

# Data Hiding via XPS Documents – A New Study

Mei-Hua Ho

Department of Computer Science  
National Chiao Tung University  
Hsinchu, Taiwan  
e-mail: mei.cs97g@nctu.edu.tw

Wen-Hsiang Tsai

Department of Computer Science  
National Chiao Tung University  
Hsinchu, Taiwan  
e-mail: whtsai@cis.nctu.edu.tw

**Abstract**—A new study on information hiding via the recently-proposed XPS (XML Paper Specification) document is conducted. Three data hiding methods utilizing the properties of the XPS document format are proposed. The first hierarchically divides an image in an XPS document in various ways for message data encoding and embedding. The second method superimposes specially-designed gradient patterns on an XPS document to hide a message. And the third method inserts invisible and width-adjustable ASCII codes between words in the text of an XPS file to embed a secret message. The proposed data hiding methods are used for the applications of covert communication, text authentication, and steganography. Measures to enhance the security of each method are also suggested. Good experimental results show the feasibility of the proposed methods.

**Keywords** - XPS document, information hiding, covert communication, authentication, gradient pattern, ASCII code.

## I. INTRODUCTION

With the progress of development in computer and networking technologies, digital text documents nowadays has become much more popular than in the past. The XPS document is a new text file format which is portable to any computer to produce identical page layouts with high printing and browsing qualities. It has become one of the major document formats used in daily communication between people. Because the data hiding technique is a good way for safe exchanges of information, it is desired to develop data hiding techniques via XPS documents in this study. To our knowledge of the literature about information hiding research, this work has not been attempted so far.

Existing studies on data hiding techniques via text documents are not as many as those via images or videos because of the lack of redundant information in texts for embedding data. Liu and Tsai [1] utilized a change tracking technique in Microsoft Word documents to disguise a stego-document as a normal collaborative document. The secret data are embedded by degeneration of word usages in the content of a cover document. Inoue et al. [2] proposed five techniques to embed secret data into XML documents, including (1) using different representations of an empty XML element, (2) inserting white spaces in tags, (3) exchanging the order of XML elements, (4) exchanging the order of attributes in XML elements, and (5) exchanging inner-tags and outer-tags. Meanwhile, Zhong, et al. [3] inserted secret data between indirect objects and modified the cross reference table in PDF files for data hiding. Also,

Wang and Tsai [4] achieved authentication of PDF files by embedding authentication signals using the modified values of PDF object parameters, resulting in a slight difference of the PDF appearance that is hard to notice by human eyes. However, studies on data hiding via XPS documents are not found yet. It is so desirable to design new methods for data hiding via XPS documents by use of new features found in the XPS format.

XPS documents contain clear logical and physical hierarchies, compared with other similar document formats. Physically, the XPS document is a compressed ZIP archive called a *package*, which consists of an XML markup file for each page as well as other resources including fonts, images, thumbnails, etc. The logical hierarchy of an XPS document is illustrated in Fig. 1. Every component or file stored in the XPS document is called a *part* of the package and the connection between parts and the document is called *relationships*. The most important part of an XPS document is the *fixed page part* because it describes how a page is rendered.

XPS documents are described by an XML-based language, resulting in a fixed-layout document. All rules and elements used to compose an XPS document are specified in the XPS Specification [5]. The <Glyphs> and the <Path> are two major elements used to create graphics and texts with a set of attributes to describe the characteristics of graphics and texts, such as position, size, color, and so on. Accordingly, by modifying the elements or the attribute of elements, the page layout may be changed. Our goal in this study is to embed data secretly by modifying the description of the pages of the original XPS document without changing the appearance.

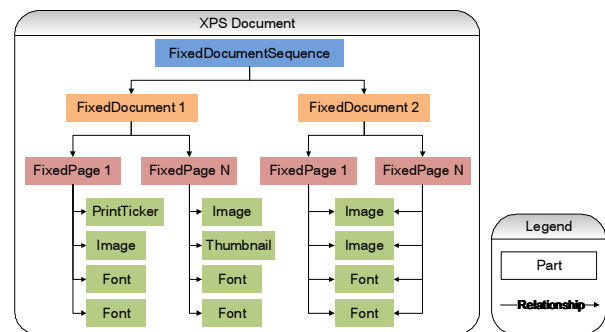


Figure 1. Logical hierarchy of an XPS document.

In this study, three new data hiding methods via XPS documents utilizing properties found in the XPS format are proposed. First, it is found that an image in the XPS document can be *partially* displayed in a page by changing the properties which describe the image. Based on this observation, the first proposed data embedding method is designed to divide an image into various block patterns to encode the message data to be hidden and reconstructs the original image *block by block* to keep the same appearance of the image in the XPS document. The method is used for covert communication in this study.

Next, it is found also that *gradient patterns* described according to the XPS specification may be made invisible when superimposed on an XPS document. Therefore, the second proposed method uses various gradient patterns to encode secret messages and superimpose them on documents without arousing notice from observers. This method is used for text authentication of XPS documents in this study.

Finally, by experiments conducted in this study, it was found that some ASCII codes used in the text string of the XPS document may be made invisible by adjusting their *advance widths*. This property of the XPS document format is good for use in data hiding applications like steganography.

In the remainder of this paper, the three proposed data hiding techniques are described in detail from Sections II through IV. Experimental results are also included respectively to show the feasibility of the proposed methods. Security enhancements for the proposed methods are discussed for each method. Finally, conclusions are made in the last section.

## II. DATA HIDING BY HIERARCHICAL DIVISION OF IMAGES IN XPS DOCUMENTS FOR COVERT COMMUNICATION









In the XPS specification, the <Path> element can be used to *create* an area to display an image in an XPS document and the Data attribute can be used to *describe* the area of the image. Accordingly, an image can be *partially* displayed by narrowing the area described by the Data attribute. And by this function, an image can be hierarchically divided into blocks using multiple <Path> elements. Utilizing these properties, we generate *block patterns* with two levels of divisions to encode message bits, and the difference in appearance between the original *cover image* and the resulting *stego-image* is found imperceptible in this study. Note that the image is *not really* divided — division of it is just conducted in the XML markup of the XPS document; the appearance of the resulting XPS document is totally unaffected and so will arouse no notice from any observer of the image in the document.

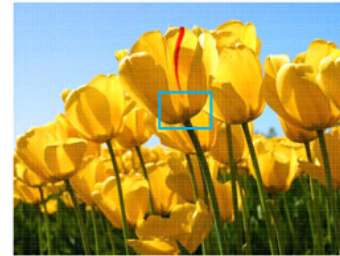
### A. Embedding data in a cover document

The data hiding process is based on the above hierarchical image division and display for message encoding makes use of a table designed in this study, which includes a list of block patterns obtained by two-level image divisions and a set of corresponding 3-bit codes, as shown in Table 1, which we call the *division pattern encoding table*

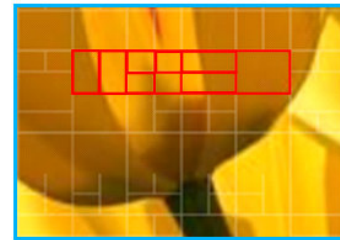
subsequently. Accordingly, a message can be embedded into an XPS document by dividing a selected cover image in it into blocks with their division patterns corresponding to the message bits. Fig. 2 shows an illustration. The detail is described in the following algorithm.

Table 1. Division pattern encoding table.

Division pattern	Corresponding binary code	Division pattern	Corresponding binary code
	000		100
	001		101
	010		110
	011		111



(a)



(b)

Figure 2. An image in an XPS document with a message embedded in it. (The edges of the blocks are emphasized on purpose in order to show the result.) (a) The entire image with division patterns superimposed (not seen in real appearance). (b) The enlarged partial view of (a) with the red rectangular part corresponds to a partial message  $S = 100\ 001\ 010\ 000$ .

### Algorithm 1. Data embedding by image division.

**Input:** a secret message  $S$ , a cover XPS document  $D$ , and a secret key  $K$ .

**Output:** a stego-XPS document  $D'$ .

**Steps:**

1. Use the secret key  $K$  as a seed to generate a sequence  $Q$  of random numbers.
2. Randomize the characters of the secret message  $S$  with  $Q$  to get a randomized message  $S'$ , and let  $l$  denote the number of characters in  $S'$ .
3. Separate  $S'$  into a series of 3-bit segments  $s_1, s_2, \dots, s_k$  according to the following steps.
  - 3.1 Add a bit 0 at the end of the 8-bit code representing each character  $c_i$ , resulting in a 9-bit segment  $c_i'$ .

- 3.2 Separate  $c_i'$  into 3-bit segments  $s_{3i+1}$ ,  $s_{3i+2}$ , and  $s_{3i+3}$ , where  $0 \leq i \leq l-1$ .
- 3.3 Add a 9-bit ending signal consisting of three 3-bit segments of the forms  $s_{k+1} = 000$ ,  $s_{k+2} = 000$ ,  $s_{k+3} = 001$  at the end of  $s_k$ , where  $k = 3 \times l$ .
4. Map the 3-bit segments  $s_1, s_2, \dots, s_{k+3}$  into a series of division patterns  $p_1, p_2, \dots, p_{k+3}$  according to Table 1.
5. Add a unit block denoted by  $p_0$  at the beginning of the series of division patterns.
6. Perform the following steps on the cover document  $D$ .
  - 6.1 Decompress the XPS file of  $D$ .
  - 6.2 Find a minimum number  $N$  such that  $N^2 \geq k+4$ .
  - 6.3 Select an image  $I$  in  $D$  and modify the XML markup file describing  $I$  to divide  $I$  in the following way.
    - 6.3.1 Divide image  $I$  into  $n \times n$  blocks,  $B_0, B_1, \dots, B_{n \times n - 1}$ .
    - 6.3.2 Divide each block  $B_i$ ,  $0 \leq i \leq n \times n - 1$ , according to the corresponding division pattern  $p_i$ , until the patterns  $p_{k+1}$ ,  $p_{k+2}$ , and  $p_{k+3}$  corresponding to the ending signal are reached and processed.
  - 6.4 Call the final divided image  $I$  the *stego-image*, and denote it by  $I'$ .
7. Recompress  $D$  (with  $I'$  in it) with the modified XML file to get a stego-XPS document  $D'$ .

### B. Extracting data from a stego-document

The extraction process is similar to the embedding process but conducted essentially in a reverse order. First, we extract the unit block embedded at the beginning of the stego-image to get the information of the block size of the division patterns so that we can decode all the remaining division patterns in the stego-image. The decoding process stops when the ending signal is extracted. Hence, even when we do not know the length of the secret message, we still know where the end of the message is in the stego-image. By using the same secret key, we can recover the correct message.

#### Algorithm 2. Extracting data from a stego-image.

**Input:** a stego-document  $D'$  and the secret key  $K$  used in Algorithm 1.

**Output:** a secret message  $S$ .

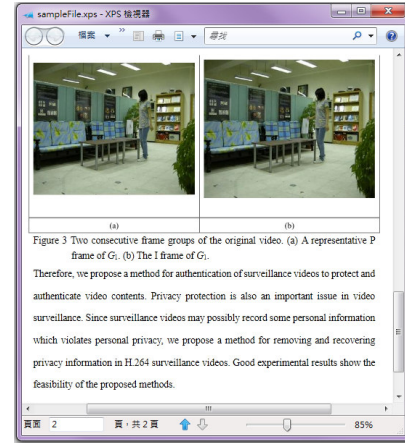
**Steps:**

1. Decompress the stego-document  $D'$ .
2. Find the XML markup file in  $D'$  which describes a stego-image  $I'$ .
3. Extract the first block from  $I'$ , supposed to be a unit block, and get its height and width.
4. Extract all subsequent division patterns  $p_1, p_2, \dots, p_k$  from  $I'$  until the division patterns  $p_{k+1}, p_{k+2}$ , and  $p_{k+3}$  corresponding to the 9-bit ending signal are encountered.
5. Decode the division patterns  $p_1, p_2, \dots, p_k$  into corresponding 3-bit segments  $s_1, s_2, \dots, s_k$  according to Table 1.

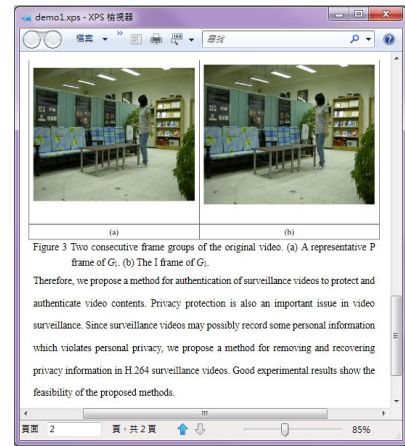
6. Concatenate every three segments  $s_{3i+1}, s_{3i+2}$ , and  $s_{3i+3}$  into one and discard the last bit of it to get an 8-bit character  $c_i$ , where  $0 \leq i \leq l-1$ .
7. Concatenate  $c_0, c_2, \dots, c_{l-1}$  into a string  $S'$ .
8. Use the secret key  $K$  to generate a random number sequence  $Q$  and use  $Q$  to reorder  $S'$  to get the desired secret message  $S$  as output.

### C. Experimental results

Fig. 3 shows an example of the experimental results where a message “I want to tell you a secret” is embedded by Algorithm 1 into an image of the XPS document shown in Fig. 3(a), resulting in the stego-document of Fig. 3(b) which looks identical to Fig. 3(a). Fig. 4 shows message data extraction results using Algorithm 2 with correct and wrong keys. The experimental results show that the proposed method is effective for covert communication applications.



(a)



(b)

Figure 3. Data hiding by division of images in XPS documents. (a) Original XPS document. (b) Stego-document.

#### D. Security considerations and enhancements

In the proposed method, the secret key and the division pattern encoding table are known only by both the sender and the receiver beforehand. Thus, a malicious user cannot extract the secret message successfully without the correct secret key. However, he/she may observe the regularity of the patterns and guess the secret message by trial and error. He/she may disturb or replace some division patterns existing in the stego-document, resulting in extracting a wrong message. To prevent these situations in advance, in addition to using a secret key to randomize the secret message content before embedding it (Step 2 of Algorithm 1), we may also use the key to reorder the binary value entries of the division pattern encoding table. Furthermore, the encoding table may be redefined randomly using different division patterns or be extended randomly to represent codes with lengths larger than 3 bits, both ways controlled by another key.

### III. AUTHENTICATION OF XPS DOCUMENT CONTENTS BY SUPERIMPOSITION OF VARIABLE GRADIENT PATTERNS

We have developed in this study an authentication process based on the second proposed data hiding method using variable gradient patterns, not only to detect whether an XPS document has been tampered with or not, but also to highlight which part of the texts has been changed.

In more detail, *linear gradient patterns* created by using the XML markup language in the XPS can be superimposed onto the pages of XPS documents. Invisibility of the patterns is achieved by changing the transparency for the document. An example is shown in Fig. 5. Accordingly, we design a table, called *gradient pattern encoding table*, which includes a list of gradient patterns and a set of corresponding codes, as shown in Table 2. Thus, an XPS document can be protected by transforming authentication signals into several linear gradient patterns which are superimposed *invisibly* onto every page of the document.

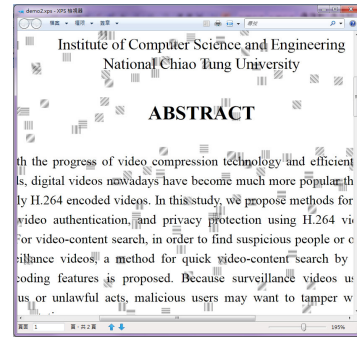
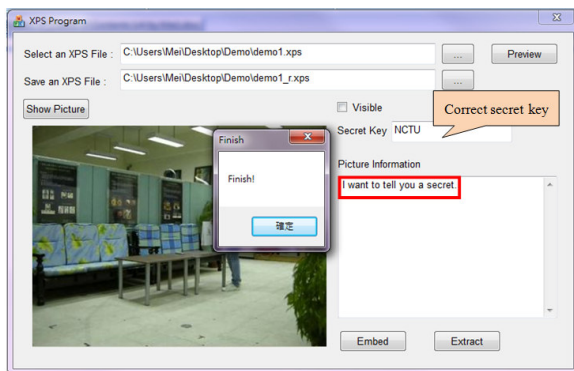
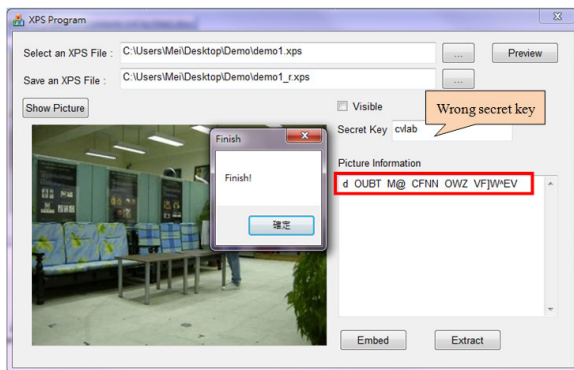


Figure 5. A protected XPS document with variable gradient patterns superimposed (the patterns are shown for illustration; they actually are transparent in the real case).



(a)



(b)

Figure 4. Data extraction from stego-document. (a) Extraction result using right key. (b) Extraction result using wrong key.

Table 2. Gradient pattern encoding table.

Gradient pattern (Repeat)				
Binary code	0000	0001	0010	0011
Gradient pattern (Repeat)				
Binary code	0100	0101	0110	0111
Gradient pattern (Reflect)				
Binary code	1000	1001	1010	1011
Gradient pattern (Reflect)				
Binary code	1100	1101	1110	1111

#### A. Authentication signal generation and embedding

To generate authentication signals, we use a hash function to create *equal-length digests* of text segments and other information contained in the <Glyphs> element. Also, we create additionally a digest of a user key and combine it with the authentication signals to prevent the authentication signals from being forged. As a result, the text content and even its position in the XPS document can be protected. The details are described in the following algorithms.

#### Algorithm 3. Authentication signal generation and embedding by variable gradient pattern supervision.

**Input:** a secret key  $K$ , a hash function  $f$  (such as MD5), and an XPS document  $D$  to be protected.

**Output:** a protected XPS document  $D'$ .

**Steps:**

1. Use the secret key  $K$  as an input to the hash function  $f$  to generate a 64-bit digest  $K'$ .
2. Choose an *unprotected* text segment  $T$  in the original document  $D$  (with all text segments in  $D$  regarded as unprotected initially).
3. Use  $T$  as an input to the hash function  $f$  to generate a 64-bit digest  $T'$ .
4. Perform the exclusive-OR operation  $\oplus$  to  $K'$  and  $T'$  to get a 64-bit authentication signal  $S = K' \oplus T'$ .
5. Separate the bits of  $S$  into a series of 4-bit segments  $t_1, t_2, \dots, t_{16}$ .
6. Map  $t_1, t_2, \dots, t_{16}$  into a series of gradient patterns  $p_1, p_2, \dots, p_{16}$  according to Table 2.
7. Starting at the beginning of the text segment  $T$ , superimpose  $p_1, p_2, \dots, p_{16}$  sequentially onto  $T$  in  $D$  invisibly by adjusting the transparency parameter value.
8. Repeat Steps 2 through 7 until all text segments are protected.
9. Recompress the modified  $D$  to get a protected XPS document  $D'$ .

#### B. Authentication signal extraction and verification

To authenticate a protected XPS document  $D'$ , we use the same secret key and hash function as those used in Algorithm 3 to compute an authentication signal for each *current* text segment  $T$  in  $D'$ . Also, we extract the authentication signal from the gradient patterns superimposed on  $T$ . The compute authentication signal is then compared with the extracted one to decide whether  $T$  has been modified or not. More details are described in the following algorithm.

#### Algorithm 4. Authentication signal extraction and verification.

**Input:** the secret key  $K$  and the hash function  $f$  used in Algorithm 3, and a protected XPS document  $D'$ .

**Output:** an authenticated XPS document  $D''$ .

**Steps:**

1. Use the secret key  $K$  as an input to the hash function  $f$  to generate a 64-bit digest  $K'$ .
2. Choose an *unauthenticated* text segment  $T$  in the protected XPS document  $D'$  (with all text segments in  $D'$  regarded as unauthenticated initially).
3. Use  $T$  as an input to the hash function  $f$  to generate a 64-bit digest  $T'$ .
4. Perform the exclusive-OR operation  $\oplus$  to  $K'$  and  $T'$  to get a 64-bit string  $T'' = K' \oplus T'$ , called *computed authentication signal*.
5. Extract the gradient patterns  $p_1, p_2, \dots, p_{16}$  which were presumably superimposed onto the text segment  $T$ .
6. Map the gradient patterns  $p_1, p_2, \dots, p_{16}$  to the corresponding 4-bit segments  $t_1, t_2, \dots, t_{16}$  according to Table 2.
7. Concatenate  $t_1, t_2, \dots, t_{16}$  into a 64-bit string  $S$ , called *extracted authentication signal*.

8. Compare the extracted authentication signal  $S$  with the computed authentication signal  $T''$ : if  $S = T''$ , then regard  $T$  as authentic; otherwise, as modified and highlight  $T$  in the display of document  $D'$ .
9. Repeat Steps 2 through 8 until all text segments in  $D'$  are authenticated.
10. Take the resulting  $D'$  as an authenticated XPS document  $D''$  with highlighted text segments if there is any.

#### C. Experimental results

An example of the experimental results of XPS document authentication is shown in Fig. 6. Fig. 6(a) shows the input XPS document. After the authentication signals were generated and embedded in it by Algorithm 3, the resulting document looks all the same and is not shown here. Fig. 6(b) shows a tampering result which includes some fake words. Fig. 6(c) shows the authentication result after Algorithm 4 was applied to Fig. 6(b). All the fake words were detected correctly and marked in red. The experimental results show that the proposed method is feasible for protection of XPS documents.

#### D. Security considerations and enhancements

Using the proposed authentication method, the text content of an XPS document can be protected: as long as either the text content or the authentication signals are modified, we can detect the modification and point out which part of the text is suspicious. In addition, because we involve a secret key in the process of generating the authentication signals, it is hard for a malicious user to forge the authentication signals even when he/she knows the proposed algorithm.

However, an obvious leak in security here is that the malicious user may create a fake text segment in the stego-document by replacing both a text segment and its corresponding authentication signals with the fake text segment and the corresponding authentication signals computed from the same XPS document. To solve this problem, we may use an additional secret key to randomize the positions where the authentication signals are superimposed; only when a user has this secret key can he/she find where the authentication signal corresponding to any text segment is embedded in the stego-document. Furthermore, we may use a third key to randomize the content of the gradient pattern encoding table (Table 2) so that without the key correct gradient patterns cannot be generated before being superimposed on text segments, thus preventing a malicious user from creating fake authentication signals to cheat.

#### IV. DATA HIDING BY WIDTH-ADJUSTABLE INVISIBLE ASCII CODES IN XPS DOCUMENTS

The third proposed data hiding method utilizes texts in XPS documents as a cover channel. Specifically, when a <Glyphs> element in the XPS specification is used to create a text segment, its *UnicodeString* attribute contains the string of text rendered by the <Glyphs> element, and its

Indices attribute specifies a series of glyph indices and their attributes used for rendering the glyphs in the text segment. The simplest syntax to represent a glyph is:

*GlyphIndex, AdvanceWidth;*



(a)



(b)



(c)

Figure 6. Authentication of XPS documents. (a) Original document. (b) A tampered version of (a) with some words modified. (c) Authenticated result with the fake texts detected and marked in red.

where the *GlyphIndex* entry, being optional, is the index of the glyph in the font and the *AdvanceWidth* entry indicates the placement for the subsequent glyph, relative to the origin of the current glyph.

Based on the above properties, we can insert some ASCII space or control codes between words and specify the advance width to be *zero* so that these codes become invisible and do not occupy any space in the display of an XPS document. Such codes are said to be *null*. Fig. 7 shows an example hiding such *null codes* into a text segment. By experiments conducted in this study, it was found that only four ASCII codes, 09, 0A, 0D, and 20, are acceptable in the UnicodeString attribute without generating errors. These four ASCII codes are used to encode message bits according to a table as shown in Table 3, which we call *ASCII code encoding table*.

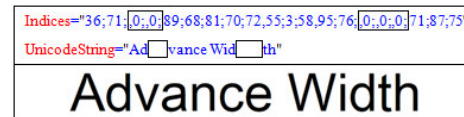


Figure 7. An example of hiding white spaces by adjusting advance width.

Table3. ASCII code encoding table.

Hex	Description	Binary code
09	Horizontal tab	00
0A	Line feed	01
0D	Carriage return	10
20	Space	11

### A. Embedding message data in a cover document by ASCII code encoding

In the third proposed data hiding method, first the input secret message in binary form is randomized by using a secret key and separated into bit pairs, which are then transformed according to the ASCII code encoding table (Table 3) into corresponding codes of 09, 0A, 0D, and 20. These ACSII codes finally are inserted between the words in the UnicodeString attributes of the text segments in the cover document, with their advance widths in the Indices attribute being set to be *zero*. This results in a stego-XPS document file with the secret message being embedded. The stego-document, when displayed, shows no difference in appearance from the original cover document, thus arousing no notice from people.

#### Algorithm 5. Data embedding by ASCII code encoding.

**Input:** A secret message *S* in binary form, a cover XPS document *D*, and a secret key *K*.

**Output:** A stego-document *D'*.

**Steps:**

1. Count the number  $l$  of bits in the message  $S$  and the number  $n$  of text segments in the cover document  $D$ .
2. Compute the value  $m = \lceil l/2n \rceil$  as the number of bit pairs to be embedded into *each* text segment in  $D$ .
3. Use the secret key  $K$  as a seed to generate a sequence  $Q$  of random numbers.
4. Randomize the bits of  $S$  with  $Q$  to get a randomized bit string  $S'$ .
5. Separate  $S'$  into pairs of bits,  $s_1, s_2, \dots, s_k$ , each being one of 00, 01, 10, and 11, where  $k = l/2$ .
6. Encode each bit pair  $s_i$  according to Table 3 to get a corresponding ASCII code  $c_i$ , which is one of 09, 0A, 0D, and 20.
7. Insert  $m$  ASCII codes between the words in the UnicodeString attribute of each text segment in  $D$  if the number of between-word spaces is larger than  $m$ ; otherwise, insert the excessive bit pairs at the end of the UnicodeString attribute.
8. Insert a parameter “;0;” into the corresponding position of each inserted ASCII code in the Indices attribute, with the parameter meaning that the advance width is zero and the glyph index is not specified.
9. Repeat Steps 7 and 8 until all ASCII codes  $c_1$  through  $c_k$  are embedded.
10. Recompress  $D$  with the modified text segments to get a stego-document  $D'$  as output.

#### B. Extracting data from a stego-document

The message data extraction process is essentially a reverse version of the embedding process, and is described in the following.

#### Algorithm 6. Data extraction process.

**Input:** a stego-XPS document  $D'$  and the secret key  $K$  used in Algorithm 5.

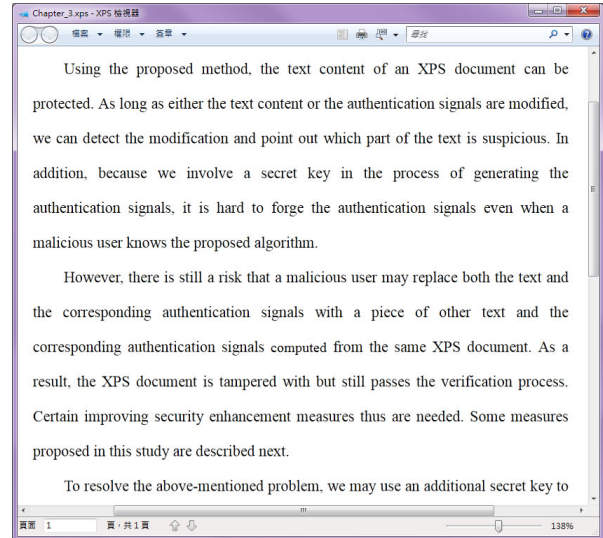
**Output:** a secret message  $S$ .

#### Steps:

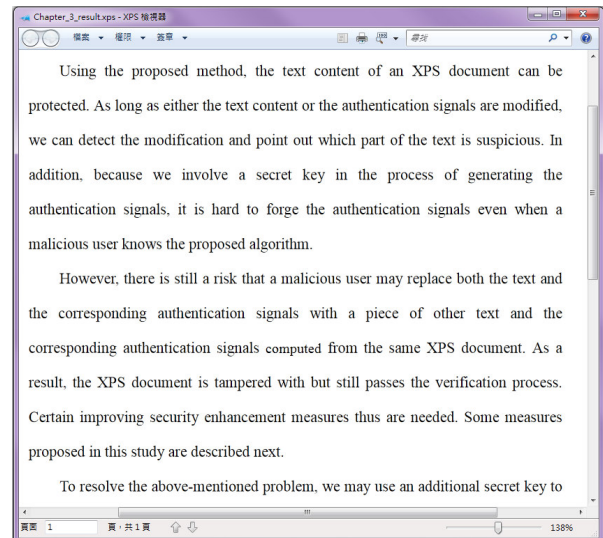
1. Extract a sequence of ASCII codes  $c_1, c_2, \dots, c_k$ , (one of 09, 0A, 0D, and 20) from each text segment in the document  $D'$  where the advance width of each  $c_i$  is zero.
2. Transform  $c_1, c_2, \dots, c_k$ , into a sequence of corresponding bit pairs  $s_1, s_2, \dots, s_k$  (one of 00, 01, 10, and 11) according to Table 3.
3. Concatenate  $s_1, s_2, \dots, s_k$  into a string  $S'$ .
4. Use the secret key  $K$  as a seed to generate a sequence  $Q$  of random numbers.
5. Use  $Q$  to reorder the bits of  $S'$  to get the desired secret message  $S$  as output.

#### C. Experimental results

Fig. 8 shows an example of the experimental results of applying Algorithm 5 to hide a message, where Fig. 8(a) shows the original cover XPS document, and Fig. 8(b) shows the resulting stego-image, which looks identical to Fig. 8(a). Fig. 9 shows the message extraction results carried out by Algorithm 6. The experimental results show the feasibility of the proposed method.



(a)



(b)

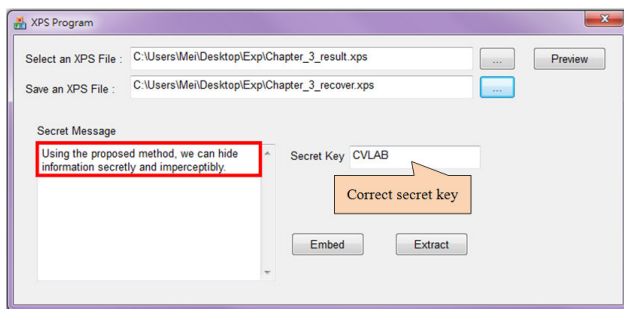
Figure 8. Data hiding by ASCII code encoding. (a) Original XPS document. (b) Stego-document.

#### D. Security considerations and enhancements

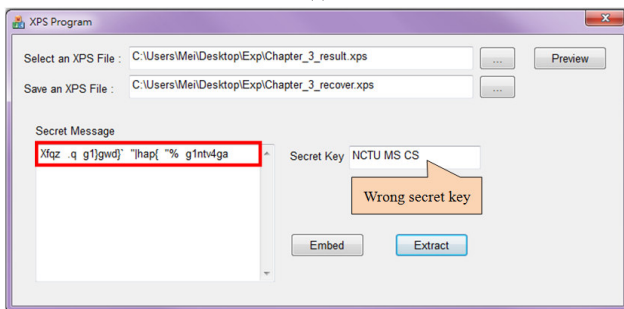
The secret key used in the proposed method described above is assigned by a user. Thus, even when a malicious user knows the proposed algorithm, the secret message cannot be extracted successfully without the key. However, a weakness here is that a malicious user may disturb the secret message by inserting or replacing some ASCII codes embedded in a stego-document. As a result, the extracted message, even with a correct key, will be erroneous.

To prevent this situation, we may duplicate the embedded codes multiply and determine the positions where these additional codes are embedded by the secret key as well. Then, when extracting the secret message, we can compare all the extracted duplicated ASCII codes and determine the correct message by voting. In this way, even part of the

secret message has been modified, we still can extract the message correctly. Moreover, we may use additionally the document authentication method (Algorithms 3 and 4) to protect not only the contents of XPS document but also the contents of the secret message. More specifically, we may transform the secret message into corresponding gradient patterns as authentication signals and superimpose them on the XPS document. The positions where these gradient patterns are superimposed can be determined by the user key. As a result, we may decide whether the secret message has been tampered with or not by the authentication signal verification process described by Algorithm 4.



(a)



(b)

Figure 9. Data extraction from stego-document. (a) Extraction result using right key. (b) Extraction result using wrong key.

## V. CONCLUSION

In this study, we investigate the new problem of information hiding via XPS documents and proposed three new data hiding techniques for different applications. First, a data hiding method for covert communication based on the

novel use of a division pattern encoding table to encode the secret message has been proposed. Also, to verify the integrity and fidelity of the text contents of XPS documents, an authentication method based on a data hiding technique has been proposed, which generates variable gradient patterns as authentication signals and superimpose them onto an XPS document invisibly. Finally, a data hiding method using width-adjustable invisible ASCII codes in the XPS document has been proposed for the application of steganography, in which a secret message is encoded by certain invisible ASCII codes found in this study and embedded into between-word spaces by adjusting their advance widths to be zero. Moreover, measures to enhance the security of each proposed method have also been suggested. The experimental results show the feasibility of the proposed methods and the appearance of the XPS document is totally unaffected after the proposed methods have been applied. Future studies may be directed to investigating more features of the XPS document specification and designing accordingly more data hiding methods for information hiding applications.

## REFERENCES

- [1] T. Y. Liu and 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*, vol. 2, no. 1, pp. 24-30, March 2007.
- [2] S. Inoue, K. Makino, I. Murase, O. Takizawa, T. Matsumoto, and H. Nakagawa, "Proposal on information hiding method using XML," *Proceedings of 1st NLP and XML Workshop*, Tokyo, Japan, Nov. 2001.
- [3] S. Zhong, X. Cheng, and T. Chen, "Information steganography algorithm based on PDF documents," *Computer. Engineering*, vol. 32, no. 3, pp. 161-163, Feb. 2006.
- [4] C. T. Wang and W. H. Tsai, "Data hiding in PDF files and applications by imperceptible modifications of PDF object parameters," *Proceedings of 2008 Conference on Computer Vision, Graphics and Image Processing*, Ilan, Taiwan, Aug. 2008.
- [5] Microsoft Co., *XML Paper Specification*, Version 1.0, Oct. 2006.