of A0 appearing in the PDF reader's window to be *the same as* that of the *original white space* represented by the ASCII code 20. Here, by the *width* of an ASCII code, we mean the width of the character represented by the code as displayed in the PDF reader's window. Note that the PDF format specifications allow assignment of this width value to each character in *each* PDF page. Then, after being inserted *between two words* in the text part of a PDF file, each A0 appears to be exactly the same as a white space exhibited by the code 20. Fig. 1 illustrates this phenomenon. That is, both A0 and 20 become white spaces. So A0 and 20 may be used in a PDF text *alternatively* as a *between-word space* to encode a message bit b according to the following *binary coding* technique:

$$\begin{aligned} &\text{if } b = 1, \text{ then replace 20 between two words by A0;} \\ &\text{if } b = 0, \text{ make no change.} \end{aligned} \tag{1}$$

We will call such a binary coding rule *alternative space coding*, and call 20 and A0 both as *spacing codes* in the sequel. For example, given the four consecutive words "the boy ran quickly" in a PDF page, there are totally three *between-word locations* among the four words, and at each location a space is seen in a PDF reader's window. We may embed three message bits "010" at these locations by replacing the original ASCII code 20 representing the space at the second between-word location with the code A0 and leave the code 20 at the other two between-word locations unchanged.

The other type of invisibility is created by setting the width of the ASCII code A0 to be *zero* in a PDF page. Then after being inserted *between two characters*, an A0 appears to be *nothing* in a PDF reader's window just like *non-existent*, as found in this study. Fig. 2(b) illustrates this phenomenon. For a comparison, Fig. 2(a) is also included. This invisibility is still true even when multiple A0's are all embedded together at a single *between-character location*. Fig. 3 illustrates this phenomenon. The code A0 here is said to be used as a *null code*, contrasting with the term

Table 1
ASCII codes selected for message representations in this study.

Index	Character	Hexadecimal code	Index	Character	Hexadecimal code	Index	Character	Hexadecimal code	Index	Character	Hexadecimal code
1	*LF*	0A	25	7	37	49	O	4F	73	g	67
2		20	26	8	38	50	P	50	74	h	68
3	!	21	27	9	39	51	Q	51	75	i	69
4	″	22	28	:	3A	52	R	52	76	j	6A
5	♯	23	29	;	3B	53	S	53	77	k	6B
6	$	24	30	<	3C	54	T	54	78	l	6C
7	%	25	31	=	3D	55	U	55	79	m	6D
8	&	26	32	>	3E	56	V	56	80	n	6E
9	′	27	33	?	3F	57	W	57	81	o	6F
10	(	28	34	@	40	58	X	58	82	p	70
11	)	29	35	A	41	59	Y	59	83	q	71
12	*	2A	36	B	42	60	Z	5A	84	r	72
13	+	2B	37	C	43	61	[	5B	85	s	73
14	,	2C	38	D	44	62	\	5C	86	t	74
15	−	2D	39	E	45	63	]	5D	87	u	75
16	.	2E	40	F	46	64	∧	5E	88	v	76
17	/	2F	41	G	47	65	_	5F	89	w	77
18	0	30	42	H	48	66	‘	60	90	x	78
19	1	31	43	I	49	67	a	61	91	y	79
20	2	32	44	J	4A	68	b	62	92	z	7A
21	3	33	45	K	4B	69	c	63	93	{	7B
22	4	34	46	L	4C	70	d	64	94	|	7C
23	5	35	47	M	4D	71	e	65	95	}	7D
24	6	36	48	N	4E	72	f	66	96	∼	7E

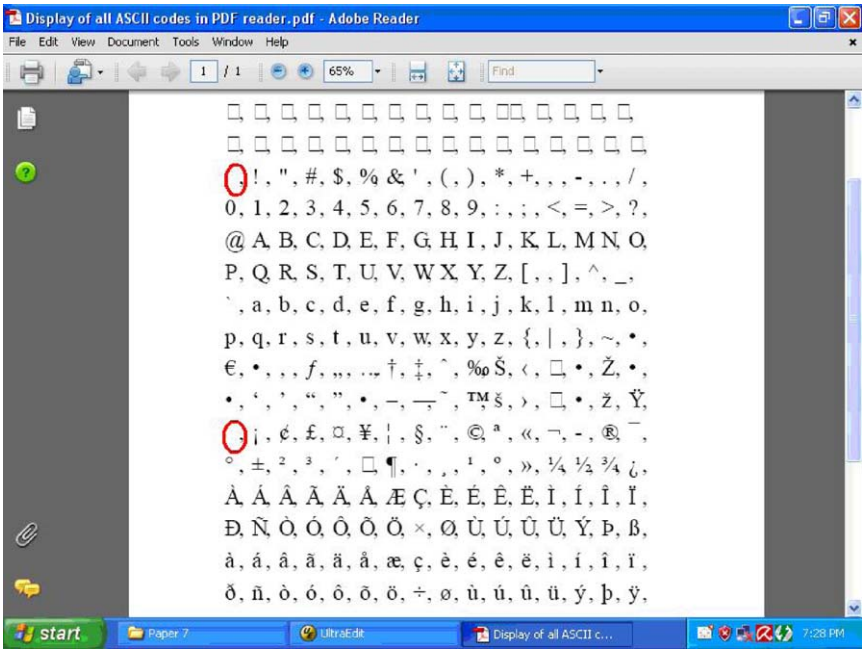

Fig. 1. Display of all ASCII codes in an Adobe Reader's window, in which only 20 and A0 appear to be white spaces (the first spaces in the 3rd and the 11th lines as circled).

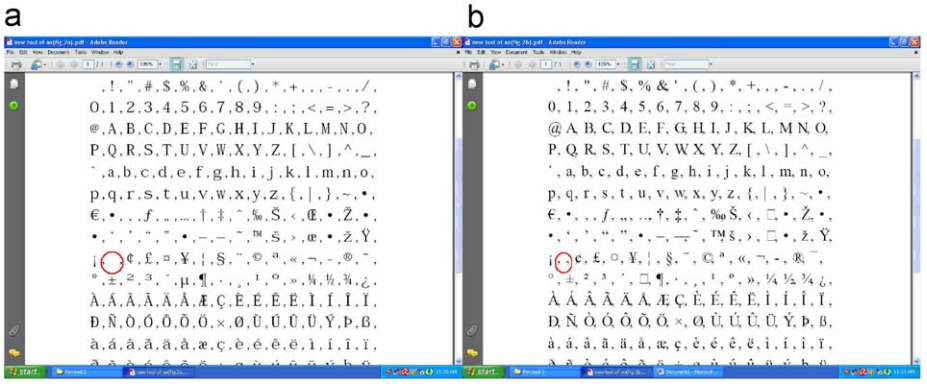

Fig. 2. Displays of all ASCII codes in Adobe Reader window, in which two different widths of A0 are included in 9th row: (a) width of A0 set to be normal so that A0 is displayed like a space after 1st comma and before 2nd comma (as circled); (b) width of A0 set to be zero so that A0 becomes non-existent after 1st comma and before 2nd comma in 9th row (as circled), as compared with Fig. 2(a).

spacing code used for alternative space coding described by (1) above.

Note that the width of the original space code 20 cannot be changed to be zero because it is used as a normal space between every two words in a PDF file. Otherwise, the words in the text in a PDF page will become *all concatenated* into a character string without between-word spaces. So, when A0 is used as a null code, the space code 20 cannot be used symmetrically to implement a binary coding like (1). But we may still use A0 to hide a *message character C* by a technique of *unitary coding* at a between-character location *L* in the following way:

if the index of *C* as specified in Table 1 is *m*,
then embed *m* consecutive A0's at *L* (2)

which will be called *null space coding*. Note that unitary coding means the use of a single symbol (A0 here) to encode messages. Also note that A0 can only be used in one of the two ways of coding and not in both in *each page* of a PDF file because its width can only be specified *once* for each PDF page.

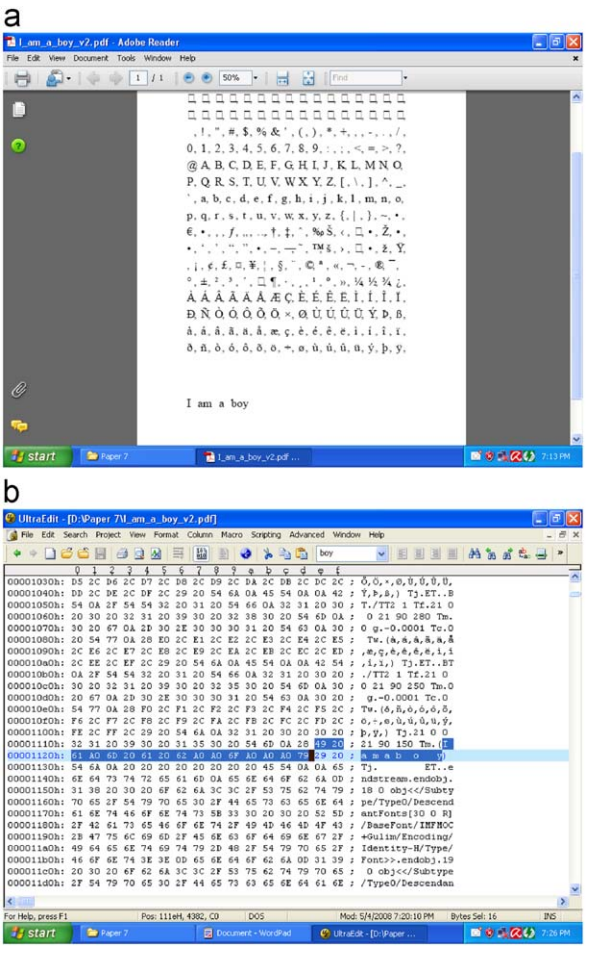

Fig. 3. Invisibility of multiple A0's at between-character locations: (a) appearance of a sentence "I am a boy" in Adobe Reader window with one, two, and three A0's inserted at locations between the characters a and m, b and o, and o and y; (b) appearances of the A0's in UltraEdit window (in highlighted portion).

Alternative space coding has the advantage of incurring no increase of the PDF file size because it just replaces the space exhibited by the code 20 by another exhibited by the code A0. However, if the between-word locations in a PDF page are few, then only a small number of bits may be embedded. On the contrary, since theoretically an *unlimited* number of A0's may be inserted as null codes at a *single* between-character location, and since there are much more between-character locations than between-word locations, the encoding efficiency of null space coding will be much higher. But an obvious disadvantage is that the resulting PDF file size will increase.

According to the above discussions, three ways of coding for use in different application conditions, namely, *pure* alternative space coding, *pure* null space coding, and a *mixture* of them can be identified. The advantages and weaknesses of the first two ways have been analyzed as above. As to the use of the third way, we may use alternative space coding first to embed as many bits in the secret message as possible, and then apply null space coding to embed all the remaining portion of the message (in unit of character) using the last pages of the PDF file. In this way, the file size increase problem of pure null space coding may be reduced.

3. Message hiding and recovery

Normal text messages may be represented by the 96 characters listed in Table 1. To apply alternative space coding for data embedding, an input secret message should be transformed first into a bit string B_1. For this, we concatenate the binary ASCII codes (each consisting of 8 bits) of the characters in the secret message to form B_1. The number of bits in B_1 is counted next and expressed as another bit string B_2 with a pre-selected length. And the two strings B_1 and B_2 then are concatenated to form a third string $B_3 = B_2B_1$. The string B_3 is finally embedded, bit by bit sequentially, into the between-word locations in the cover PDF file according to Rule (1) above. Note that it is necessary to embed the number of message bits (expressed as B_2) in order to extract the same number of hidden message bits correctly during the message recovery process.

To implement null space coding, the secret message is regarded as a string of characters represented by the 96 ASCII codes listed in Table 1. However, to reduce the total number of inserted A0's to make the resulting stego-file size as small as possible, we propose *not* to apply Rule (2) above directly where the number of A0's used to encode a character C is *just* the index value m of C as specified in Table 1. Instead, we apply the technique of *Huffman coding* here, aiming to use less A0's to encode a character with a higher occurrence frequency in the secret message. That is, we assign one A0 to encode the character with the largest occurrence frequency in the message, two A0's to encode the character with the second largest occurrence frequency, and so on. And those characters among the 96 ones which do not appear in the secret message are not encoded, nor processed. The encoding result is summarized as a table, called the *null space coding table*, with the entries filled with the corresponding numbers of A0's so obtained. For example, given the message "This is a covert communication method," after counting the occurrence frequencies of the 16 distinct characters in it, the corresponding null space coding table is as shown in Table 2 in which only 16 entries are filled.

Every message will have a distinct null space coding table. For message decoding, the table should be embedded as well in the cover file as part of the hidden data. In practice, we do not embed all the content of the table, but the numbers of A0's only, into the first 96 consecutive between-character locations in the text of the cover PDF. *No* A0 is embedded at a corresponding between-character location to represent each *unprocessed* character mentioned previously.

The data recovery process is essentially a reverse of the data hiding process, with retrieval of the null space coding table conducted first, followed by extraction and decoding of the hidden message, when the mixture coding is used. More detailedly, for each page of the PDF file we check if any A0 is used in the page. If so, then we check further the

Table 2
Null space coding table for message "This is a covert communication method."

Index	Character	#A0's embedded	Frequency	Index	Character	#A0's embedded	Frequency	Index	Character	#A0's embedded	Frequency	Index	Character	#A0's embedded	Frequency
1	*LF*	12	1	25	7			49	O			73	g		
2		1	5	26	8			50	P			74	h	9	2
3	!			27	9			51	Q			75	i	2	4
4	″			28	:			52	R			76	j		
5	#			29	;			53	S			77	k		
6	$			30	<			54	T	13	1	78	l		
7	%			31	=			55	U			79	m	5	3
8	&			32	>			56	V			80	n	10	2
9	′			33	?			57	W			81	o	3	4
10	(			34	@			58	X			82	p		
11	)			35	A			59	Y			83	q		
12	*			36	B			60	Z			84	r	15	1
13	+			37	C			61	[			85	s	11	2
14	,			38	D			62	\			86	t	6	3
15	−			39	E			63	]			87	u	16	1
16	.			40	F			64	^			88	v	17	1
17	/			41	G			65	_			89	w		
18	0			42	H			66	‘			90	x		
19	1			43	I			67	a	7	2	91	y		
20	2			44	J			68	b			92	z		
21	3			45	K			69	c	4	3	93	{		
22	4			46	L			70	d	14	1	94	|		
23	5			47	M			71	e	8	2	95	}		
24	6			48	N			72	f			96	~		

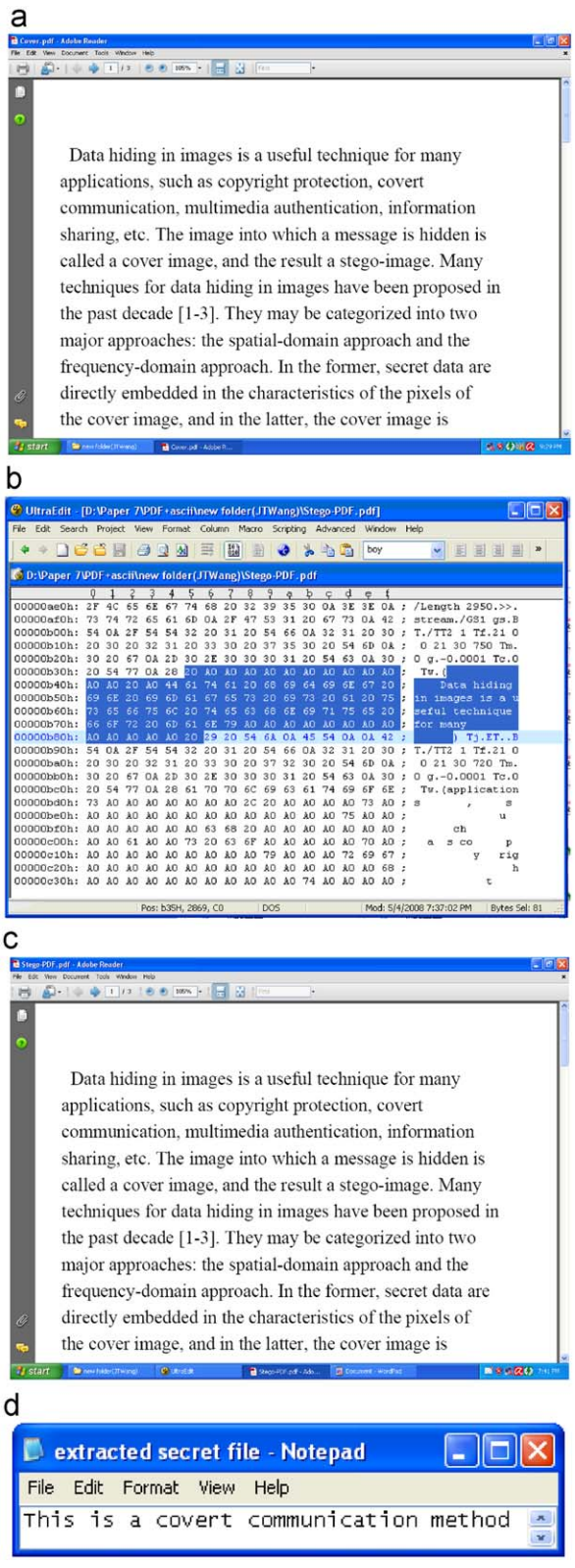

Fig. 4. An experimental result of null space coding: (a) cover file seen in Adobe Reader 8.1.2 window; (b) stego-file seen in UltraEdit window; (c) stego-file seen in Adobe Reader 8.1.2 window; (d) Extracted message.

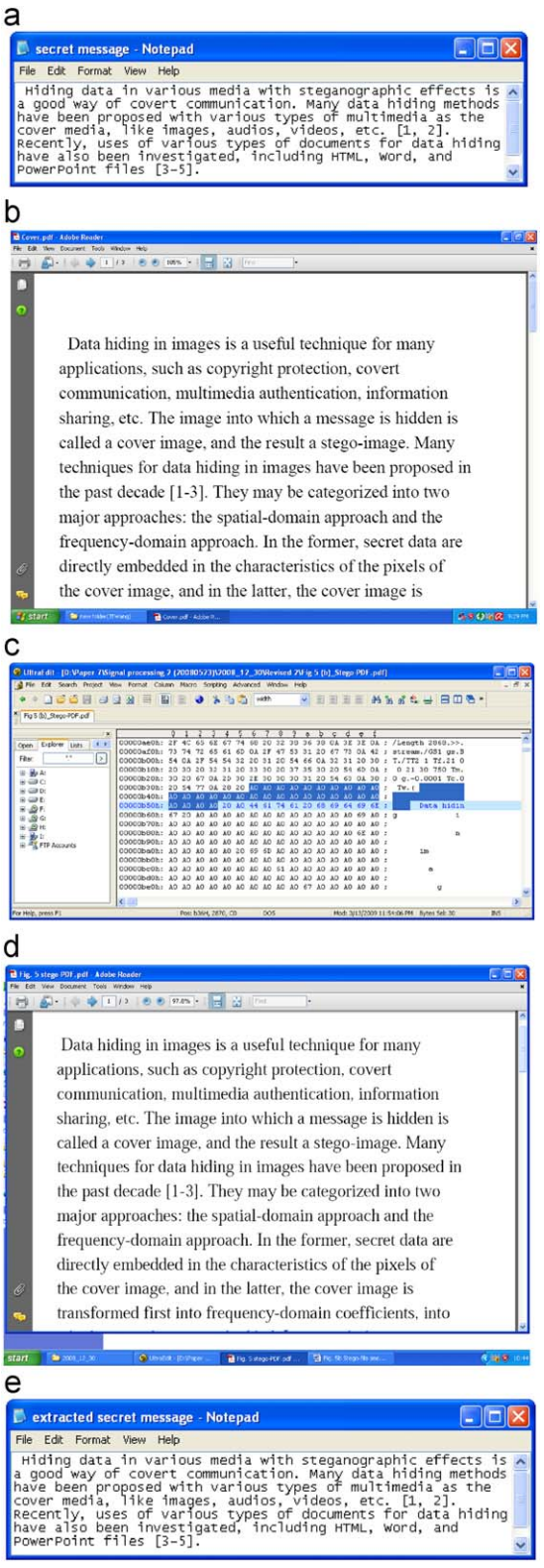

Fig. 5. Another experimental result of null space coding: (a) input message; (b) cover file seen in Adobe Reader 8.1.2 window; (c) stego-file seen in UltraEdit window; (d) stego-file seen in Adobe Reader 8.1.2 window; (e) extracted message.

Table 3
Null space coding table for a secret message with 374 characters.

Index	Character	#A0's embedded	Frequency	Index	Character	#A0's embedded	Frequency	Index	Character	#A0's embedded	Frequency	Index	Character	#A0's embedded	Frequency
1	*LF*	30	1	25	7			49	O			73	g	13	9
2		1	60	26	8			50	P	26	2	74	h	14	9
3	!			27	9			51	Q			75	i	3	30
4	″			28	:			52	R	37	1	76	j		
5	#			29	;			53	S			77	k	40	1
6	$			30	<			54	T	38	1	78	l	19	6
7	%			31	=			55	U			79	m	15	9
8	&			32	>			56	V			80	n	9	17
9	′			33	?			57	W	39	1	81	o	5	25
10	(			34	@			58	X			82	p	20	5
11	)			35	A			59	Y			83	q		
12	*			36	B			60	Z			84	r	11	10
13	+			37	C			61	[	27	2	85	s	6	20
14	,	12	9	38	D			62	\			86	t	7	20
15	−	31	1	39	E			63	]	28	2	87	u	16	9
16	.	22	4	40	F			64	^			88	v	17	9
17	/			41	G			65	_			89	w	23	4
18	0			42	H	24	2	66	‘			90	x		
19	1	32	1	43	I			67	a	4	26	91	y	21	5
20	2	33	1	44	J			68	b	29	2	92	z		
21	3	34	1	45	K			69	c	10	10	93	{		
22	4			46	L	36	1	70	d	8	19	94	|		
23	5	35	1	47	M	25	2	71	e	2	31	95	}		
24	6			48	N			72	f	18	8	96	∼		

width of the A0 as specified in the PDF format specification of this page—if the specified width is non-zero, then it is decided that alternative space coding was applied to the current PDF page, and all between-word A0's or 20's are extracted and decoded to get the length and the body of the message; otherwise, it is decided that null space coding was applied, and all the between-character A0's are extracted and decoded to get the coding table as well as the secret message.

4. Experimental results

First, we report the results of two of the experiments we conducted for null space coding. In the first experiment, the input secret message is "This is a covert communication method" to which the constructed corresponding null space coding table is shown in Table 2. After the table content followed by the message was embedded with a Java program written in this study into the cover PDF file shown in Fig. 4(a), the initial part of the stego-file appearing in the UltraEdit window is shown in Fig. 4(b), from which we can see the repeating A0's encoding each of the 96 commonly used characters. In particular, we can see in the highlighted portion the 12 A0's representing the ASCII code 0A (line feed), the single A0 representing the space code 20, the 13 A0's representing the character *T* (the first character in the secret message), and so on. The stego-file appears as Fig. 4(c) which is identical to Fig. 4(a). The recovered message is shown in Fig. 4(d).

In the second experiment, the input message is 374 characters in length (including space characters), as shown in Fig. 5(a). The constructed null space coding table for it is shown in Table 3. The results are shown in Figs. 5(b–e), where all figures are of the same meaning of those of Fig. 4. In particular, the stego-file appears as Fig. 5(d) which is identical to Fig. 5(b), and the recovered message is shown in Fig. 5(e). This experiment also shows the correctness of the proposed method, and so do the other experiments we have conducted and not reported here.

5. Discussions

About the hiding capacity of the proposed method, given a PDF file F with N between-character locations, if the proposed pure null space coding technique is adopted for data embedding, then the file may be used to embed a message of at most $N-96$ characters, allowing 96 locations for use in saving the null space coding table. Generally speaking, this yields a large capacity because usually there are very many characters in a text-type PDF document so that $N-96$ is large. We compare this hiding capacity here with that yielded by the method proposed by Zhong et al. [12] to show the effectiveness of the proposed method. In Zhong et al. [12], an example of experimental results using a file of six PDF pages with 24,026 characters was reported, and the number of embedded bits was 46,704 which are equivalent to $46{,}704/8 = 5838$ characters. Contrastively, our method, by using the same file size, can embed $N-96 = 24{,}026-96 = 23{,}930$ characters which is about four times larger.

Furthermore, about the data embedding rate, the proposed alternative space coding can be applied at *any* between-word space. According to Wikipedia [14], the average length of an English word plus a space is six characters, which is equivalent to 6×8 bits. Since the proposed alternative encoding technique embeds a bit at each between-word location, the bit embedding rate is $r_1 = 1/(6 \times 8) = 1.08\%$ which is not high. However, the other proposed coding technique, null space coding, embeds a character between two characters by using a coding table of 96 characters; therefore, by assuming N as the number of characters in the PDF file, the null space coding method can embed $(N-96) \times 8$ bits message, so that the bit embedding rate may be figured out to be $r_2 = [(N-96) \times 8]/(N \times 8)$, which approaches 100% when N is large!

About the security of the proposed method, as usual we assume that the proposed algorithms are known to the public. So, the embedded message may be tampered with by an illegal user who knows the algorithms. A solution to this problem is to use a secret key to randomize the message data as well as the data-embedding locations in advance using a random number generator before the message data are embedded. This makes the hidden data to be recoverable only when the correct key is used as input to the data extraction process.

6. Conclusion

A new covert communication method via PDF files is proposed. Two techniques of message data encoding, namely, alternative space coding and null space coding, using the special ASCII code A0 have been proposed for message embedding in the texts of PDF files. In the two techniques, the message bits are embedded respectively between words or between characters and become invisible in the windows of common PDF readers. For between-word embedding, the width of A0 is set to be the same as that of the space code 20, and for between-character embedding, the width is set to be zero. Experimental results show the feasibility of the proposed method. Future researches may be directed to applying the data hiding scheme to other applications like watermarking of PDF files for copyright protection, authentication of PDF file for tampering proof, etc.

Acknowledgments

This work was supported jointly by the NSC projects nos. 96-2752-E-009-006-PAE and 96-2422-H-009-001. We also thank Mr. Jiun-Tsung Wang for his programming support and suggestions.

References

[1] S. Katzenbeisser, F.A.P. Petitolas, Information Hiding Techniques for Steganography and Digital Watermarking, Artech House, Boston, Massachusetts, USA, 2000.

[2] F.A.P. Petitcolas, R.J. Anderson, M.G. Kuhn, Information hiding—a survey, Proceedings of the IEEE 87 (7) (July 1999) 1062–1078 special issue on Protection of Multimedia Content.
[3] S. Voloshynovskiy, T. Pun, J.J. Fridrich, F. Pérez-González, N. Memon, Security of data hiding technologies, Signal Processing 83 (10) (2003) 2065–2067.
[4] M. Barni, F. Bartolini, T. Furon, A general framework for robust watermarking security, Signal Processing 83 (10) (October 2003) 2069–2084.
[5] D.C. Wu, P.H. Lai, Novel techniques of data hiding in HTML documents, in: Proceedings of 2005 Conference on Digital Contents Managements & Applications, Kaohsiung, Taiwan, June 2005, pp. 21–30.
[6] T.Y. Liu, W.H. Tsai, A new steganographic method for data hiding in microsoft word documents by a change tracking technique, IEEE Transactions on Information Forensics and Security 2 (1) (March 2007) 24–30.
[7] T.Y. Liu, W.H. Tsai, Robust watermarking in slides of presentations by blank space coloring: a new approach, LNCS Transactions on Data Hiding and Multimedia Security, accepted for publication.
[8] Adobe Systems Incorporated, Portable Document Format Reference Manual, version 1.7, November 2006, 〈http://www.adobe.com〉.
[9] W. Bender, D. Gruhl, N. Morimoto, A. Lu, Techniques for data hiding, IBM System Journal 35 (3, 4) (February 1996).
[10] H.M. Meral, E. Sevinc, E. Unkar, B. Sankur, A.S. Ozsoy, T. Gungor, Syntactic tools for text watermarking, in: Proceedings of SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents, San Jose, CA, USA, January 29–February 1, 2007.
[11] M. Topkara, U. Topkara, M.J. Atallah, Information hiding through errors: a confusing approach, in: Proceedings of SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents, San Jose, CA, USA, January 29–February 1, 2007.
[12] S. Zhong, X. Cheng, T. Chen, Data hiding in a kind of PDF texts for secret communication, International Journal of Network Security 4 (1) (January 2007) 17–26.
[13] ASCII Code—the extended ASCII table, retrieved April 30, 2008, from 〈http://www.ascii-code.com/〉.
[14] Wikipedia: size comparisons, retrieved March 5, 2009, from 〈http://en.wikipedia.org/wiki/Wikipedia:Size_comparisons〉.