Journal of Information Security and Applications 61 (2021) 102875

Contents lists available at ScienceDirect

Journal of Information Security and Applications

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

StealthPDF: Data hiding method for PDF file with no visual degradation

Minoru Kuribayashi [a], KokSheik Wong [b,*]

[a] Graduate School of Natural Science and Technology, Okayama University, Okayama, Japan
[b] School of Information Technology, Monash University Malaysia, Jalan Lagoon Selatan, 47500 Bandar Sunway, Selangor Darul Ehsan, Malaysia

ARTICLE INFO

Keywords:
Data hiding
StealthPDF
Complete quality preservation
Authentication
Space
PDF

ABSTRACT

Conventional data hiding methods for PDF file insert a payload data by slightly modifying the position of characters in a document. Even if the changes are small, a certain degree of visual distortion is inevitably introduced to the PDF file. In this work, we propose a new data hiding method that splits the space value between characters. Specifically, a space value is split into two or more related values. Except for the first value which is reserved to store the corrective data, each of the related values encodes a segment of the payload data. When the PDF file is opened by a PDF viewer, the visual appearance is exactly the same as its original counterpart, i.e., complete quality preservation. To prevent direct observation of PDF file, access control is introduced by setting an owner password, which is a built-in function in the PDF standard. In the best case scenario, 38,160 bits can be hidden, while the observed file size increase is 12,776 Bytes.

1. Introduction

The portable document format (PDF) [1] was developed by Adobe Systems Society to serve as a page description language with the aim to preserve the formatting of a file. PDF is platform independent and it has then been widely adopted in a broad spectrum of applications. It can contain interactive elements such as buttons for form entries and audio/video clips. PDF is optimized for Web rendering, where texts are displayed first followed by image/hypertext links. PDF is now the de facto standard for digital document, and in fact many documents appear in this form, including research articles, government/official documents, credit card statements, medical report, tickets, to name a few. It is expected that more PDF files will be in circulation thanks to initiatives such as paperless office. In fact, anyone can generate a PDF document by using free online and offline resources such G. Suite and OpenOffice.

Similar to other digital contents, PDF suffers from illegal distribution and manipulation. According to [2], two thirds of all published scientific articles are illegally shared online at no cost, and these articles are usually stored in the form of PDF. This is a serious threat to the academic publishers' business model. In addition, many leaked documents are in the form of PDF. For example, within the Panama Papers [3], 2.2 millions of the leaked documents are in the form of PDF, while email tops the chart with 4.8 millions leaks. In addition, PDF can also be forged or fabricated, hence putting the integrity of a PDF file in question.

Data hiding, which essentially inserts some payload data into a host content, can serve as a solution to the aforementioned problems. For example, data such as workstation ID, IP address, user information, etc. can be hidden into the PDF file during its production or compilation. The inserted data can then be extracted for various purposes, for example, to claim ownership over a PDF file (watermark), to trace illegal distribution of classified document (fingerprint), to check whether the document is genuinely coming from another person (authentication), etc. [4,5]. While the aforementioned auxiliary data can be hidden into the images (including vector graphics) or other contents within a PDF document by using the conventional data hiding techniques, it is rather common to have text-only PDF file. Therefore, in this work, our discussions and the proposed method focus on hiding data into a PDF file by manipulating texts (including spaces) within a PDF file.

Specifically, Zhong et al. [6] proposed to manipulate the preset distance between two consecutive words in a PDF file to hide data. Similarly, Lee et al. [7] hide a watermark into a PDF file by exploiting a pair of characters, namely, "0x20" and "0xA0", which are both displayed as blank characters in a PDF file. Furthermore, Lin et al. [8] proposed a method to hide data into a PDF file which is compliant to ISO-8859. It combines a data hiding method for PDF file and an encryption method based on quadratic residue. In addition, techniques for hiding data into spaces between text strings have also been adapted to hide data into PDF file [9–14]. Although low distortion is accomplished by these conventional space-based data hiding techniques, it causes irregularities in the space values as discussed in [14]. To eliminate visual distortion, Iwamoto et al. [15] divide the object in each line of the PDF file into two smaller objects according to the data bit to be

* Corresponding author.
E-mail addresses: kminoru@okayama-u.ac.jp (M. Kuribayashi), wong.koksheik@monash.edu (K. Wong).

https://doi.org/10.1016/j.jisa.2021.102875

Available online 10 June 2021

hidden. However, the payload is small while the increase in file size is large. In addition, it is not difficult to dictate the existence of hidden data in the processed PDF file if the descriptions of the document are observed.

Although various methods are put forward to hide data into PDF file, they suffer from either low payload and/or inevitable visual distortion. Therefore, in this paper, we propose a new data hiding method for PDF file that never causes visual distortion in the processed PDF file. The main idea is to split each space value into a combination of two or more related values. Specifically, a blank descriptor is inserted between the split numbers in order to maintain the original space length. The splitting behavior changes depending on the payload data to be hidden. The secrecy of the hidden data is managed by setting an owner password, which is regarded as a secret key in the proposed method. Our work makes the following contributions: (a) complete visual quality preservation of the PDF file after data hiding, (b) scalable payload, where more data can be accommodated at the expense of file size increment, and; (c) reversible, where the original PDF can be completely restored. One of the potential applications of the proposed method is for the purpose of authentication. When the processed PDF reaches the rightful viewers (i.e., who possess the key to view), each viewer can extract the hidden data and verify whether the document is genuine. This feature is useful when dealing with important documents, for example those from the government, official receipt, certificate, etc. to combat forgery.

The rest of this paper is organized as follows: In Section 2, we review the specification of the PDF file structure and summarize selected conventional data hiding methods which are related to our proposed method. The proposed method is detailed in Section 3, and experimental results are presented in Section 4. Finally, conclusions to this study are drawn in Section 5, followed by some suggestions for future work.

2. Preliminaries

2.1. PDF file structure

The Portable Document Format (PDF) was initially released in 1993 and it has then been standardized as a digital format for representing document in ISO 32000 [1] in 2008. PDF was developed by Adobe Systems to enable users to exchange and view electronic documents easily and reliably, irregardless of the environment in which they were created as well as the platform on which they are viewed. Fig. 1 illustrates the structure of PDF file. In particular, a PDF file consists of four principle components, namely, *Header*, *Body*, *Cross-Reference Table (CRT)*, and *Trailer*.

The *Header* is the first line of the PDF file, and it indicates the version of the PDF specification. Next, *Body* holds all the data in the document, which will be rendered by a PDF viewer. It consists of a sequence of objects representing the contents of a document. These objects include various components of the document, for example, fonts, pages, images, and other multimedia elements, to name a few. Specifically, the stream object is the sequence of bytes that make up an assemble of data including texts and their positions, which are encapsulated between the keywords "stream" and "endstream". It is noteworthy that a PDF document can be encrypted to protect its contents from unauthorized access, where any strings within the streams are encrypted and hence the entire stream is encrypted.

Within a stream, a text object begins with the 'BT' operator, implying "Begin a Text object", and ends with the 'ET' operator, implying "End a Text object". Then, within the text objects, some operators are defined to represent the state and position of texts. Among these operators, the 'Tf' operator specifies the text style and font size, while the 'Td' operator specifies the space offset of the beginning of the current line. Two operators are required to show texts on a page, namely, the 'Tj' and 'TJ' operators. In particular, the 'Tj' operator only shows a text string (e.g., ABC), while the 'TJ' operator shows one or more text strings (e.g., Twinkle twinkle little star), allowing individual glyph positioning. Namely, the TJ operator shows the characters and spaces between characters.

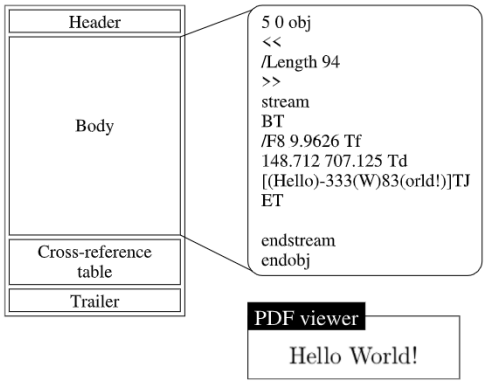

Fig. 1. Structure of PDF file.

For each TJ operator, a collection of space values are specified for the text characters. In Fig. 1, the syntax "`[(Hello)-333(W)83(orld!)]TJ`" represents the text characters "Hello World!", along with two space values −333 and 83. The characters for rendering on the screen are enclosed between parenthesis "`()`", which work as a deliminator between the actual characters and space value. When a variable-length byte-encoded representation of Unicode is used, the angled bracket "`<>`" is utilized as the deliminator for each character (e.g., `<0041>`). Referring again to "`[(Hello)-333(W)83(orld!)]TJ`", the former space value (i.e., -333) dictates the space between the words, while the latter (i.e., 83) dictates the space between the characters "W" and "o". In other words, the space value dictates the horizontal position of the characters. To reduce the file size, the sequence of bytes in the stream object can be compressed by using zlib/deflate compression method in general.

Next, *CRT* contains the references to all the objects in the document. It allows random access to objects in the file, where each object is represented by one entry in the table, and it contains information about the indirect objects in the file. Last but not least, *Trailer* gives the location of the *CRT* and certain special objects within the *Body* of the file.

In the event the contents of a PDF file are updated incrementally, the changes can be appended to the end of the file with the possibility of adding new *CRTs*. The resulting PDF then consists of multiple components. Specifically, after the original *Body*, *CRT*, and *Trailer*, the updated part of these components are appended, where the updated *CRT* contains entries only for objects that have been modified, replaced, or deleted. The updated *CRT* indicates the offset (from the beginning of the PDF file) to the updated/new objects and overrides the old offset so that the most recent version for each object is recognized by a PDF viewer. Although it is convenient to append the updated part containing a hidden data to the end of the PDF file, information about the history of modification is left in the updated PDF file, essentially leaving the evidence of the hidden data in the open. From the attacker's point of view, the existence of hidden data can be checked easily by referring to such information. Hence, instead of the appending operation, the direct modification of both *Body* and *CRT* is commonly performed to secretly insert data into a PDF file. Our work will also take this approach.

It is noteworthy that we only presented the essential components and functionality of PDF, which are sufficient for us to detail our proposed data hiding method. Readers who are interested in the detailed information about PDF may refer to the standardization document [1].

2.2. Conventional data hiding methods in PDF

Due to the redundancy in *Body* of a PDF file, some methods exploits its description to hide data. Among the operators in *Body*, the space values enclosed within the 'TJ' operator are commonly utilized to hide data. Once the *Body* is modified for data hiding, it violates the constraints in the PDF document, i.e., the descriptions in *CRT*. When a PDF viewer detects any irregularities in the format, it refuses to open the file. Therefore, *CRT* needs to be rectified.

The conventional methods focus on the trade-off between the payload (number of bits that can be hidden) and the distortion caused by the hiding operations. Specifically, the space values are extracted from each line of texts and they are gathered into a vector, which is in turn treated as a 1-dimensional signal for further processing. For example, in Bitar et al.'s method [10], the quantization index modulation (QIM) [16] with spread transform dither modulation (STDM) is employed. A host signal is created from some x-coordinate (horizontal position) values associated to the 'Td' operator in a PDF file. Here, each payload bit is hidden into L samples extracted from the values in the 'Td' operator, where each payload bit is encoded into a codeword of length L by using a spread sequence. It achieves transparency and robustness due to its spreading effect, but the payload is small because it requires $p \times L$ samples to embed p-bit of payload. However, Hatoum et al. [11,12] then pointed out the vulnerability of the STDM QIM method, and improved the operation to resist against the principal component analysis and independent component analysis. On the other hand, Kuribayashi et al. [13] regard a collection of space values as a host vector, then transform them into the frequency domain, and hide data into the frequency components by using Dither Modulated (DM) QIM method. They keep the DC component unmodified so that the total length of each line is preserved. In the above dither modulation techniques, pseudo-random numbers are generated to randomize the quantized values according to a secret key. Due to the characteristics of dither modulation, it is difficult to analyze the hidden data without a secret key. In addition, the distortion induced by the hiding operation is widely spread over several space values, and hence, the degradation of visual quality is less noticeable.

As summarized above, most conventional methods aim to hide data without severely degrading the visual quality of processed PDF file. However, from the steganographic point of view, it is not difficult to find the existence of a hidden data within a given PDF file. In fact, as reported in [14], the statistical distributions of the space values are changed after executing the hiding operations, and these changes can be detected. Even if the visual distortion can be suppressed by controlling the payload and the quantization step size, visual distortion is inevitable and hence the secrecy of the hidden data cannot be ensured in the conventional methods.

In addition to the aforementioned methods, the description in text string has also been exploited to hide data into PDF [15]. Suppose that, on each line, each text string is represented by a single 'TJ' operator. The vertical position for each line is specified by a 'Td' operator. If the text string is split into two parts, the horizontal position of the second one is shifted to the right according to the horizontal size of the first one to maintain their relative position when rendered by a PDF viewer. To hide data, the splitting of text strings represented by a single 'TJ' operator on each line depends on the payload bit to be hidden, where the splitting position is specified by a secret key. Although this method does not lead to any visual distortion, the payload is small and the increase of file size becomes a problem. In addition, once the document of PDF file is observed by an text editor, it is easy to determine the existence of hidden data.

3. Proposed method

In this section, we put forward a new data hiding method for PDF file. Different from the conventional methods, the proposed method never changes the visual appearance, and the payload is scalable. In addition, the secrecy of the hidden data is managed by a simple access control mechanism defined in the PDF standard.

3.1. Data insertion

Given a predefined integer value k, the payload μ in binary representation is first divided into k-bit segments $m_i^\star$, so that $\mu = m_1^\star \parallel m_2^\star \parallel \ldots \parallel m_\tau^\star$, where '$\parallel$' refers to the concatenation operation, and $\tau = \lceil |\mu|/k \rceil$ for $|Z|$ being the cardinality of Z and $\lceil Z \rceil$ being the smallest integer greater than or equals to Z. Then, for each space value x, we hide $t-1$ segments by splitting the space value x into t values, where $t \geq 2$.

3.1.1. Basic method

Let x_j be the jth space value for $j \geq 1$,. The following vector is then formed:

$$\boldsymbol{m}^{(j)} = (m_{t_j+1}, m_{t_j+2}, \ldots, m_{t_j+t-1}), \tag{1}$$

where m_i is the decimal equivalent of $m_i^\star$, and $t_j = (j-1)\times t$. The space value x_j is then decomposed as follows:

$$x_j = x'_j + \sum_{i=t_j+1}^{t_j+t-1} m_i. \tag{2}$$

It follows that

$$x'_j = x_j - \sum_{i=t_j+1}^{t_j+t-1} m_i. \tag{3}$$

One may imagine that the value x'_j 'absorbs' the difference between the sum of m_i and x_j, which is needed for compensating the space value as well as reversibility purposes. In the case of $t = 2$, x_j is represented by two values m_{2j-1} and x'_j. Then, given a 'TJ' operator, the space value x_j is represented as "m_{2j-1} () x'_j". It is important to note that this change in representation does not lead to any visual distortion. For example, suppose that an original space value of '200' appears as:

```
[(He)200(llo)]TJ.
```

If $t = 2$ and $m_1 = 135$, then the space value is split into two values:

```
[(He)135()65(llo)]TJ.
```

If $t = 3$, then two payload segments can be inserted. For example, suppose $m_1 = 32$ and $m_2 = 112$, then

```
[(He)32()112()-56(llo)]TJ.
```

After performing the aforementioned hiding operations, no visual distortion will occur when the processed PDF file is displayed by using any PDF viewers. In addition, the payload of the hidden data can be controlled by manipulating the parameters k and t. If the total number of space values within the PDF file is N, then the payload is $k \times t \times N$ bits. Hence, the payload becomes higher when k is increased. However, the drawback of using a larger k is the increase of file size because the parenthesis "()", which indicates a blank character with no length, is inserted between the space values.

Similar to the conventional methods, when the space values enclosed within the 'TJ' operator are modified to hide data, some PDF viewers refuse to open the modified PDF file because the references to the *CRT* are incorrect. As mentioned in Section 2.1, the contents of a PDF file can be updated incrementally and the changes can be appended to the end of file. However, if one implements such an approach, it is easy to study the history of changes, which may suggest the existence of some hidden data. Hence, the *CRT* must be revised according to the changes of space values as a post-processing.

In the proposed method, we split a space value x_j into t decimal numbers, namely, m_i for $i = t_{j+1}, \ldots, t_{j+t-1}$ and x'_j. It leads to an increase of characters in a PDF file for inserting the payload data because each digit of m_i is represented by one character. For example, in case of $k = 4$, the decimal equivalent of m_i ranges from 0 to 15. Among the 16 decimal numbers, 10 of them (viz., $0, 1, \ldots, 9$) are represented by

Table 1
Increase of digits with respect to k when $t = 2$.

k	Number of characters N_c for m_i	Efficiency E
1	$\frac{2}{2} = 1$	3.0000
2	$\frac{4}{4} = 1$	1.5000
3	$\frac{8}{8} = 1$	1.0000
4	$\frac{10}{16} + 2 \times \frac{6}{16} = \frac{11}{8}$=1.375	0.8438
5	$\frac{10}{32} + 2 \times \frac{22}{32} = \frac{27}{16} \approx 1.688$	0.7375
6	$\frac{10}{64} + 2 \times \frac{54}{64} = \frac{59}{32} \approx 1.844$	0.6406
7	$\frac{10}{128} + 2 \times \frac{90}{128} + 3 \times \frac{28}{128} = \frac{137}{64} \approx 2.141$	0.5915
8	$\frac{10}{256} + 2 \times \frac{90}{256} + 3 \times \frac{156}{256} = \frac{329}{128} \approx 2.570$	0.5713
9	$\frac{10}{512} + 2 \times \frac{90}{512} + 3 \times \frac{412}{512} = \frac{713}{256} \approx 2.785$	0.5317
10	$\frac{10}{1024} + 2 \times \frac{90}{1024} + 3 \times \frac{900}{1024} + 4 \times \frac{24}{1024} \approx 2.916$	0.4916

1 character, while the others (viz., $10, 11, \ldots, 15$) are represented by 2 character. On average, the expected number of characters N_c to represent the decimal equivalent of m_i is 1.375 ($= 10/16 + 2 \times 6/16$). Table 1 records N_c for various k values and the corresponding efficiency E. The efficiency is regarded as the expected number of characters required to embed 1 bit data, and it is defined as

$$E = \frac{N_c + 2}{k}. \tag{4}$$

Since two characters are appended to represent the empty word '()', $N_c + 2$ characters are required to insert the k-bit payload segment m_i, and hence, the efficiency is calculated by Eq. (4). The table indicates that the larger k is, the higher the efficiency of an uncompressed PDF file becomes.

3.1.2. Improved method

A hurdle faced by our proposed method is the file size increment due to data hiding. This is an unavoidable problem because we are injecting additional characters into the host PDF file. Therefore, we aim to suppress this increment as much as possible. In this section, we introduce some restrictions on the number of digits that can be added/subtracted from the original space value by controlling the length of the payload segment to be hidden, which in turns control the file size increment. Due to the standardization of format for PDF, the parenthesis "()" is represented as two characters and each digit of the number m_i (decimal equivalent) is also represented as one character when a PDF file is not compressed. As such, when $t = 2$, at least 3 characters are added to embed m_i. In the basic method, the change on the number of digits of x_j is not considered. From the compression efficiency point of view, we should suppress the increase of the newly introduced characters, namely, the increase of digits in x'_j.

Suppose that $t = 2$ and x_j is a positive number represented by 3 digits. When a data segment m_i of length k is selected from the range $[0, 2^k - 1]$, $x'_j = x_j - m_i$ is still of 3 digits when the value of x_j is in the range of $[99 + 2^k, 999]$. For a negative integer, its negative sign must be added and it is represented by one character. Hence, if the range of x_j is $[-1000 + 2^k, -100]$, the number of characters for representing x'_j is not changed from that of x_j, namely "$-***$". Table 2 summarizes the conditions imposed on x_j in the case of $t = 2$ such that the data embedding process does not change the number of digits in x'_j.

According to the statistical distribution of space values x_j in the target PDF file, two candidates for the range of x_j are selected for a given k. For instance, when $k = 8$, the ranges $[99 + 2^k, 999]$ and $[-1000+2^k, -100]$ are selected. Then, x_j is represented by 3 characters if $x_j \geq 0$; otherwise, it is presented by 4 characters including the negative sign "$-$". Under such a condition, the number of characters to represent x'_j is equal to that of x_j. Thus, in the improved method, we perform the data insertion only when x_j is within the selected ranges so that we do not to increase the number of characters for x'_j.

Table 2
Conditions on the range of x_j for maintaining the same number of digits.

Sign of x_j	Range of x_j	k	Digits
Positive	$[-1 + 2^k, 9]$	≤ 3	1
	$[9 + 2^k, 99]$	≤ 6	2
	$[99 + 2^k, 999]$	≤ 9	3
	$[999 + 2^k, 9999]$	≤ 12	4
Negative	$[-10 + 2^k, -1]$	≤ 3	2
	$[-100 + 2^k, -10]$	≤ 6	3
	$[-1000 + 2^k, -100]$	≤ 9	4
	$[-10000 + 2^k, -1000]$	≤ 12	5

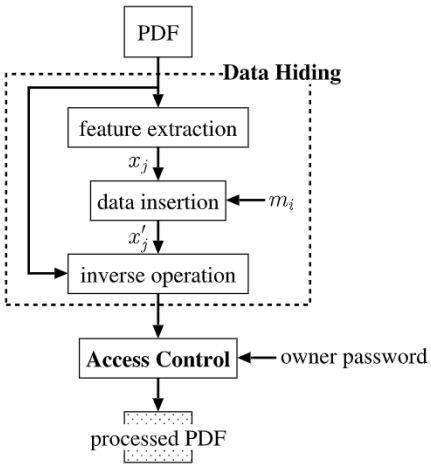

Fig. 2. Overview of the operations.

3.2. Extraction

To extract the hidden data, for each line of texts, the 'TJ' operator is located. After skipping the first space value (i.e., x'_j), every occurrence of the pattern '$()I$' is recorded for I being a positive integer less than $2^k - 1$. Here, the regular expression $()I^+$ can be used. Each decimal value m_i is then retrieved and converted back to its binary equivalent $m_i^\star$. However, addition leading zeros are concatenated to $m_i^\star$ so that the resulting segment is of length k-bit. For example, if $m_i = 7$, then $m_i^\star = 0111$ and 000111 for $k = 4$ and 6, respectively. Therefore, k is required during extraction.

The proposed data hiding method is reversible because the original PDF file can be reconstructed. Specifically, during the extraction of the hidden data, the original space values can be calculated and the original layout within each 'TJ' operator can be reconstructed.

3.3. Access control

Although we can hide data into a PDF file without causing any visual distortion by using the proposed method, it is easy for an attacker to determine the existence of the hidden data by directly observing the space values enclosed by the 'TJ' operator. In order to keep the hidden data concealed, we exploit the permission feature in the PDF file by setting an owner password. Fig. 2 illustrates an overview of the operations in the proposed method.

According to the specification of PDF [1], a standard security handler allows up to two passwords for a document, namely, an *user* password and an *owner* password. If any password is set, the PDF

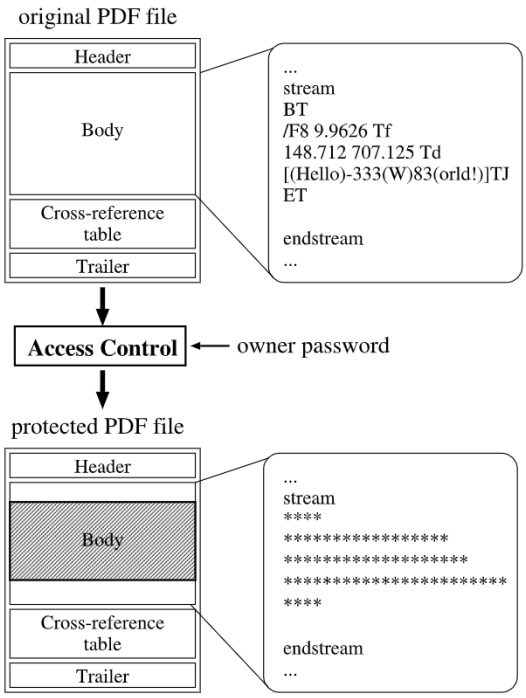

Fig. 3. Protection of PDF file with owner password. The shaded part is encrypted.

document is encrypted. Then, the specific permissions and the information required to validate the passwords are stored in an encryption dictionary of the security handler. Specifically, the user password is utilized to prevent anyone from opening the document for viewing. Once it is set by the owner, the PDF viewer will prompt for the user password whenever one attempts to open the protected document for viewing. In addition to the viewing restriction, the owner password can be utilized to set certain access permissions/restrictions such as printing, copying, adding annotations, editing, etc. Note that by opening the document with the owner password, it allows the user to have full access to the document. On the other hand, the user password is used only for the authentication process to open the protected PDF file (i.e., viewing). When the access permissions/restrictions are specified, any modification to the protected PDF is prohibited without the owner password. In case someone attempts to open an encrypted document which is protected by an user password, the PDF viewer first tries to authenticate the encrypted document by requesting for the user password. Even if "no viewing" restriction is set by the owner, the stream object, which is the sequence of bytes placed between "stream" and "endstream", is encrypted and the flag information is protected by the owner password. An illustration of the operation is shown in Fig. 3. Thus, a direct observation of text string within the stream object is prohibited. Even when a PDF viewer opens a PDF file which is protected by an owner password, it is difficult to observe the original strings in the stream objects within the Body by using an hex-editor.

There are many programs that support PDF restrictions by configuring a PDF owner password. For example, in Adobe Acrobat, the PDF owner password is called the *change permissions password*. It is also referred to as the PDF permissions password, restriction password, or PDF master password, depending on the PDF reader or writer. The access permissions are specified in the form of flags corresponding to various operations such as printing, changing the document, document assembly, content copying, content copying for accessibility, page and graphics extraction, commenting, filling of form fields, signing, creation of template pages and so on.

When the visual appearance of the PDF file is completely the same before and after data hiding, it is difficult to determine the existence of the hidden message. Thus, by using our proposed method, a sender can easily hides some data into a PDF file, and protects it by setting an owner password with "no viewing" restriction. If the owner password is shared, the intended receiver can extract the hidden data. In this case, the visual appearance is not changed by the hiding operations, and only the file size is increased in comparison to its original counterpart. Hence, the only hint to determine the existence of hidden data is the file size. If the variation of file size is small, not much can be inferred from the file size and hence the analysis for the existence of hidden data becomes difficult.

When the document is opened with the correct owner password, one will gain full-access to the protected document. Hence, the extraction of hidden data and restoration of the original PDF file can only be performed when the correct owner password and the parameter k are in possession.

3.4. Dependency on PDF converter

This section highlights the differences in a PDF file produced by different PDF converters, then justifies the limitation of the proposed hiding method when considering these differences. Since PDF has been established as the de facto standard for digital document format, many PDF converters are released for various kinds of programs. For the purposes of this work, we focus on the PDF generated from LaTex and Microsoft Word.

3.4.1. LaTeX

In LaTeX, a PDF file can be created by compiling a source file. One of the popular LaTeX compilers is the TeX Live,[1] which provides a comprehensive TeX system with binaries for Linux, macOS, and Windows operation system environments. For the purpose of reproducibility we consider a free cloud-based LaTeX environment, namely, OverLeaf.[2] On this platform, four compilers are available for selection, namely, pdfLaTeX, LaTeX, XeLaTeX, and LuaLaTeX.

Without loss of generality, consider the LaTeX source file as follows:

```
\documentclass{article}
\begin{document}
Hello World!
\end{document}
```

When the above source file is complied at OverLeaf (accessed on 10 March 2020), the followings are observed for each compiler available at OverLeaf. Here, we decompress the PDF file (which is compressed) to observe the characters and space values enclosed by the 'TJ' operator.

- pdfLaTeX

```
[(Hello)-333(W)83(orld!)]TJ
```

- LaTeX

```
[(H)0.199721(e)-0.460706(l)-0.335727(l)
-0.335727(o)-333.801(W)82.0632(o)
0.433749(r)-1.06477(l)-0.335727(d)
-0.575882(!)-0.335727]TJ
```

- XeLaTeX

```
[<003e0032004800480051>-333<0071>82
<005100600048002f0035>]TJ
```

[1] http://www.tug.org/texlive/.
[2] https://www.overleaf.com/.

- LuaLaTeX

```
[<003E0032004800480051>-333<0071>83
<005100600048002F0035>]TJ
```

It is apparent that different compiler produces different output. Specifically, for XeLaTeX and LuaLaTeX, the characters are represented by different encoding (i.e., word) and enclosed by angled brackets (i.e., "<>"), although the space values between characters are similar to that of pdfLaTeX. Therefore, the proposed data hiding method can be applied, where the space values can be split into some values by separating them with "()". It is interesting that a space value is given for every character in case of LaTeX, and the values are non-integer. It is also noteworthy that the space values are slightly different for different compiler. In any case, the proposed operation can be applied to hide data without causing any visual distortion.

The splitting of words and their space values are strongly dependent on the LaTeX compiler. Therefore, the texts appear slightly different in each case, although they are all generated from the same LaTeX source codes. In addition, the description of the aforementioned output PDF files are not unified because different compiler implements different algorithm, and hence the file sizes are also different. Although the proposed method changes the file size after hiding data and setting the permission, the natural fluctuation in file size due to the differences in LaTeX compilers adds more complexity to the analysis for the existence of the hidden data, hence favoring the hider.

3.4.2. Microsoft Word

There are many ways to convert a Microsoft Word file to PDF. Suppose the string "Hello World!" is written in a Word file (Version 16.0.4978.1000) using the "Century" font at 10.5pt and exported as a PDF file. The resulting PDF is analyzed and the following is observed:

```
[(H)3(el)-2(l)-2(o)] TJ
[(W)61(orld)-7(!)] TJ
```

When the Word file is uploaded to Google Docs and then downloaded directly as a PDF file, the text string is observed to be decomposed into each character and the 'Tj' operator is used in this case. The position of each character is specified by the 'Td' operator as follows:

```
113.390625 -146 Td <002B> Tj
10.1079559 0 Td <0048> Tj
6.9999695 0 Td <004F> Tj
3.891983 0 Td <004F> Tj
3.891983 0 Td <0052> Tj
7.7839661 0 Td <0003> Tj
3.4999847 0 Td <003A> Tj
12.4459534 0 Td <0052> Tj
7.7839661 0 Td <0055> Tj
5.4459839 0 Td <004F> Tj
3.891983 0 Td <0047> Tj
7.7839661 0 Td <0004> Tj
```

Note that each character is independently described by the 'Td' and 'Tj' operators. For such a situation, the proposed method cannot be applied. However, it should be noted that the use of a pair of 'Td' and 'Tj' operators to describe each character is very inefficient. In fact, the file size becomes significantly larger due to high redundancy in the descriptions in the PDF file. Thus, it is advisable to generate the PDF file by using other software for efficient data storage, as well as for data hiding purposes.

Next, the same Google Doc is downloaded as an OpenDocument (.odt) file. When the downloaded file is converted into PDF by using LibreOffice (Version 5.3.6.1[3]), the description in the 'TJ' operator is enclosed by angled bracket:

```
[<0102>-3<03>1<03>1<04>1<05>-6<06>55<04>1
<07>1<03>1<08>-3<09>]TJ
```

Similar to XeLaTeX and LuaLaTex, the space value can be split by using parentheses "()" without causing errors.

Last but not least, the same OpenDocument (.odt) file is opened by using Pages (Version 8.2.1) in Mac OS, and it is converted to PDF. It is found that the description enclosed by the 'TJ' operator is similar to that of pdfLaTeX, although the decomposition of text string is different. Therefore, it is possible to use the proposed method in this case.

Based on the aforementioned observations, we conclude that our proposed data hiding is sufficiently versatile because it is applicable to the PDF file generated by various converters. It is noteworthy that the structure of a PDF file is strongly dependent on the operations performed by the converter software. Specifically, the PDF document can be rendered by using different operators (i.e., Td, Tj, or TJ), and the frequency of space values (i.e., between every consecutive pair of characters vs one space value for a sequence of characters) can vary. Besides, there are additional factors which affect the file size. For example, the choice of font to represent characters also changes the description used by the operators and the space values. Furthermore, the attachment of specific fonts further changes the total file size. Moreover, the page style including the margins of a page also affects the description.

4. Experiments

The proposed data hiding method is implemented in C++ language and shell script in Cent OS linux (Version 7.7.1908) environment. The test PDF document is generated by Latex using TeX Live 2016. Specifically, the compiler is e-pTeX 3.14159265-p3.7-160201-2.6 (utf8.euc), and the PDF converter is the extended version of dvipdfm-0.13.2c. The document is the first 5 chapters of Genesis in the Old Testament of the Bible, which contains 3178 words with 15694 characters and 153 lines. The total number of space values in the file is $N = 3971$. The same text-based content is copied into a Word file and a PDF is generated by using Microsoft Word 2016 (Version 16.0.4978.1000). For all experiments, we insert as much data as possible (i.e., randomly generated bit stream) into the PDF by utilizing all space values in the document. Here, we use the "PDFtk" tool[4] to rectify the *CRT* and to decompress/compress the PDF file. The PDFtk tool also enables us to set the owner password to the PDF file with a simple command. Specifically, "StealthPDF" is set as the owner password by using PDFtk tool with no restriction, i.e., it allows all features of access control. It was set by using the following command in PDFtk:

```
$ pdftk input.pdf output output.pdf \
 owner_pw StealthPDF allow AllFeatures.
```

Even for such a setting, direct observation of the values in the 'TJ' operator is difficult because the sequence of bytes within the stream object is encrypted, although a PDF viewer can display the document without requesting for the password.

For the rest of the presentation, the term *processed PDF* refers to a PDF with hidden data, i.e., the product of the proposed method. In the next subsections, we discuss the influence of parameters t and k, file size increment due to setting password, influence of converter on the proposed method, as well as the relative performance between the proposed and conventional data hiding methods.

[3] https://www.libreoffice.org/.

[4] https://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/.

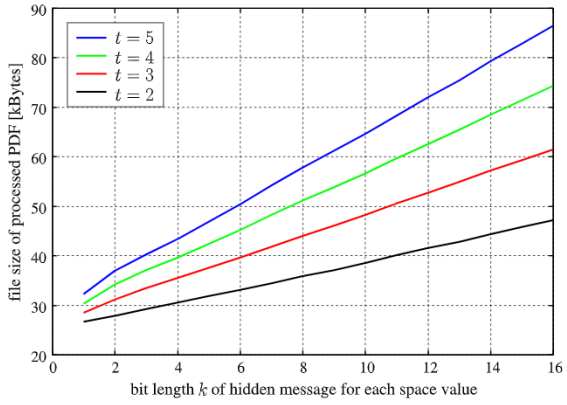

Fig. 4. File size of the processed PDF for different k and t, where the original size is 24767 Bytes.

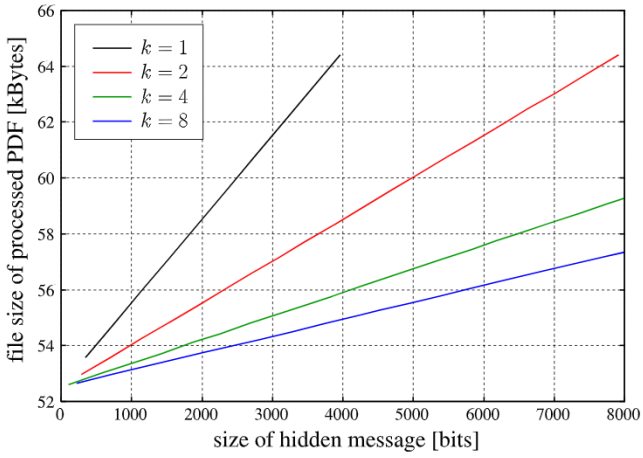

Fig. 5. File size of the processed PDF without compression when $t = 2$.

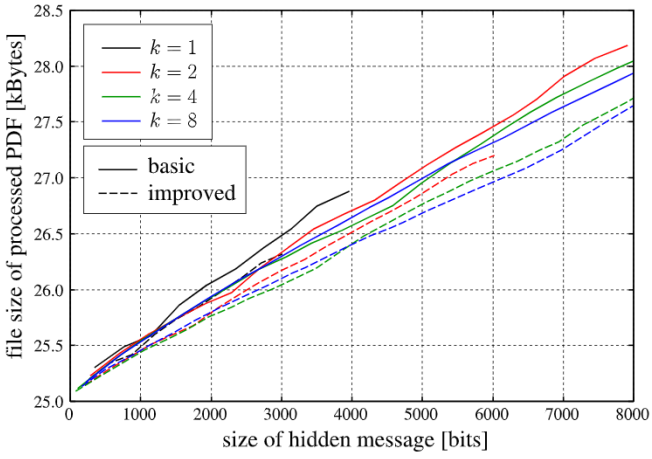

Fig. 6. Comparison of file size for the processed PDF with compression for basic and improved methods when $t = 2$.

4.1. Dependency on t and k

To embed data into the PDF file, we first decompress the document in the PDF file and insert one t-bit segment (i.e., $\boldsymbol{m}^{(j)}$) of the payload μ into each space value x_j. The *CRT* is then rectified by using PDFtk to validate the PDF file. The results of file size increment are plotted in Fig. 4 for various combinations of k and t. Here, we run the experiment for 100 different random payload data and report the average results. In general, the file size increases more when either k or t increases. Furthermore, it is observed that the gradient of the graph increases when t increases. This is an expected phenomena because each space value is split into t values and the number of inserted blank parenthesis, i.e., "()", is increased accordingly. As a result, the file size also increases.

The increase of file size of the processed PDF is measured by holding $t = 2$. Fig. 5 shows the results for the uncompressed files. As analyzed in Section 3.1, the larger k is, the smaller the increase of file size becomes. The size of the processed PDF is measured both for the basic and improved methods, and the comparison of performances is plotted in Fig. 6. In this experiment, we select $[99+2^k, 999]$ and $[-1000+2^k, -100]$ as two ranges for x_j. Due the non-linear performance of compression algorithm, the file size does not always becomes smaller for larger k when the size of the payload data is small. In any case, the improved method can suppress the increase of file size when compared to the basic method. However, the improved method slightly sacrifices the maximum payload because the space values without the two ranges are excluded for hiding data. Unless stated otherwise, we use the basic method in the rest of experiments.

Next, we calculate the amount of increased file size Δ, which is the difference of file size between the processed PDF and the original PDF plus $|\mu|$ (i.e., the payload size). The results are recorded in Table 3. The cost incurred for hiding data into the PDF file can be measured by considering the ratio $R := 8\Delta/|\mu|$. Here, smaller R implies better performance (i.e., more economic), and vice versa. It is observed that, in general, when k or t increases, the ratio R gets smaller, and this trend is more apparent when $k \geq 4$. The results also suggest that the file size of the processed PDF is increased when t increases.

We then repeat the processes by using the PDF file generated from Microsoft Word. However, this time, the owner password is set after hiding data. The results are recorded in Table 4. While the file size increment is larger in comparison to those observed in Table 3, the PDF generated from Word can accommodate larger payload in comparison to the PDF generated from Latex. More importantly, a similar trend in file size variation is observed. In other words, irregardless of the PDF converter, file size increases when either k or t increases, and the ratio R improves when either k or t increases. The results also suggest that setting an owner password does not affect the behavior of the proposed method.

Table 3
Increase of file size [Bytes] and ratio R for the PDF generated from Latex. The numbers in parenthesis are the sizes of the payload data μ [bits].

k	t			
	2	3	4	5
1	1850.1	3921.1	5828.3	7755.1
	(3971)	(7942)	(11913)	(15884)
R	3.73	3.95	3.91	3.91
2	3420.5	6589.9	9700.2	12422.3
	(7942)	(15884)	(23826)	(31768)
R	3.45	3.32	3.26	3.13
4	6074.3	11011.1	15139.5	18937.7
	(15884)	(31768)	(47652)	(63536)
R	3.06	2.77	2.54	2.38
8	11329.7	19495.2	26628.9	3333.0.9
	(31768)	(63536)	(95304)	(127072)
R	2.85	2.45	2.24	2.10
16	22704.9	37007.3	49805.3	61951.1
	(63536)	(127072)	(190608)	(254114)
R	2.86	2.33	2.09	1.95

4.2. File size increment due to setting password

In this section, the effect of setting access control on file size is measured by setting $t = 2$. The increase of file size due to setting access control is tabulated in Table 5, where the results are collected

Table 4
Increase of file size [Bytes] and ratio R for the PDF generated from Word. The numbers in parenthesis are the sizes of payload data μ [bits].

k	t			
	2	3	4	5
1	5535.9	9726.4	13688.2	17869.4
	(8601)	(17202)	(25803)	(34404)
R	5.15	4.52	4.24	4.16
2	8452.1	15218.7	22212.6	28639.6
	(17202)	(34404)	(51606)	(68808)
R	3.93	3.54	3.44	3.33
4	13881.3	25451.7	34907.2	43339.0
	(34404)	(68808)	(103212)	(137616)
R	3.23	2.96	2.71	2.52
8	26793.4	45683.9	61185.5	75620.9
	(68808)	(137616)	(206424)	(275232)
R	3.12	2.66	2.37	2.20
16	52686.9	84339.3	111539.8	137554.7
	(137616)	(275232)	(412848)	(550464)
R	3.06	2.45	2.16	2.00

Table 5
Comparison of file size [Bytes], where the size of the original PDF file is 24767 [Bytes] and that of its authorized file is 25029 [Bytes].

k	Owner password		diff
	off	on	
1	26617.1	26876.5	259.1
2	27927.8	28187.5	259.1
4	30581.7	30841.3	259.6
8	35836.8	36096.7	259.9
16	47212.0	47471.9	259.9

Table 6
Comparison of file size [Bytes] for different converters at OverLeaf.

Converter	Owner password		diff
(OverLeaf)	off	on	
pdfLaTeX	47127	47313	186
LaTeX	42100	42295	195
XeLaTeX	28155	28335	180
LuaLaTeX	28998	29176	178

by considering the processed PDF. In general, the file size increases after setting the permission irregardless of the value of k in use because the strings in the stream objects are encrypted and additional flags are appended to capture the specific access control set by the owner.

To further evaluate the file size increment with respect to the size of the hidden data, addition experiments are conducted by using 26 PDF files downloaded from the IEEE Xplore website. As photos and figures are also involved in the PDF files, we calculate the increase of file size for evaluation. The results are plotted in Fig. 7 for $(k,t) = (8,2)$. In the best case scenario, we can embed 38,160 bits of data with a penalty of 12,776 Bytes increase in file size.

4.3. Sensitivity to converter

We focus on the variation of file size when using different PDF converter. Note that no data is hidden here into the PDF to highlight that the file size varies significantly when different PDF converters are used. First, a LaTeX source file is compiled in Overleaf by using all four available compilers, and the file size of the resulting PDFs are recorded in Table 6 for both cases of setting (i.e., on) and not-setting (i.e., off) the owner password. Here, the document in all PDF files are compressed. As expected, although the Latex source file is the same, the size of the output PDF files are different. For example, the file size of the PDF produced by pdfLaTeX is about 68% larger than that of XeLaTeX. In addition, when a PDF file is opened by Adobe Acrobat Reader DC

Table 7
Comparison of file size [Bytes] when a Word file is exported as a PDF file through various paths of conversion. Here, no passwords are set.

Converter	Original	After PDFtk
Microsoft Word	68292	103451
PDFtk	103451	103522
Google Doc	77631	77469
LibreOfice	36596	36437
OpenOffice	52465	52351
Pages	48554	42247

Table 8
Comparison of file size [Bytes] of compressed PDF files with the method in [17] under the same amount of payload.

	Payload [bits]					
	2040	4248	8296	16696	24896	31768
Nursiah et al. [17]	26061	26929	28245	31431	34470	36863
Proposed	25964	26745	28020	30924	33752	36126

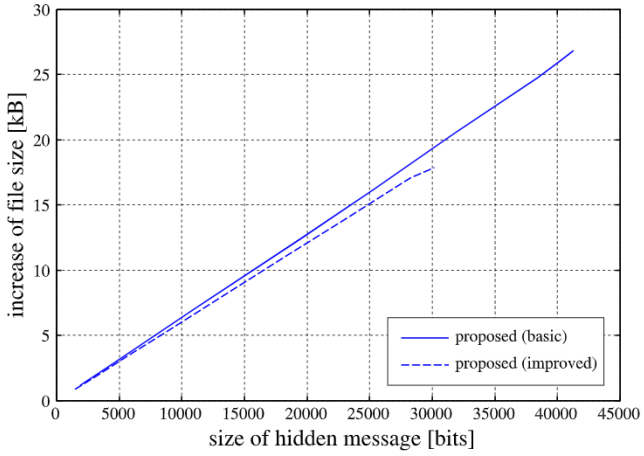

Fig. 7. The graph of *average file size increment* vs *size of hidden data* for 26 PDF files downloaded from the IEEE Xplore website. Here, $t = 2$ and $k = 8$..

(Version 2019.021.20061) and a copy is saved by using a different file name, the file size also changes. It is because Adobe Acrobat Reader DC rearranges the structure in a PDF to reduce the file size.

Next, the PDF file generated from Word is considered. The original PDF file size is 68,292 Bytes, and it expands to 203,731 Bytes when it is decompressed. The file size becomes 103,451 Bytes (without hidden data) after re-compression by using the PDFtk tool. The same Word document source is subsequently uploaded to Google Doc and downloaded as different file formats. By using some popular word processor programs, we produce PDFs and record their file size in Table 7. Results suggest that the range of the PDF file size (i.e., 2nd column) is large, hence making file size an unreliable feature to determine the existence of hidden data. For example, according to Table 4, when inserting 8601 bits into the PDF file using $(k,t) = (1,2)$, the change in file size due to data insertion is, on average, 4831 Bytes. In comparison, according to Table 7, the variation of file size due to differences in converter is 61,204 Bytes, which is bigger by at least 10 folds. Therefore, file size cannot be considered as a reliable feature to determine the present or absence of any hidden data.

One may argue that the converter information (i.e., Producer) can be easily found in the PDF, hence giving some hints about the absence/presence of hidden data (e.g., no data can be hidden into a PDF generated from Google Doc). However, this section of the PDF is not protected and therefore it can be easily modified. In fact, we can exploit it to further deceive the untrained eyes by stating that the processed PDF is converted from "Google Doc" although it is not the case.

Table 9
Functional comparison among the proposed and conventional data hiding methods.

Method	Scalability	Visual Distortion	Capacity	Operators	File size increment
Proposed	Yes	None	Large	TJ	Small
Iwamoto et al. [15]	No	None	Small	Td, TJ	Large
Bitar et al. [10]	Yes	Subtle	Small	Td	Small
Kuribayashi et al. [13,14,18]	Yes	Subtle	Middle	TJ	Small
Nursiah et al. [17]	Yes	None	Large	TJ	Small

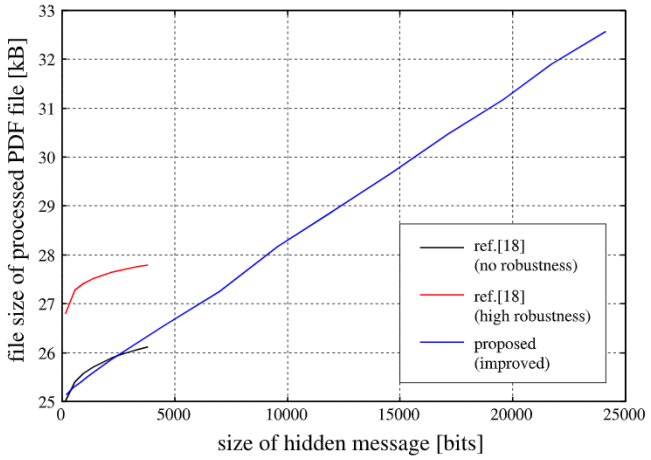

Fig. 8. Comparison with the method in [18], where $t = 2$ and $k = 8$ in the proposed method.

4.4. Comparison

In this section, comparison between the proposed and conventional data hiding methods is presented. As mentioned in Section 2.2, Iwamoto et al. hide the payload data by dividing texts in each 'TJ' operator without causing distortion. As a text string is represented by one 'TJ' operator for each line in the original PDF file, the number of text strings represented by the 'TJ' operator is 153. If each text string is split according to each payload bit, the payload in Iwamoto et al.'s method is only 153 bits, while the average increase of file size is $1,413.4$ Bytes. On the other hand, when only the first 2 text strings represented by the 'TJ' operator are considered in the proposed method with $t = 2$ and $k = 6$, 162 payload bits can be embedded, while the increase of file size is only 125.8 Bytes in average. Therefore, the proposed method can hide a similar payload (in fact, slightly larger in this case), while the penalty on file size is smaller in comparison to Iwamoto et al.'s method. We also compared the proposed method against another distortion-free data hiding method designed by Nursiah et al. [17]. Specifically, we hide the same amount of payload by using [17] and compare the file size of the resulting (compressed) PDF files. The results are recorded in Table 8. Although both the proposed method and [17] are able to offer as much room as needed to hide data, results suggest that the proposed method produces PDF of smaller file size when compared to [17].

Next, among the conventional methods that subtly distort the visual appearance, we consider Kuribayashi et al.'s method [18] because it can hide more payload data with less distortion. Specifically, when 162 payload bits are hidden by using the smallest embedding strength, the file size becomes, on average, 1,161 Bytes larger than the original file. When the maximum capacity with smallest strength of [18] is considered, it can hide 3802 bits, while the observed increase of file size is $1,348.3$ Bytes. Fig. 8 also shows the comparison with the method [18] in terms of the size of payload data and file size. We consider the improved method with $(k, t) = (8, 2)$ by selecting two ranges, namely, $[355, 999]$ and $[-744, -100]$. Results suggest that the proposed method can embed significantly more payload data than the Kuribayashi et al.'s method [18]. Even in the case of low capacity, the increase of file size in [18] is comparable to that of the proposed method. However, it should be noted that the advantage of [18] is its tolerance against additive noise. Although the degree of distortion caused by hiding in [18] is suppressed, visual distortion is inevitable. Such a characteristic may be exploited by attacker to determine the existence of hidden data, even if the permission is restricted by setting an owner password. Comparatively, our proposed method offers a lower payload for $(k, t) = (1, 2)$, but more payload can be accommodated by considering a larger k or t at the expense of larger file size increment. In addition, the proposed method never causes visual distortion, hence completely preserving the quality and appearance of the original PDF file.

Table 9 summarizes the functional comparison among the proposed method, Iwamoto et al.'s method [15], Kuribayashi et al.'s method [18], and Nursiah et al.'s method [17]. Based on the summary, we conclude the our proposed method is very competitive because it offers both scalability in terms of payload and it completely preserves the visual appearance of the document.

5. Conclusions

In this work, we put forward a data hiding method for PDF file that completely preserves the visual appearance of the original file. The proposed method splits space values into multiple related values, where data can be hidden into these related values. The secrecy of the hidden data is managed by setting an owner password, which is a common practice when handling PDF documents. Due to the controlled access to the processed PDF file, the observation of space values is restricted. It enables us to conceal the existence of hidden data from attackers. Although file size is increased after hiding data, the diversity of descriptions in PDF file makes the analysis on file size difficult, because different PDF converter outputs different PDF file with different size.

As future work, we will investigate into potential optimization to further suppress file size increment when there are more space values than the length of the payload bits. Furthermore, we want to explore the 'Td' operator for data hiding so that the proposed method can be deployed, including those generated from Google Doc. Moreover, it is interesting to consider the use of fingerprinting technique so that illegal users can be identified from the PDF file leaked from an access-controlled database.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has been partially supported by the Kayamori Foundation of Information Science Advancement, JSPS, Japan KAKENHI Grant Number 19K22846, and the Fundamental Research Grant Scheme (FRGS) MoHE, Malaysia Grant under the project - Recovery of missing coefficients - fundamentals to applications (Grant No. FRGS/1/2018/ICT02/MUSM/02/2).

References

[1] Incorporated AS. Document Management — Portable Document Format — Part 1: PDF 1.7. International Organization for Standardization; 2008, ISO 32000-1:2008.

[2] Graber-Stiehl I. Science's pirate queen. 2018.

[3] Frederik Obermaier VW, Bastian Obermayer, Jaschensky W. About the panama papers. 2016.

[4] Cox I, Miller M, Bloom J, Fridrich J, Kalker T. Digital Watermarking and Steganography. second ed. Morgan-Kaufmann; 2007.

[5] Katzenbeisser S, Petitcolas FAP. Information Hiding. Artech House; 2015.

[6] Zhong S, Cheng X, Chen T. Data hiding in a kind of PDF texts for secret communication. Int. J. Network Security 2007;4:17–26.

[7] Lee IS, Tsai WH. A new approach to covert communication via PDF files. Signal Process 2010;90(2):557–65.

[8] Lin H-F, Lu L-W, Gun C-Y, Chen C-Y. A copyright protection scheme based on PDF. Int. J. Innov. Comput., Inform. Control 2013;9(1):1–6.

[9] Por LY, Delina B. Information hiding: a new approach in text steganography. In: Proc. ACACOS'08; 2008. p. 689–695.

[10] Bitar AW, Darazi R, Couchot J-F, Couturier R. Blind digital watermarking in PDF documents using spread transform dither modulation. Multimedia Tools Appl 2017;76(1):143–61.

[11] Hatoum MW, Darazi R, Couchot J-F. Blind PDF document watermarking robust against PCA and ICA attacks. In: Proc. of SECRYPT2018, vol. 2; 2018. p. 420–427.

[12] Hatoum MW, Darazi R, Couchot J-F. Normalized blind STDM watermarking scheme for images and PDF documents robust against fixed gain attack. Multimedia Tools Appl 2019.

[13] Kuribayashi M, Fukushima T, Funabiki N. Data hiding for text document in PDF file. In: Proc. IIHMSP'17; 2017. p. 390–398.

[14] Kuribayashi M, Fukushima T, Funabiki N. Robust and secure data hiding for PDF text document. IEICE Trans. Inform. Syst. 2019;E102-D(1):41–7.

[15] Iwamoto T, Kawamura M. Proposal for invisible digital watermarking method for PDF files by text segmentation (in Japanese). IEICE Tech. Rep., EMM2016-59 2016;116(303):31–5.

[16] Chen B, Wornell GQ. Quantization index modulation: a class of provably good methods for digital watermarking and information embedding. IEEE Trans Inform Theory 2001;47(4):1423–43.

[17] Nursiah N, Wong K, Kuribayashi M. Reversible data hiding in PDF document exploiting prefix zeros in glyph coordinates. In: APSIPA 2019; 2019. p. 1298–1302.

[18] Kuribayashi M, Wong KS. Improved DM-QIM watermarking scheme for PDF document. In: IWDW'19. LNCS, Springer, Heidelberg; 2019, p. 171–83.