A Permutation-based Reversible Data Hiding Method with Zero Visual Distortion

Wendi Zhu[†], KokSheik Wong[†], Minoru Kuribayashi[‡]
[†]Monash University Malaysia, Malaysia.
[‡]Tohoku University, Japan.
{wzhu0045@student., wong.koksheik@}monash.edu, kminoru@tohoku.ac.jp

***Abstract*—Recently, several approaches have been designed to hide data in portable document format (PDF) files. These approaches have demonstrated their advantages in different application scenarios, including copyright verification, covert communication/steganography, and content forensics. However, they often suffer from visual distortion or lack universal applicability. In this work, we propose a reversible and transparent method that exploits the coding properties of text objects (i.e., substrings) in a PDF-compliant document to embed data. In particular, the position information of the substrings is adjusted to hide data, where each unique permutation of the substrings encodes a bit sequence. Subsequently, the distance of each substring from the left margin is corrected so that the processed PDF has the exact layout or appearance of the original PDF, hence completely preserving the quality of the original PDF file. In the best-case scenario, to hide one bit of data, 5.88 bits of the PDF file are required, i.e., $1 : 5.88$. In addition, this method can be deployed in tandem with conventional data hiding methods to hide more data and to hide data in different ways.**

I. Introduction

Data hiding is a process of concealing data in a host such as text, image, audio, and even DNA sequences to serve specific purposes [1], [2]. In recent years, much effort has been put into data hiding research for images and videos, but research into data hiding algorithms for text document has received relatively less attention. However, as generative artificial intelligence (AI) improves, more users turn to large language models (LLM) for various creative projects. It is essential to acknowledge that the rapid advancement of LLM also serves as a continuous reminder of the dangers associated with fake news and hence the need to safeguard the copyrights and integrity of textual contents [3]. Among the digital file formats for text, PDF is the most adopted format due to its extensive list of features, including preserving layout (look-and-feel), providing universal accessibility, supporting interactive features, to name a few [4]. More importantly, PDF documents remain ubiquitous in the current digital landscape, where online working and learning are becoming increasingly popular. In addition, despite the proliferation of these technologies, we have not significantly advanced in creating more universal file types in addition to the PDF. However, the platforms for accessing PDF content are rapidly expanding. As the digital ecosystem evolves, ensuring the integrity and safety of content within PDFs becomes paramount, not just for safeguarding intellectual property but also for protecting personal and sensitive information.

As the PDF document standard evolves, many innovative and clever methods have been developed for hiding data within PDF files. However, almost all conventional data hiding methods fail to maintain visual consistency after the data is hidden. It is due to the exploitation of visible text features in PDF files, including font type, size, and color, to hide data. Our method distinguishes itself from the conventional methods by ensuring that the document's appearance remains identical to that of the original document, i.e., complete quality preservation (CQP) [5], [6]. We leverage the inherent characteristics of the PDF format to facilitate data hiding, where only invisible features are manipulated to hide data. Our work makes the following contribution: (a) exploiting the rendering order of substrings within the 'TJ' operators to hide data, which is the first of its kind; (b) achieving complete quality preservation in the processed PDF file; (c) achieving reversibility by removing the inserted data, and; (d) enabling integration with conventional methods to hide more data and hide data in different ways.

II. Related Work

This section establishes the rationale of our study by providing an overview of the PDF file structure, examining conventional distortionary and distortion-free data hiding methods, identifying research gaps and defining quality-preserving techniques.

A. Overview of PDF file structure

Portable document format (PDF) has four integral segments: *Header*, *Body*, *Cross-Reference Table (CRT)*, and *Trailer* (Fig. 1). The *Header*, which only includes the PDF ISO standard version, starts with '%PDF-' followed by '1.4' or '2.0', indicating the PDF build standard. The *Body* contains critical content, including various operators that facilitate the organization and presentation of the text. For instance, the 'Tf' operator indicates the font type used, and the 'TJ' operator controls the text content rendering (Fig. 1). The *CRT* facilitates access to indirect objects within a PDF file, while the *Trailer* assists PDF processors in quickly locating essential elements.

The key advantage of PDFs is their portability, ensuring consistent document appearance across diverse platforms. Each

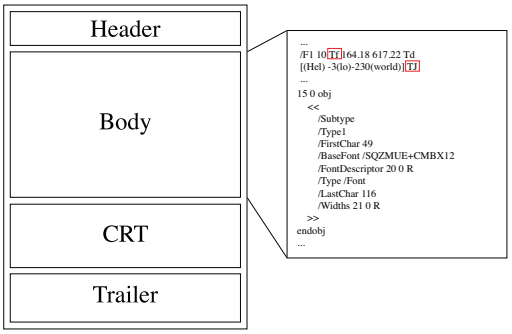


Fig. 1. PDF Structure

PDF file complies with ISO guidelines [7] and supports various media forms [8]. PDFs are also more secure than formats such as Microsoft Word, leading to increased focus on enhancing their security features [9]. Important PDFs are often encrypted with user and owner passwords, where the user password allows viewing, while the owner password sets restrictions on copying and editing [5]. These characteristics make PDFs the preferred choice for handling important documents.

B. Synonym-based data hiding method

Researchers have explored synonym-based methods to hide data in text documents, replacing words like 'teaching' with 'tutoring' to encode data such as '01', while keeping 'teaching' to encode '00' [10]. However, synonym substitution can lead to significant content distortion and cannot conceal a large payload, as achieving larger payloads requires significantly longer text. Xiang et al. proposed using a Word Indexing Compression (WIC) algorithm to compress the cipher text before hiding, thereby enhancing the feasibility of using synonyms for data hiding [11].

C. Image-based data hiding method

Due to its portability, researchers continue to study data hiding methods in PDF files by converting them into images and modifying the content at the pixel level. Li et al. modified the text boundaries to embed data, causing slight page distortion imperceptible to humans, but it suffers from having a limited capacity, i.e., only 2 bits can be embedded per text line [12]. Khosravi et al. devised a method that adjusts the details of multiple character spacing to hide data [13]. Their approach leveraged different text editors' varying settings for correcting space, to embed hidden data in the document without notable yet inevitable distortion.

D. Open space data hiding method

Besides synonym-based methods, researchers also attempted to manipulate the open space due to its transparency. UniSpaCh by Por et al. [14] exploited multiple Unicode space characters to encode the data in the files. However, a statistical analysis could detect irregularities in using Unicode space characters, which is rare in an ordinary document.

Many open space methods cannot achieve complete quality preserving (CQP) [14], but some are distortion-free and more transparent than the earlier method . One such method manipulates non-breaking spaces ('A0' in hexadecimal) in PDF editors [15]. This approach substitutes the original ASCII code space '20' by 'A0' to encode '1' in binary or leaves it unchanged (i.e., remains as '20') to encode '0'. However, embedding data n in the spaces between characters allows higher capacity but leads to file size expansion. Furthermore, embedding data in spaces between words will decrease the capacity, and some editors may display the 'A0' characters visibly.

E. Data hiding method based on PDF's feature

Some researchers have dealt with the PDF structure standard. Kuribayashi et al. introduced a technique that utilizes the 'TJ' operator to manipulate the encoding of the space information [5], [16]. An integer I is split into $(I-m_i)+(m_i)$, where m_i is the decimal equivalent of the bit sequence to be embedded. Since the space value I remains unchanged, Kuribayashi et al.'s method completely preserved the visual consistency. However, this method introduced new pairs of parentheses '()' followed by the numbers m_i. This feature leads to a diminished embedding efficiency and a surge in redundancy.

A data hiding method is considered CQP if the original and modified documents (i.e., embedded with data) are identical. The methods mentioned above mainly distort the visual / layout of the processed document due to hiding data. Therefore, there is a gap in exploring more CQP approaches. Before our current work, CQP data hiding research mostly manipulated the whitespace between substrings to conceal information. We will contribute to providing a new permutation-based method with no visual distortion by adjusting the rendering order of the text showing operators.

III. Methodology

We propose a method to alter the order of substrings, then subsequently restore the text width and spacing by adjusting their glyphs. The framework consists of two main parts: (1) hiding data, and; (2) extracting the hidden data and restoring the original PDF file. The following subsections detail the subprocesses involved in each part of the framework.

A. Embedding data

1) Encrypting the data: Without loss of generality, we assume that the data m to be hidden is encoded in binary. We divide m into non-overlapping segments m_j, where each segment can assume a different length k_j, which is defined later in Eq. (1). Each segment m_j is then encrypted using a stream cipher to obtained the encrypted segment n_j.

2) Determining substrings' order: We embed each encrypted segment n_j by forcing a group of selected substrings to appear in a specific order. To facilitate the presentation, suppose that we have a PDF file that encodes only the phrase "Hello world". "Hello world" can be encoded using three substrings, for example, `(Hel)`, `(lo)`, and `(world)`. We

TABLE I
ALL POSSIBLE PERMUTATIONS OF THREE SUBSTRINGS. HERE, *Lehmer Code* IS USED FOR ASSOCIATING EACH PERMUTATION WITH A 2-BIT SEQUENCE.

Permutation	2-bit
u_1, u_2, u_3	00
u_1, u_3, u_2	01
u_2, u_1, u_3	10
u_2, u_3, u_1	11
u_3, u_1, u_2	-
u_3, u_2, u_1	-

refer to the aforementioned substrings as u_1, u_2, and u_3, respectively.

Since the total number of substrings for this specific line is $N = 3$, there are $3! = 6$ unique permutations for these substrings. In other words, there are six unique orders to display the substrings. Each permutation can subsequently be associated with k bits, where

$$k = \lfloor \log_2(N!) \rfloor. \quad (1)$$

Table I lists all possible permutations for the case of $N = 3$ and one specific way (out of ${}_6P_4 = 360$ ways) to map 4 arrangements to an $k = 2$ bits sequence. Here, we use *Lehmer Code* [17] to map each permutation to a unique decimal number and prepare its k-bit binary representation. Next, we identify the specific permutation of substrings based on the encrypted data segment n_j to be embedded. For the specific mapping stipulated in Table I, the substrings are rearranged to assume the order O_j of $(u_2,\ u_1,\ u_3)$ to hide n_j = '10'. Specifically, the substrings will appear in the order of O_j. Using the example above, the order is '`lo`', '`Hel`' and '`world`'.

Note that the number of substrings N_j appearing on the j^{th} line of the PDF document dictates the maximum number of bits that can be embedded. In general, $k_j = \lfloor \log_2(N_j!) \rfloor$ bits can be embedded in the j^{th} line. In fact, the length of the j^{th} data segment depends on k_j, i.e., len(m_j) = len(n_j) = k_j, where len(z) returns the length of the input z. Furthermore, a specific mapping rule is needed for each $N_j \in \mathbb{N}$, which can be generated using a private key κ.

3) Retrieving characters' width: In most cases, after hiding the data segment n_j using the proposed permutation-based mapping, the order of the substrings differs from that of the original document (e.g., '`lo`', '`Hel`', and '`world`' as mentioned above). To rectify the order so that they appear exactly where they were in the original document, we adjust the spacing between every two consecutive substrings. Specifically, each content stream refers to the font dictionary, where its font program and related information are stipulated. This reference is located in the 'Tf' operator of the content stream, specifically, the first operand. It can be subsequently mapped to the actual dictionary using a font entry under the resource dictionary. An example of a font dictionary and font graphics Width array is shown in Fig. 2(a) and (b), respectively.

Specifically, the font dictionary stores crucial information that dictates the width of individual glyphs. We are particularly

```
15 0 obj
  <<
    /Subtype /Type1
    /FirstChar 49
    /BaseFont /SQZMUE+CMBX12
    /FontDescriptor 20 0 R
    /Type /Font
    /LastChar 116
    /Widths 21 0 R
  >>
endobj
```

(a) Font dictionary

```
21 0 obj [563 563 563 563 563 0 0 0 0 0 0
0 0 0 0 0 0 0 813 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 547 0 0 0
513 0 0 625 0 0 0 0 0 0 0 625 0 459 0 438]
endobj
```

(b) Font graphics Width array

Fig. 2. Example of (a) font dictionary, and (b) font graphics Width array.

interested in the following three entries: *FirstChar*, *LastChar*, and *Widths*. Specifically, *Widths* is an array that contains the width of the glyph for characters whose Unicode ranges from *FirstChar* to *LastChar*, inclusive. In other words, the glyph width of a character with unicode i is recorded as the $(i + \mathit{FirstChar})^{th}$ entry in the *Widths* array. Glyph width determines the horizontal space a character occupies on a page, viewed as non-overlapping rectangles from top to bottom, each containing one row of text.

4) Modifying space offsets: Once the glyphs' widths are retrieved from the *Widths* array, the space offsets can be computed. Specifically, let d_i denote the distance of the i^{th} substring from the left margin in the original text position. Here, d_i can be decomposed as follows:

$$d_i = d_{i-1} + w_{i-1} + s_i, \quad (2)$$

where w_i is the width of the i^{th} substring (i.e., summation of its characters' width), and s_i is the space offset between the i^{th} and $(i-1)^{th}$ substrings. Note that the actual encoded value is the opposite of s_i (i.e., $-s_i$). When the value of s_i is increased from 1 to 100, this adjustment causes the subsequent substrings to shift towards the left. This observation is critical for understanding the dynamics of the system in the upcoming sections. Fig. 3 illustrates the visual semantics of the distance d_i, the width w_i, and space offset s_i.

Next, the order of the permuted substrings is visually corrected to render the original sequence of substrings. Let i' denote the order of i^{th} substring in the processed 'TJ' (i.e., containing the permuted substrings). The space value $s_{i'}$ for each string is recomputed as:

$$s_{i'} = d_i - d_{i'-1} - w_{i'-1}, \quad (3)$$

where s_i' refers to the space offset between the $(i')^{th}$ and $(i'-1)^{th}$ substrings, and $d_{i'} - 1$ is the distance of the $(i'-1)^{th}$ substring from the left margin. For example, suppose an original 'TJ' tag is rendered as

$$[(\texttt{Hel})3(\texttt{lo})230(\texttt{world})]\ \text{TJ}. \quad (4)$$

Here, the width of each substring is computed by adding the width of its individual characters (reference is made to the *Widths* array). Specifically, $w_{\text{“Hel”}} = 1351$, $w_{\text{“lo”}} = 757$, and $w_{\text{“world”}} = 2346$. Subsequently, the original distance from the

left margin is computed for each substring:

$$\begin{aligned} d_{\text{“Hel”}} &= 0; \\ d_{\text{“lo”}} &= d_{\text{“Hel”}} + w_{\text{“Hel”}} + s_{\text{“lo”}} \\ &= 0 + 1351 + 3 = 1354; \\ d_{\text{“world”}} &= d_{\text{“lo”}} + w_{\text{“lo”}} + s_{\text{“world”}} \\ &= 1354 + 757 + 230 = 2341. \end{aligned} \tag{5}$$

Suppose our substrings are permuted as follows to encode the data segment '10':

$$[-s_{\text{“lo”}}\ (\texttt{lo})\ -s_{\text{“Hel”}}\ (\texttt{Hel})\ -s_{\text{“world”}}\ (\texttt{world})]\ \text{TJ}. \tag{6}$$

By referring to the distance values d_i stipulated in Eq. (5), the space values $s_{i'}$ are adjusted as follows:

$$\begin{aligned} s_{\text{“lo”}} &= d_{\text{“lo”}} = 1354; \\ s_{\text{“Hel”}} &= d_{\text{“Hel”}} - d_{\text{“lo”}} - w_{\text{“lo”}} \\ &= 0 - 1354 - 757 = -2111; \\ s_{\text{“world”}} &= d_{\text{“world”}} - d_{\text{“Hel”}} - w_{\text{“Hel”}} \\ &= 2341 - 0 - 1351 = 990. \end{aligned} \tag{7}$$

To complete the correction step, the 'TJ' tag is updated to:

$$[-1354\ (\texttt{lo})\ 2111\ (\texttt{Hel})\ -990\ (\texttt{world})]\ \text{TJ}. \tag{8}$$

With the aforementioned updates, the text "Hello world" will appear exactly the same as in the original PDF file. In essence, instead of the original rendering sequence, the offset for the substring '`lo`' (i.e., $s_{\text{“lo”}}$) is first computed, and the substring '`lo`' is rendered. Subsequently, the offset for the substring '`Hel`' is computed, and the substring '`Hel`' is rendered. The same process is applied to handle '`world`'.

5) Setting permissions: According to PDF standards, assigning both owner and user passwords is common for protecting sensitive documents. The user password allows viewing and, if permitted by the owner, content copying. We restricted document access to viewing only (which is a commonly implemented setting, e.g., credit card statement), ensuring that our raw data remains unexplored and the no-distortion cover remains unsuspicious.

B. Extracting data

Contrary to the embedding process, we must open the processed and protected PDF file using the password and perform the extraction operation. Since each 'TJ' operator is independent, we only need to iterate through all the 'TJ' operators from the beginning of the PDF document instead of extracting them in reverse order.

First, the number of substrings on the j^{th} line N_j is determined. Next, the original ordering of the substrings is extracted from the processed text (with hidden data) by comparing the definite distance of each substring from the left margin, i.e., the entity d_i. Specifically, d_i is determined by rearranging the terms as follows:

$$d_i = s_{i'} + d_{i'-1} + w_{i'-1}. \tag{9}$$

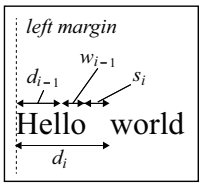


Fig. 3. Illustration of the semantic for d_i, d_{i-1}, w_{i-1} and s_i. Here, '`world`' is the i^{th} substring.

Here, a smaller d_i implies that the associated substring s_i is closer to the left margin and hence the i^{th} sub-string would appear earlier in the original order, and vice versa.

For illustration purposes, we can examine the example captured by Eq. (7), where the number of substrings on the j^{th} line is $N_j = 3$. By comparing the distances d_i of the substrings '`world`', '`Hel`', and '`lo`', we can determine their original positions as 3^{rd}, 1^{st}, and 2^{nd}, respectively. This is deduced from the fact that $d_{\text{“Hel”}} < d_{\text{“lo”}} < d_{\text{“world”}}$. Consequently, this ordering is represented by the permutation (u_3, u_1, u_2). By referencing the mapping rule in Table I, we can extract the hidden data, i.e., '01'. This process is repeated for the remaining rows for further extraction until the entire document is processed. The extracted encrypted data segment n_j is decrypted to obtain the original data segment m_j. The data segments m_j are subsequently concatenated to form the complete data m.

IV. EXPERIMENTS

This section evaluates and validates the proposed method in terms of transparency, capacity, and robustness. The proposed data hiding method is implemented in Python3. The first five chapters of Genesis in the Old Testament of the Bible (generated by Latex using TeX Live 2016) and the story The Romance of Three Kingdom (https://threekingdoms.com/), and generated using Microsoft Word 2016 (Version 16.0.17328.20184) are utilized for experiment purposes.

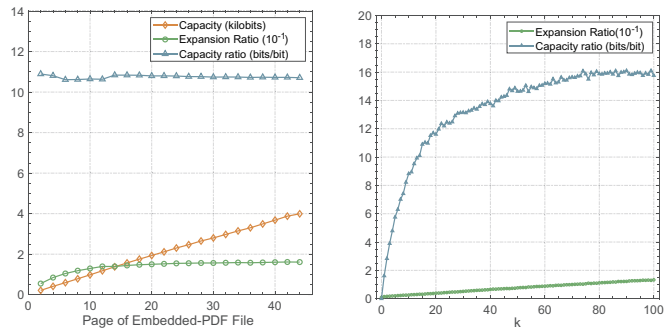
A. Transparency

Transparency refers to the level of similarity between the processed PDF and its original counterpart, where higher similarity implying higher transparency, and vice versa. We adopt SSIM image quality metric to quantify the similarity. The results consistently showed an SSIM of 1.0, indicating that our data hiding method does not introduce any visual distortions (i.e., CQP) and thus maintains the highest possible level of transparency.

B. Capacity

This section examines how the performance of the proposed method is influenced by the number of pages in the PDF file and capping the number of bits to be embedded using each 'TJ' line. The metrics adopted to measure the rate of change in file size ϕ and the capacity ratio η are defined as:

$$\text{Expansion Ratio, } \phi = \frac{\text{Increment of File Size}}{\text{Original File Size}}, \tag{10}$$



(a) Capacity/ϕ/η vs number pages (b) Capacity/ϕ vs parameter k

Fig. 4. Performance evaluation for various number of pages and parameter values for k.

$$\text{Capacity Ratio, } \eta = \frac{\text{Size of hidden data (bits)}}{\text{File size increment (bytes)} \times 8}. \quad (11)$$

The results in Fig. 4(a) show that capacity (orange diamond) increases linearly with the number of pages, using all text rows for data hiding. The capacity ratio η (blue triangle) remains constant at ~ 0.17 data bits per PDF bit. On the other hand, the file size expansion ratio ϕ (green circle) gradually increases with the number of pages but plateaus when the number of pages exceeds 30.

Our method favors shorter PDFs, with $\phi = 5.5\%$ for hiding $21,339$ bits in a two-page PDF. However, for PDFs with ≥ 30 pages, ϕ remains nearly constant. For instance, $\phi = 15.7\%$ for concealing $280,165$ bits in a 30-page PDF and rises slightly to 16.1% for $399,256$ bits in a 44-page PDF. This minimal increase in ϕ is likely due to the fundamental base size of PDF documents, where added data represents a smaller percentage of the total file size as the document lengthens, slowing the rate of ϕ increase.

Figure 4(b) examines how the parameter k (bit length embedded into each 'TJ' line) affects the performance of our data hiding method. Adjusting k helps control the file size expansion ratio ϕ, capping the payload per line between 0 and 100 bits. As k increases, so does ϕ. For example, with $k = 29$ bits, $\phi = 4.9\%$. Thus, selecting a suitable k value depends on the acceptable level of ϕ.

C. Robustness and Security

To understand practical applications, we evaluated the Bit Error Rate (BER) for data extraction after applying standard lossless compression (ZIP/Flate) on PDFs, embedding at maximum rate on all pages. Our experiments confirmed that data embedded in PDFs created by Latex or Microsoft Word remained unchanged after compression and decompression, achieving a BER of 0.

As for security, our proposed method offers the first-of-its-kind feature, i.e., allowing data extraction from the copied-and-pasted texts. Imagine a scenario where an attacker, during a business meeting or critical event, attempts to rapidly copy the contents of a PDF into her document. She will find that the text appears disordered and chaotic upon pasting, because the substrings are deliberately reordered to hide data. This unexpected strategy effectively shields the file from rapid copy-paste actions. If the copy-paste operation is indeed performed, the unique order can be examined to determine the hidden data, which can contain data such as the owner, document source, etc. More importantly, our proposed method allows two ways of data extraction. The first way is the typical data extraction pathway using the processed PDF file, and the second way is to examine the pasted texts. This mode of data extraction will be further explored in our future work.

V. DISCUSSIONS

A. Comparison with conventional methods

To demonstrate the advantages of the our proposed method, we conduct a comparative analysis, examining the performance of both past representative methods and those within the same category as ours.

Li et al.'s method [12] converts a PDF host document to an image, then modifies minor section of the edge of the characters. However, they can only embed 2 bits of data in a single line, hence its capacity is very low for practical use. On the other hand, Lee & Tsai's classic method [15] exploits the special character 'A0' to mark spaces [15], where they proposed two ways: replace the '20' code by 'A0', which leads to no file size increment but suffers from low capacity, or; to add additional 'A0' space between the characters which brings high capacity but at the expense of increasing the file size.

As for the method using the 'TJ' operator, Kuribayashi et al. utilized space lengths in the 'TJ' operators by considering them as a one-dimensional feature, making subtle distortion [16]. In a different work, Kuribayashi & Wong [5] introduce no-distortion after hiding data in a PDF file, but their work suffers from large ϕ, for example, hiding 3,802 bits leads to $\phi = 12\%$.

More importantly, our proposed method manipulates the source code of the PDF so that it can be used on top of other conventional methods. For instance, since StealthPDF's solution introduces '()' to split the space value [5], it will increase the substrings in the 'TJ' operator, but with more substrings, we can hide more data. As an illustration, when setting $k = 29$ as narrated in Section IV-B, we can continue to embed data in a PDF processed by StealthPDF [5]. Here, we could embed $4,275$ bits, which is a 14-bit improvement over our method alone and a total of $42,435$ bits when combined with StealthPDF. On the other hand, if we combine our method with Jiang et al.'s method [18], we can add 4,275 bits directly to their capacity because their targeted embedding location is the commentary section of the dictionary, which does not conflict with our embedding process.

B. Limitation and future work

Although our method offers excellent transparency, there are mutual checks and balances on the performance of data hiding, as stated by Deljavan Amiri et al. [19]. Specifically, we are limited in the embedding capacity by the number of 'TJ' operators in the PDF file. In addition, since our approach

relies on the 'TJ' operators, it is not available in the face of 'Tj' operators.

In the future, we want to investigate more fonts and other PDF editors' editing behaviour to verify the general applicability of the proposed method. In addition, for the documents encoded using the 'Tj' operator, we will also consider using other examined operators as reported in the literature.

VI. Conclusions

In this work, we proposed a permutation-based method to hide data in text-containing PDF files. We employ an interesting technique that modifies the order of substrings to encode bit sequence. Subsequently, we adjust the glyphs of the permuted substrings to restore text width and spacing to render the exact same appearance as the original PDF, hence completely preserving the quality of original host PDF file. At the receiver's end, the hidden data can be extracted, and the original PDF file can be restored when the correct passwords and keys are in possession. Our proposed method stands out for its ability to embed data discreetly without compromising the visual integrity of the document, bolstered by encryption and access control to enhance security. It demonstrates a balanced performance between data hiding capacity and minimal file size overhead, where it requires $2,873$kB in the PDF file to embed over $21,000$ bits of payload. Our method's compatibility with the conventional data hiding methods expands its utility and security dimensions, making it a versatile tool for various applications, including copyright tracking, secure communication, and digital forensics.

Future research should expand the capabilities of text and multimedia operators in PDFs to fully utilize their potential as dynamic content containers. Additionally, we plan to explore ways to exploit the permuted or semi-randomized text produced by the copy-and-paste operation from our processed PDFs to achieve specific applications.

References

[1] D. Artz, "Digital steganography: Hiding data within data," *IEEE Internet computing*, vol. 5, no. 3, pp. 75–80, 2001.

[2] H.-J. Shiu, K.-L. Ng, J.-F. Fang, R. C. Lee, and C.-H. Huang, "Data hiding methods based upon dna sequences," *Information Sciences*, vol. 180, no. 11, pp. 2196–2208, 2010.

[3] A. Jo, "The promise and peril of generative ai," *Nature*, vol. 614, no. 1, pp. 214–216, 2023.

[4] A. Cakir, "Usability and accessibility of portable document format," *Behaviour & Information Technology*, vol. 35, no. 4, pp. 324–334, 2016.

[5] M. Kuribayashi and K. Wong, "StealthPDF: Data hiding method for PDF file with no visual degradation," *Journal of Information Security and Applications*, vol. 61, p. 102 875, 2021.

[6] K. Wong, M. N. M. Nazeeb, and J.-L. Dugelay, "Complete quality preserving data hiding in animated gif with reversibility and scalable capacity functionalities," in *Digital Forensics and Watermarking*, Cham: Springer International Publishing, 2021, pp. 125–135, ISBN: 978-3-030-69449-4.

[7] J. T. Hackos, "International standards for information development and content management," *IEEE Trans. Professional Communication*, vol. 59, no. 1, pp. 24–36, 2016.

[8] B. A. de Boer, A. T. Soufan, J. Hagoort, *et al.*, "The interactive presentation of 3d information obtained from reconstructed datasets and 3d placement of single histological sections with the 3d portable document format," *Development*, vol. 138, no. 1, pp. 159–167, 2011.

[9] A. Castiglione, A. De Santis, and C. Soriente, "Security and privacy issues in the portable document format," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1813–1822, 2010.

[10] F. Peng, X. Li, and B. Yang, "Improved pvo-based reversible data hiding," *Digital Signal Processing*, vol. 25, pp. 255–265, 2014.

[11] L. Xiang, W. Wu, X. Li, and C. Yang, "A linguistic steganography based on word indexing compression and candidate selection," *Multimedia Tools and Applications*, vol. 77, no. 21, pp. 28 969–28 989, 2018.

[12] L. Li, H.-J. Zhang, J.-L. Meng, and Z.-M. Lu, "Robust pdf watermarking against print–scan attack," *Sensors*, vol. 23, no. 17, p. 7365, 2023.

[13] B. Khosravi, B. Khosravi, B. Khosravi, and K. Nazarkardeh, "A new method for pdf steganography in justified texts," *Journal of information security and applications*, vol. 45, pp. 61–70, 2019.

[14] L. Y. Por, K. Wong, and K. O. Chee, "Unispach: A text-based data hiding method using unicode space characters," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1075–1082, 2012.

[15] I.-S. Lee and W.-H. Tsai, "A new approach to covert communication via pdf files," *Signal processing*, vol. 90, no. 2, pp. 557–565, 2010.

[16] M. Kuribayashi, T. Fukushima, and N. Funabiki, "Robust and secure data hiding for PDF text document," *IEICE Trans. Information and Systems*, vol. 102, no. 1, pp. 41–47, 2019.

[17] D. Lehmer, "Teaching combinatorial tricks to a computer," in *Symposia in Applied Mathematics*, vol. 10, American Mathematics Society, 1960, pp. 179–193.

[18] Z. Jiang, H. Wang, and S. Han, "A robust pdf watermarking scheme with versatility and compatibility," *Multimedia Tools and Applications*, pp. 1–27, 2024.

[19] M. Deljavan Amiri, M. Meghdadi, and A. Amiri, "HVS-based scalable image watermarking: A novel approach to image copyright protection against scalable compression," *Multimedia Tools and Applications*, vol. 78, no. 6, pp. 7097–7124, 2019.