2018 IEEE Conference on Multimedia Information Processing and Retrieval

SecureCMerge: Secure PDF Merging Over Untrusted Servers

Neha Sharma, Priyanka Singh and Pradeep K. Atrey
Albany Lab for Privacy and Security, College of Engineering and Applied Sciences
University at Albany, State University of New York, Albany, NY, USA
neha.mnnit4055@gmail.com, psingh9@albany.edu, patrey@albany.edu

***Abstract*—Merging two or more PDF files is an operation that is commonly performed by users, most often using freely available online tools, such as pdfmerge, or cloud-based services, such as cloudconvert. These free online servers cannot always be trusted and can often pose great risk to the confidentiality of the PDF files. In this paper, we present a method, called SecureCMerge, to merge the PDF files using free online merge sites in a secure way. Our core idea is to encrypt the content (text and image blocks) in the PDF files using Shamir's Secret Sharing (SSS) scheme before uploading it to a PDF merge server. We also show that the SSS scheme is merge homomorphic and the proposed method, by virtue of using the SSS scheme, provides information theoretic security. Experimental results demonstrate that the proposed method accomplishes secure PDF merging at an acceptable overhead in computation time and size.**

***Keywords*-Encrypted Domain; PDF Merging; Cloud Computing**

I. Introduction

PDF (Portable Document Format) is a file format commonly used on the Internet. A PDF file typically contains text and image blocks, and there are a number of operations that are commonly performed on the PDF files. Merging, i.e. combining two or more PDF files, is one such operation that is often carried out by the users. For instance, employers generally require applicants to combine their resume with other documents such as transcripts to make a single PDF file before submitting an application.

While the freely available PDF software, such as Adobe Acrobat Reader, provides many functions like searching, highlighting and commenting in a PDF document, they generally do not support the merge operation. Therefore, users use either paid software, such as Adobe Acrobat Professional, or freely available online services, such as www.pdfmerge.com, to merge their PDF documents. The latter option, by virtue of being free, is used extensively. However, this free online service comes at the expense of security and privacy. The PDF documents, which in some cases could be highly confidential, are uploaded to a third party server. These third party servers, often cannot be trusted as the documents can be accessed and misused by potential adversaries. The confidential information contained in these documents may include names, addresses, social security numbers, bank account details, judicial information to be used as evidence in a court of law by a jury, or information with national security implications [1]. Disclosure of such sensitive information makes a user vulnerable to online theft and fraud [2].

Maintaining the security and privacy of PDF files while using online PDF merge services is a significant challenge, which we address in this paper. The core idea behind our approach is to first encrypt the PDF files in such a way that only the PDF content is encrypted but the PDF file structure is preserved, then to perform the merge operation on the encrypted PDF files using an online PDF merge service, and finally to download and decrypt the merged PDF file to obtain the unencrypted merged file.

Although the proposed approach does not restrict us to use any specific encryption method, in this work[1] we use Shamir's secret sharing (SSS) scheme to create unintelligible shares of the PDF file. The key advantages of the SSS scheme over a conventional symmetric encryption method like Advanced Encryption Standard (AES) are: i) the SSS scheme does not use any key while the AES method requires a key that is used for encryption and decryption, and ii) compared to the computationally secure AES method, the SSS scheme is information theoretically (or perfectly) secure. However, despite having many benefits, the SSS scheme has the limitation that it needs at least two different servers for the merge operation.

In the past decade or so, the SSS scheme has been widely used to protect text [3], image [4], video [5] and audio data [6]. More recently, due to the low computation cost and information theoretic security, the SSS scheme has also been used for a number of operations in encrypted domain, such as audio noise reduction [7]. To the best of our knowledge, this is the first paper that not only employs the SSS scheme to protect PDF documents, but also utilizes its homomorphic property to enable the merge operation on the encrypted PDF files.

The rest of this paper is organized as follows. In Section II, we describe the PDF internal data structure and how text and images are stored in the PDF. In this section, we

[1]The work was performed at the University at Albany, and the first author worked as an intern with the third author.

0-7695-6354-6/18/$31.00 ©2018 IEEE
DOI 10.1109/MIPR.2018.00087

Table I: PDF structure

Structure	Text in PDF	Images in PDF
1 1 0 obj	1 stream	1 1 0 obj
2 << /Length 2 0 R	2 BT	2 << /Type /XObject
3 >>	3 /F1 24 Tf	3 /Subtype /Image
4 stream	4 1 0 0 1 260 254 Tm	4 /Name /Image1
5 data to be inserted here	5 (Hello World)Tj	5 /Width 480
6 endstream endobj	6 ET	6 /Height 651
7	7 endstream	7 /BitsPerComponent 8
8 2 0 obj	8 /ColorSpace /DeviceGray	8 /ColorSpace /DeviceGray
9 108	9 /Length 31800	9 /Length 31800
10 endobj	10 /Filter /DCTDecode>>	10 /Filter /DCTDecode>>
		11 stream ... endstream endobj

also provide a brief description of the SSS scheme. The proposed method for share generation, share merging and share reconstruction is discussed in Section III. In Section IV, we discuss the experimental results. We conclude the paper and propose future work in Section V.

II. Background and Preliminaries

A. PDF Structure

A simple PDF contains four parts: a header, a body, a cross-reference table and a trailer. The header consists of the PDF version and an option line to specify if the PDF contains binary data. The body consists of a series of objects that are used in the document. The text information such as character value and font, the images and the structuring information are encoded into these objects. The cross-reference table specifies the position of the objects. The trailer contains structural information such as the length of the document and a reference to the root object. The overall structure of the PDF document is mostly hierarchical like a graph [8]. The root object, which is present in the trailer, points to a page container object that references the pages, and so on. Different information is present in different objects, which can be shared by referencing them. The objects are readable from all other objects. The order of the objects does not matter, as all of them are treated in the same way.

Table I presents a brief description of the PDF structure [9]. Line numbers are included for readability. The general structure of a PDF document is shown in the leftmost column of the table. The data is often preceded by a dictionary (lines 2-3) that gives additional information about the data. In line 2, the length of the stream is determined by referencing the second object with the ID 2 0 located at line 8 that contains 108 as its data (line 9). In this way, all information about the text and image is embedded in different objects, which can be referenced from each other. All text-related commands are between a BT (line 1) and an ET (line 6) which stands for Begin Text and End Text, respectively (refer to the middle column of Table I). There are 3 different matrices that keep track of the current writing position. These matrices have 6 values, as shown in line 4. The first four represent a rotation matrix: scale-x, skew-x, skew-y, and scale-y respectively. The rotation matrix can be used to write landscape text or upside down. Also it gives scaling parameters, which are multiplied with the actual font sizes. The next 2 values give the position on the paper in pixels (x and y coordinates). Font type and font size are determined by the command in line 3. The images are stored in Xobjects (refer to the rightmost column of Table I). To actually extract the picture, we need the filter (as given in line 10), in this case DCTDecode, which is the PDF name for JPEG encoding. The stream simply contains a jpg file that can be copied without further modifications. Each image is present as a rectangular block, like a bitmap. The width and height are provided to determine the dimensioning of the rectangle and the BitsPerComponent (see line 7) indicates the color depth. The Colorspace (given in line 8) maps the colors to the RGB values they are painted in.

B. SSS Scheme

Shamir's Secret Sharing was proposed by Adi Shamir in 1979 [10]. It is a form of secret sharing where no key is required for encryption and decryption of the secret. The scheme divides the secret into n random looking shares in a way such that any k or more shares can contribute toward the perfect reconstruction of the secret, whereas no information is revealed by getting access to $k-1$ or fewer shares. This is called a k out of n secret sharing scheme, where k is the minimum number of shares required to reconstruct the secret and $2 \leq k \leq n$. To share a secret S among n participants, a polynomial function $f(x)$ of degree $k-1$ is constructed using k random coefficients a_1, a_2 ... $a_{(k-1)}$ chosen from a finite field *GF(q)*, where a_0 is S, and q is a prime number > a_0. The $f(x)$ is given as follows:

$$f(x) = (a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}) mod q \qquad (1)$$

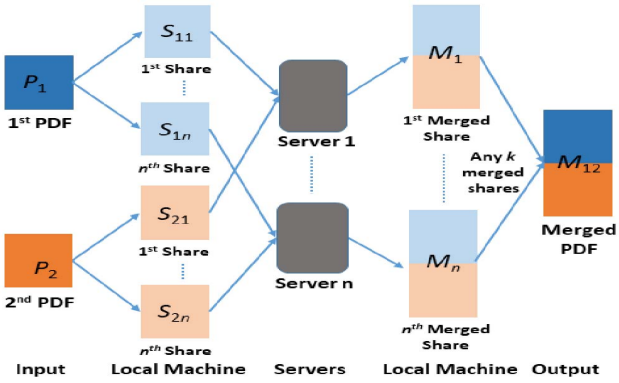

Figure 1: Workflow of the proposed method

The secret can be obtained at $f(0)$, i.e. $f(0) = a_0 = S$ using any of the k shares.

III. Proposed Method

The basic flow of the proposed method is depicted in Figure 1. In the proposed method, we secure the PDF files by obfuscating the information into random shares. Every element of the PDF, be it text, images or tables, is encrypted into n shares using the SSS scheme. Thereafter, the shares of the input PDFs are uploaded to different servers, where they are merged with their corresponding shares. To reconstruct the PDF at the user end, any k merged shares are downloaded and Lagrange interpolation is used on them to compute the decrypted merged PDF. The detailed steps are described as follows.

A. Content Extraction and Share PDF Generation

As described earlier in section II-A, the content of a PDF is embedded in the form of objects. To extract the text and image objects from a PDF, we have used a third party library. Each PDF page is imported into a pdfImportedPage object, which consists of a series of objects (text object, image object or path object). Each text object provides essential information about the text, such as height, width, font name, font size, display bounds and stroke color. Each image is extracted in the form of a bitmap, along with other information such as its width and height. The extracted text and image data is used to create n text and image random shares using the (k, n) SSS scheme. In the case of text, each character of the text is used to create share characters, which in turn creates the share text. Share texts are used to draw the share PDFs using the position, font type and size information from the original PDF. Similarly, each pixel of the image is used to create the share pixel, which eventually contributes to making the share image. These share images are drawn in the share PDFs, keeping all other information intact.

For each PDF to be merged, n shares of each input PDF are created using equation (1) (refer to section (II-B)), and the corresponding shares of the PDFs are sent to different servers for merging. Figure 1 illustrates the share creation of two PDFs (P_1 and P_2) into n shares $(S_{11}, S_{12}, \ldots, S_{1n})$ of P_1 and n shares $(S_{21}, S_{22}, \ldots, S_{2n})$ of P_2. Algorithm 1 outlines the steps in the content extraction and share generation process.

Algorithm 1: Content Extraction and Share Generation
Input: Secret PDFs $P_1, P_2, \ldots, P_m$
Output: Share PDFs S_{i1}, S_{i2} . . . S_{in}
Description:

```
for each page of P_i, 1 ≤ i ≤ m do
    objects[] = extract all objects from the page
    for each j^th object do
        if objects[j] is Text object then
            shares[] = shareTextSSS(objects[j].Text, n, k)
            for a = 1 to n do
                Draw shares[a] in blank PDF S_i1, S_i2, ..., S_in
            end for
        end if
        if objects[j] is Image object then
            shares[] = shareImageSSS(objects[j].Image, n,
            k)
            for a = 1 to n do
                Draw shares[a] in blank PDF S_i1, S_i2, ..., S_in
            end for
        end if
    end for
end for
```

As outlined in Algorithm 1, each input PDF is processed page by page, and each page is processed object by object. The shares of Text and Image objects are created separately using the functions shareTextSSS and shareImageSSS, respectively. In the shareTextSSS function, each extracted text object is processed character by character. Each text object is used to create n share text objects. These share objects are then drawn to the share PDFs by copying all the other information of the text such as font type, font size, position, etc. Here, we have used a prime number $q = 127$ (refer to equation (1)), assuming that the characters extracted from the PDF will lie in the range of [0, 127]. After applying the SSS scheme on each character, we shifted its value with a constant value, 33. This is done because 0-31 ASCII values are non-printable characters, and if used in drawing a PDF, we would not be able to extract the same ASCII code, as these values are not recognizable by the PDF document. Also, we have mapped the extended ASCII values that are greater than 127 to the printable characters. The character value 32 (blank space) is also treated as a special case, as it plays a great role in defining the structure of the PDF. In the shareImageSSS function, each image object is processed as bitmap. The bitmap extracted from each image object

is used to create n share image bitmaps using the SSS scheme. In this case, the prime number q (refer to equation (1)) is chosen as 251, which is the nearest prime number lower than 255, which is the maximum RGB value of a pixel . The pixels having values in the range [252-255] are reduced to 251. This reduction does not create any perceptual loss.

B. Share PDFs Merging

As discussed in section 3.1, n shares are created for each input PDF. The corresponding shares of the two input PDFs, i.e. S_{11} and S_{21}, are sent to Server 1, S_{12} and S_{22} are sent to Server 2, and so on. On the servers, these shares are merged together and the merged shares M_1 and M_2 are obtained, as depicted in Figure 1. In our case, the merging of shares is performed using freely available online sites such as pdfmerge.com, smallpdf.com, etc. Each set of shares is merged on a different server to ensure security, as servers with less than k shares cannot reconstruct the secret data. The detailed steps are given in Algorithm 2.

Algorithm 2: Share PDF Merging
Input: Share PDFs for m input PDFs, $S_{1i}, S_{2i}, \ldots, S_{ni}$, $1 \leq i \leq m$
Output: Merged Share PDFs M_i, $1 \leq i \leq m$
Description:

1: Upload the input share PDFs by creating HTTP requests
2: POST request to the n online servers (e.g. pdfmerge.com) by using HTTP clients
3: $M_i = S_{1i} o S_{2i} o \ldots o S_{ni}$, $1 \leq i \leq m$; o denotes merge

C. Secret Merged PDF Reconstruction

To obtain the merged secret PDF, a minimum of k merged shares out of n are downloaded and Lagrange interpolation method is used to combine them. The detailed steps are given in Algorithm 3.

Algorithm 3: Merged Secret PDF Reconstruction
Input: Any $k \leq n$ merged PDF shares M_i, $1 \leq i \leq k$
Output: Merged Secret PDF $M_{12\ldots m}$
Description:

1: Download any k merged share PDFs M_i, $1 \leq i \leq k$
2: **for** each page of M_i **do**
3: objects[] = extract all objects from the page
4: **for** each j^{th} object **do**
5: **if** objects[j] is Text object **then**
6: shares[] = Extract all corresponding text shares from the merged PDF shares
7: recText = reconTextSSS(shares[],n,k)
8: Draw recText object in blank PDF($M_{12\ldots m}$)
9: **end if**
10: **if** objects[j] is Image object **then**
11: shares[] = Extract all corresponding image shares from the merged PDF shares
12: recImage = reconImageSSS(imageShares[], n, k)
13: Draw recImage object in blank PDF($M_{12\ldots m}$)
14: **end if**
15: **end for**
16: **end for**

In Algorithm 3, k merged share PDFs are used for reconstruction. The merged share PDFs are also processed page by page and each page is processed object by object. The corresponding objects from each share PDF are extracted and decrypted to get the original data (text/image). Here, text and image objects are processed separately using functions reconTextSSS and reconImageSSS, respectively. These functions are used for calculating the reconstructed character ASCII values (in the case of a text object) and RGB pixel values (in the case of an image object). These reconstructed objects are then drawn in the resultant merged PDF. Figure 1 illustrates the reconstruction of the merged PDFs M_1 and M_2 to obtain the merged secret PDF M_{12}.

D. Security Analysis

The security of the proposed method is leveraged by the information theoretic security of the SSS scheme. It is known that the SSS scheme is a partially homomorphic scheme, which means that it supports addition and scalar multiplication operations on the encrypted data, i.e. shares. In our case, we will merge two PDFs, P_1 and P_2 using the proposed method. First, we create the shares of P_1 and P_2, i.e. $S_{11}, S_{12}, \ldots, S_{1n}$ and $S_{21}, S_{22}, \ldots, S_{2n}$. Next, we perform the merge operation on the shares as: $M_1 = S_{11} o S_{21}, M_2 = S_{12} o S_{22}, \ldots M_n = S_{1n} o S_{2n}$. Finally, we reconstruct the secret merged PDF M_{12} using any k merged shares. Here, we provide two lemmas to demonstrate that the SSS scheme is merge homomorphic and the proposed method possesses information theoretic security.

Lemma 1. *The SSS scheme is merge homomorphic, i.e. E(P_1) o E(P_2) = E($P_1 o P_2$), where E represents the encryption performed via SSS.*

Proof: Due to the properties of the SSS scheme, the shares of P_1 and P_2 (obtained by performing E on them), i.e. $S_{11}, S_{12}, \ldots, S_{1n}$ and $S_{21}, S_{22}, \ldots, S_{2n}$, respectively, are information theoretically secure, which also implies that they are statistically independent from each other. Further, if the pairs of shares (S_{11}, S_{21}), (S_{12}, S_{22}), …, (S_{1n}, S_{2n}) are statistically independent, their merged values $S_{11} o S_{21}$, $S_{12} o S_{22}$, …, $S_{1n} o S_{2n}$ will also be statistically independent, since the merge operation just concatenates one after another and does not change their values. Therefore, the merged shares are information theoreti-

Table II: Computation time and size for the resultant merged PDFs

Input PDFs (KB)	Content type	Share creation time (in ms) (n=2)	Merging time per share(in ms)		Reconstruction time(in ms) (k=2)	Merged PDF size (KB)	
			Server 1	Server 2		Proposed method	Traditional method
88, 89	Text	927	4259	4803	5753	176	176
219, 217	Text	1108	4674	5709	10518	165	441
219, 503	Text	7257	14844	80390	87959	1661	726
503, 503	Text	12794	27308	57866	163294	3145	1007
96,83	Image	1703	13349	23765	2501	764	178
96,467	Image	23849	54896	153408	31072	3634	561
194,170	Image	9846	25321	97299	15409	1871	363
467,433	Image	14119	76494	205043	23374	3541	897
19,839	Text and Image	10596	13868	55351	11172	968	855
147,390	Text and Image	14602	26754	98666	40310	2366	544
196,187	Text and Image	10543	14028	13897	23192	868	380
4541,390	Text and Image	48592	64537	64399	125075	6260	4941

cally secure, which proves that the SSS scheme is merge homomorphic. ■

Lemma 2. *The proposed method possesses information theoretic security.*

Proof: As the proposed method performs the merge operation using the SSS scheme, which is information theoretically secure and merge homomorphic as per Lemma 1, it possesses information theoretic security. ■

IV. Experimental Results

We implemented the PDF secret sharing method using CSHARP on a 2.40GHz i5 CPU with 8GB RAM and developed a Windows based application. The application[2] allows a user to choose two PDF files, create their shares, upload them to two different servers for merging and download the merged PDF file. In our application, we have chosen the parameters k and n as 2 and 2 respectively, but the proposed method allows us to choose any value of k and n. However, the greater the value of n, the more third party PDF merge servers are required. The communication between the application and the servers is enabled by calling the Web APIs (online servers) using HTTP client (POST request). As we adopted the (2,2) SSS scheme, we used the following two servers for merging two sets of share PDFs: www.pdfmerge.com and www.sodapdf.com.

The proposed method and the application have been tested on a variety of PDF files and the results for a few of them are listed in Table II, categorized by the type of the content (text, image or both) in the PDF files. As detailed in Table II, when we input two text PDF files of 88 KB and 89 KB each, the total share creation time is around 927 ms. The merging time of corresponding shares of both input PDFs is around 4259 ms (Server1) and 4803 ms (Server2). The overhead of secret merged PDF reconstruction was the greatest as it took around 5753 ms to provide us with the output PDF. The data clearly suggests that the complexity of reconstructing the secret is relatively higher than that of creating secret shares. It is observed that the computational overhead of the proposed method is mainly comprised of the share creation and secret reconstruction time, which is marginal and can very well be acceptable for the security of data. Since the proposed method is applied at the content level, of each character in the case of text and each pixel in the case of images, the processing time is directly proportional to the content size present in the PDF sample. As can be seen in the tables, the larger the number of characters in the text blocks, and the larger the size of the image blocks (greater number of pixel values) in a PDF document, the greater the processing time. It is also observed that the proposed method provides the merged PDFs of larger size compared to the traditional online PDF merge sites. A possible reason for this overhead could be that these sites employ advanced PDF libraries that provide better compression. We intend to address this issue in future. The processing time to merge at each server is also recorded, as reported in Table II. Note that the merge operation at two different servers can be done in parallel to improve the overall time of the operation.

PDF content could have correlating adjacent samples, and the use of random coefficients as a blinding factor in equation (1) to generate the shares eliminates this correlation. Thus, individual shares do not reveal information about the secret PDF file. The security of the data is ensured as only these shares are uploaded to the servers for merging; the secret creation and secret reconstruction are done at the user end.

Figure 2 shows some samples of the original individual share, the merged share and the reconstructed PDF documents. Two input PDFs are converted into two shares of each. Figures 2(a) and 2(c) show the two input PDF

[2]A demo can be found on: [https://youtu.be/UMOMhQiT928]

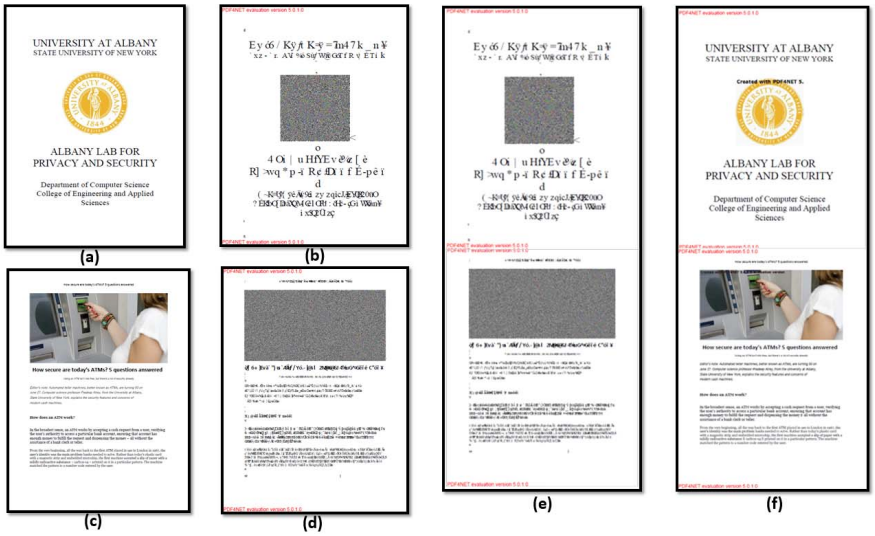

Figure 2: Output for different stages of the proposed method

files, and their first shares are shown in Figures 2(b) and 2(d) (only one share of each PDF shown for brevity). The corresponding shares of each input are uploaded at the different servers to perform the merge operation. The merged share, which is obtained by concatenating the first shares of the two input PDFs at one of the two servers, is downloaded and is shown in Figure 2(e). When the two merged shares are combined at the user end, we get the resultant merged PDF document, as shown in Figure 2(f). The extra text in the share and merged PDF files (PDF4NET evaluation version 5.0.1.0) is the watermark stamped by the PDF parsing library, as we used its free trial version, and it can be easily removed with the use of the purchased version of the library. With that one exception, it is quite evident from the results that we are able to achieve a perceptually acceptable reconstructed merged PDF.

V. Conclusion and Future Work

We demonstrated that merging two or more PDF documents can be performed in encrypted domain at the third party untrusted servers. The idea was to use the SSS scheme to encrypt the content of the PDF files before uploading them to the online web services that provide PDF merging. The adoption of the SSS scheme ensures information theoretic security. Future work would aim to make the scheme robust against the cross-linkage attack and to optimize the application using some concepts of threading to make it more efficient.

References

[1] I. A. Mattoo, "Security issues and challenges in cloud computing: A conceptual analysis and review." *International Journal of Advanced Research in Computer Science*, vol. 8, no. 2, pp. 46 – 48, 2017.

[2] A. V. Subramanyam, S. Emmanuel, and M. S. Kankanhalli, "Robust watermarking of compressed and encrypted JPEG2000 images," *IEEE Transactions on Multimedia*, vol. 14, no. 3, pp. 703–716, 2012.

[3] M. Sudha and C. Thanujat, "Randomly tampered image detection and self-recovery for a text document using shamir secret sharing," in *IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology*, Bengaluru, India, 2017, pp. 688–691.

[4] P. Singh and B. Raman, "Reversible data hiding based on Shamir's secret sharing for color images over cloud," *Information Sciences*, vol. 422, pp. 77 – 97, 2018.

[5] Y. Liu, L. Chen, M. Hu, Z. Jia, S. Jia, and H. Zhao, "A reversible data hiding method for H.264 with Shamir's (t, n)-threshold secret sharing." *Neurocomputing*, vol. 188, pp. 63 – 70, 2016.

[6] N. M. Yoeseph, F. A. Purnomo, B. K. Riasti, M. A. Safiie, and T. N. Hidayat, "Steganography on multiple MP3 files using spread spectrum and Shamir's secret sharing," *Journal of Physics: Conference Series*, vol. 776, no. 1, pp. 012–089, 2016.

[7] A. M. Yakubu, N. C. Maddage, and P. K. Atrey, "Securing speech noise reduction in outsourced environment," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 13, no. 4, p. 51, 2017.

[8] T. Hassan, "Graphwrap: A system for interactive wrapping of pdf documents using graph matching techniques," in *The 9th ACM Symposium on Document Engineering*, Munich, Germany, 2009, pp. 247–248.

[9] J. Tiedemann, "Improved text extraction from pdf documents for large-scale natural language processing," in *Computational Linguistics and Intelligent Text Processing*, Kathmandu, Nepal, 2014, pp. 102–112.

[10] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.