Protection of Integrity and Ownership of PDF Documents using Invisible Signature

Imad Fakhri Al Shaikhli*, Akram M. Zeki†, Rusydi H. Makarim*, Al-Sakib Khan Pathan*
*Department of Computer Science, †Department of Information System
Faculty of Information and Communication Technology, International Islamic University Malaysia
Gombak, Selangor, Malaysia
imadf@iium.edu.my

***Abstract*—Distributing digital document through insecure channel leads to the question of the integrity and the ownership of the file on the receiver's side. Employing hash function and digital signature algorithms solves the difficulties of the receiver to proof the approval of the document from the sender as well as detecting if the file has been altered by illegitimate parties. In this paper, a method to attach digital signature in Portable Document File (PDF) as invisible watermark is proposed. We show that the space character that splits the field in cross-reference table can be overwritten without corrupting the file. The method to embed and extract the signature are proven to be efficient, since the offset index for cross-reference table are available at the end of the file. Our proposed method to embed the signature in the PDF file does not increase the size of the file and offer flexibility for the user to implements the digital signature scheme of his/her own preference.**

I. Introduction

Digital watermarking is a process of embedding information to a data that describe the ownership or the authenticity of it. The main focus of digital watermarking is on aspects of copyright protection, to protect the intellectual property from various threats that can have direct or indirect effects. It differs from steganography on its purpose (steganography is a practice of altering the data to embed a secret message).

Digital watermarking has been widely implemented in various format, such as images, sound files, and document format. Some of the scheme are implemented as visible watermark for ownership protection[1][2][3][4]. Several other scheme are belongs to the category of invisible watermark [5][6][7][8][9]

In this paper, we propose a new invisible watermark method to embed digital signature in portable document format (PDF) that does not increase the size of the file. The structure of PDF file is explained in the second section. In section 3, a brief explanation about digital signature algorithm is given. Section 4 is the description of the method we use to embed and extract the signature in the PDF file. The proof of concept of the method is presented in section 5 using an example. Last section is the conclusion, and the discussion of several open problems.

II. Structure of PDF Document and File

Portable document format (PDF) is an open-standard, initiated by Adobe, to provide cross-platform compatible document format. It encapsulates complete description of document, including the graphics, text, and other information need to display it.

A PDF file consists of a collection of objects that construct the PDF document with associated structural information. There are eight (8) basic types of objects in PDF documents : boolean values, numeric objects, strings, names, arrays, dictionaries, streams, and the null objects.

Boolean objects identified by keyword true or false. It can be used as the values of the array elements as well as dictionary entries.

Numeric objects consist of two main types, namely integer objects and real objects. Integer objects is representation of mathematical integer number, whereas real objects approximate mathematical real numbers, with limited range and precision. Integer numbers are written as one or more decimal digits, optionally preceded by sign, and possibly converted to real objects if it exceeds the implementation limit. A real value is written in the similar fashion with integer, with an optional sign, a leading, trailing, or embedded period. Unlike the integer objects, if the value of real objects exceeds the implementation limit, an error will occurs.

String objects is a sequence of bytes, where each bytes is an unsigned integer value between 0 and 255. String objects can be written as sequence of literal characters enclosed with parentheses. It can also be written as hexadecimal data enclosed with angle brackets.

Name objects is uniquely defined as a sequence of characters. It is unique in the sense if there are two or more objects made up of the same name, they are identically the same objects. Name object is also case-sensitive, uppercase and lowercase character are considered different.

Array object is a one-dimensional collection of objects, that written as a sequence enclosed in square brackets. The PDF arrays may consist of more than one type of objects. The array elements can be in any combination of strings, integer objects, dictionaries, or other objects. In order to construct array with multiple dimensions, it can be created by putting array objects as elements of other array objects, nested to any depth.

A dictionary object is an associative tables consist of dictionary entries, where each entry contains a pair of objecs, called key and value. They key must be type of name

objects, and it must be unique in the same dictionary. The value in the dictionary can be any kind of objects. Similar with array objects, a dictionary may exist inside another dictionary. Note that dictionary key must be distinct in the same dictionary. If the key appear more than once, its value is undefined.

Objects such as images and page descriptions are represented as stream objects. It is an unlimited sequence of bytes that the representation is determined by the context in which the stream is referenced. A stream object consists of a dictionary followed by zero or more bracketed between the keywords *stream* and *endstream*

The null objects is the type of object that does not equal with any other objects, and denoted by keyword *null*.

A. Structure of PDF File

General structure of PDF file is divided into 4 main parts, consist of header, body, cross-reference table, and trailer. The header describes the version of PDF specification used in the file. The format of the header follows `%PDF-X.X` where `X.X` represents the specification version. For instance, a PDF file that conforms with specification 1.7 will have `%PDF-1.7` in the first 8-byte as header.

The body of PDF file is the largest sub-structure that contains a sequence of indirect objects representing the whole content of the document. The objects are the components of the document such as images, fonts, texts. The low-level implementation of these objects are based on the object type described in the previous section, such as dictionary objects, array objects, or stream objects.

Cross reference table is the only component in PDF file with fixed format, that allows the entry in the table to be accessed randomly. It contains information that allow random access to the indirect objects in the file. Each indirect object is specified by one-line entry specifying the offset of the object within the body of the file. This makes any PDF consumer does not need to read the entire file to locate any particular objects.

```
xref
0 6
0000000003 65535 f
0000000012 00000 n
0000000054 00000 n
0000000000 00007 f
0000000225 00000 n
0000000501 00000 n
```

Figure 1. Example of cross-reference table with one section

Every cross-reference section starts with a keyword **xref**. Initially, there would be only one section in the entire table. However, the content of cross-reference table may also consist of more than one cross-reference sections, that will be added each time the file is updated. Each subsection starts with a line containing two positive integer number separated by one space character. The first number is the object number of the first object in the subsection and the second number tells the number of entries in the subsection. Every cross-reference entry is written in one line and has 20 bytes length including the end of line marker. The entry consist of 3 fields separated by space character. The first field is a 10-digit byte offset padded with leading zeros if necessary. It tells the number of bytes from the beginning of the file until the beginning of objects. The second field is the 5-digit generation number. Following the generation number, is a one-byte flag for an in-use object or free object, denoted by **n** or **f**, respectively.

The last part of the PDF file is the trailer. It enables any PDF reader application to quickly find the offset of cross-reference table and certain special objects. Every PDF consumer application must read the PDF file from the end of it. The first part of the trailer is the trailer dictionary. It contains a series of key-value pairs enclosed in double angle brackets. Each key entry in the trailer dictionary may contain keywords *size*, *prev*, *root*, *encrypt*, etc (please refer to [10] for further description). The structure of trailer dictionary will be followed by keyword *startxref*, at the line after the closing angle brackets. *Startxref* is used to indicate the location of the *xref* keyword in the last cross-reference section. This will be followed by the end-of-line marker, *%%EOF*, at the last part of the PDF file.

III. Digital Signature

The concept of digital signature is analogous to the physical signature, to prove that the owner itself approves the document. Digital signature scheme is similar to public-key encryption, it involves public and private keys and the algorithm that uses these keys.

Let $\mathcal{D}$ denote the digital document, K_{priv} and K_{pub} are the private key and the public key, respectively. Digital signing is a function on the sender's side that takes K_{priv} and $\mathcal{D}$ to produce the digital signature $\mathcal{S}$ associated with $\mathcal{D}$ ($\mathcal{S} = Sign(K_{priv}, \mathcal{D})$). The receiver does the verification process, the inverse of sign function, that takes the K_{pub}, $\mathcal{S}$, and $\mathcal{D}$ in order to verify the ownership of $\mathcal{D}$.

The security of digital signature are based on hard mathematical problem. Given K_{pub}, it must be computationally infeasible for an attacker to determine K_{priv} associated with K_{pub}, or any other private key that produces the same signature as K_{priv}. On the other hand, given K_{pub} and a set of signed documents $X = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$ together with a set of the signatures produced from all the element in X, $Y = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$, an attacker must not be able to determine a valid signature on any document $\mathcal{D}$ where $\mathcal{D} \notin X$

Most of digital signature schemes are only able to sign small amount of data. Thus, signing large amount of data is

an inefficient way, since the size of the signature can be as large as the document itself and it consumes high computational resources. In practice, the solution for such problem is to use one-way hash function, that computes a fixed-length k-bit hash-value associated with a digital document in arbitrary size. $(H : \{0,1\}^* \rightarrow \{0,1\}^k)$. The signature function then takes the hash value of $\mathcal{D}$ and the K_{priv} to obtain the signature ($\mathcal{S} = Sign(H(\mathcal{D}), K_{priv})$). The receiver will then perform the verification algorithm and check if the hash value computed from the verification function is match with the hash value of the digital document.

Various digital signature schemes are available publicly such as RSA[11], ElGamal[12], NTRU[13]. In this paper, we use Digital Signature Algorithm (DSA) with SHA1 based on the standard that NIST propose[14].

A. Digital Signature Algorithm (DSA)

DSA is variant of ElGamal digital signature with shorter signature size. Sender will choose two large prime numbers, p and q that are relatively prime, $p \equiv 1 \pmod q$ and an element $g \in \mathbb{F}_p^*$ of exact order q. Sender then pick a secret exponent s and compute $v \equiv g^s \pmod p$. The public key parameter will consist of p, q, g, and v.

When the sender wants to sign a digital document $\mathcal{D}$, $1 \leq \mathcal{D} \leq q-1$, he must choose a random number e in the range $1 \leq e \leq q-1$ and computes $L_1 = (g^e \bmod p) \bmod q$ and $L_2 = (\mathcal{D} + sL_1)e^{-1} \pmod q$. L_1 and L_2 are the signatures associated with document $\mathcal{D}$.

The verification algorithm takes L_1 and L_2 to compute $V_1 = \mathcal{D}L_2^{-1} \pmod q$ and $V_2 = L_1 L_2^{-1} \pmod q$. The verification process will check if $(g^{V_1} v^{V_2} \bmod p) \bmod q$ is equal to L_1.

It is not hard to proof the correctness of DSA. The verifier computes $(g^{V_1} v^{V_2} \bmod p) \bmod q$ which is equivalent to $(g^{\mathcal{D}L_2^{-1}} g^{sL_1 L_2^{-1}} \bmod p) \bmod q \equiv (g^{(\mathcal{D}+sL_1)L_2^{-1}} \bmod p) \bmod q$. Since $L_2 \equiv (\mathcal{D} + sL_1)e^{-1} \pmod q$, thus $(g^{(\mathcal{D}+sL_1)L_2^{-1}} \bmod p) \bmod q \equiv (g^e \bmod p) \bmod q = L_1$

IV. Proposed Method

The property of cross-reference table in PDF file structure, that its the only fixed format across the PDF file, may lead to possibility of existence of redundant space. Based on the description of cross-reference table structure, there are two space characters that splits the three fields (offset, generation number, and flag) in each entry and one space character that marks the end of the line. Our initial observation tells that if a user overwrite the space characters, the PDF consumer would still be able to read the file properly.

Assume that the PDF file is never been updated incrementally, that there is only one cross-reference table subsection at the end of the file. Given n, the length of the signature, and l, the number of entry in the cross-reference table. If $n \leq 3l$, then the signature's bytes can be embedded sequentially in each of the space characters available in every entry without increasing the size of the PDF file. In total, the algorithm will traverse $\lceil n/3 \rceil$ entries in the cross-reference table.

The algorithm to encode the signature in the cross-reference table can be described as follows : the encoder will read the PDF file from the end-of-file to get the offset of the cross-reference table, it will then go to the offset mentioned and embed the signature in each cross-reference table entry.

Algorithm 1 Algorithm to embed the signature

Input: PDF file F and private key K_{priv}
Output: Signed PDF file

```
D = Read(F)
S = Sign(H(D), K_priv)
xref_offset = getStartXrefOffset(D)
c = 0
for i = xref_offset to len(D) do
  if c ≥ len(S) then
    break
  else
    if D[i] = 0x20 then
      D[i] = S[c]
      c = c + 1
    end if
  end if
end for
return D
```

The function *getStartXrefOffset()* does backward searching starts from the end of the PDF file to get the offset for cross-reference table. The procedure will then find the available space character in the cross-reference table and overwrite it with the value of the signature.

The process of decoding the signature from the PDF file can be done in similar fashion. The procedures will execute *getStartXrefOffset()* to get the offset of cross-reference table. Since the receiver knows the length of the signature, n , it only needs to iterate through $\lceil n/3 \rceil$ entries in the table. For every offset that indicates the beginning of the entry in cross-reference table, the procedure will check if there is a signature's byte in the 10, 16, and 18 bytes after the entry's offset. This is due to the length of byte offset, generation number, and the flag. The entry's offset will then incremented to the next 20 bytes to check the next entry in the table.

The method to encode and decode the signature to the PDF file does not consume high computational resources. The encoder does not need to read the entire file in order to embed or remove the signature from the file itself. The complexity of the function is depend only on the size of the signature, thus it proves that the algorithm is in linear complexity class.

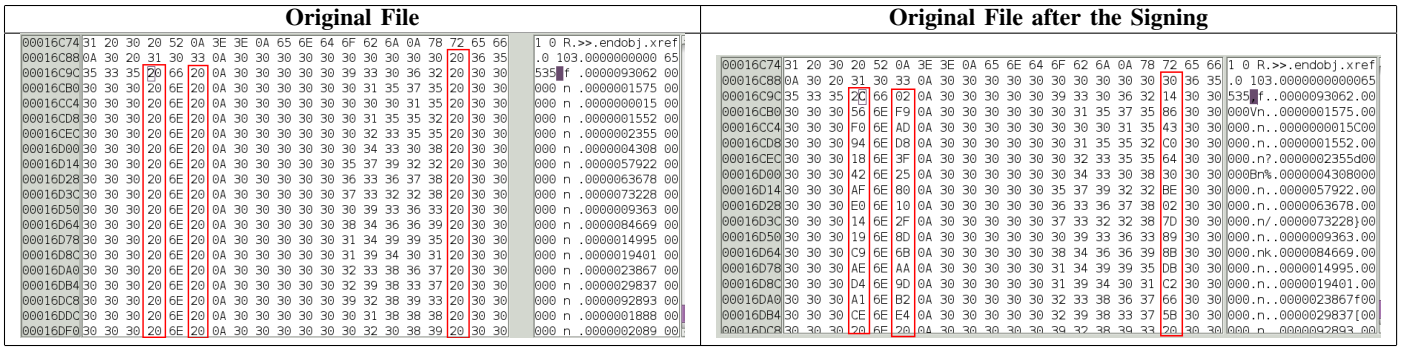

Table I
THE RESULT OF THE PDF FILE BEFORE AND AFTER THE SIGNATURE IS EMBEDDED

Algorithm 2 Algorithm to decode and verify signature

Input: A signed PDF file F and public key K_{pub}
Output: The validity of the signature

```
D = Read(F)
xref_offset = getStartXrefOffset(D)
S = []
while True do
  if D[xref_offset + 10] ≠ 0x20 then
    S.append(D[xref_offset + 10])
    D[xref_offset + 10] = 0x20
  else
    break
  end if
  if D[xref_offset + 16] ≠ 0x20 then
    S.append(D[xref_offset + 16])
    D[xref_offset + 16] = 0x20
  else
    break
  end if
  if D[xref_offset + 18] ≠ 0x20 then
    S.append(D[xref_offset + 18])
    D[xref_offset + 18] = 0x20
    xref_offset = xref_offset + 20
  else
    break
  end if
end while
return Verify(D, S, K_pub)
```

V. RESULT

We demonstrate the proposed method explained in the previous example using DSA and SHA-1. Although there have been numerous attack on SHA-1[15], the example we provide in this section is only to proof the practicality of the proposed method. User may use other variants of the hash function or the digital signature in the implementation.

We signed a PDF file with size 95467 bytes, contains 103 entries in the cross-reference table. The signature produced has 47 bytes length, embedded to 16 entries in the table. In the attached result, the differences appear and shows all the space character in the original file that replaced by the signature.

Besides maintaining the size of the file, this scheme also offer flexibility for the sender to choose the algorithm of his/her preference. Currently the signature scheme that adobe supports are RSA-1024, RSA-2048, RSA-4096, DSA-4096 with SHA-1, SHA-256, SHA-384, SHA-512, and RIPEMD160[16]. Practically, without using the method that we propose, if the user wants to use other variants of hash function or digital signature scheme, they must wait for adobe to standardize it for the new release of PDF version.

VI. CONCLUSION

In this paper, we introduced a new approach of embedding signature in PDF file as invisible watermark. The user would not notice the differences in the file after it has been signed since the modification is done outside the body of the PDF file. The scheme we proposed does not introduce any additional space in the file and the complexity to encode and decode the signature depends on the length of the signature itself. This scheme offer a feasible solution for the user who wants to use the digital signature scheme that is not standardized by Adobe.

However, some of the open questions left for the improvement. The main open question is to provide an alternative solution when the number of entry in cross-reference table is not enough to store the signature. It is also important to conceal the signature in the PDF file in order to harden the process of altering or removing the signature by illegitimate parties.

REFERENCES

[1] M. Kankanhalli, Rajmohan, and K. Ramakrishnan, "Adaptive visible watermarking of images," in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, vol. 1, jul 1999, pp. 568 –573 vol.1.

[2] B.-B. Huang and S.-X. Tang, "A contrast-sensitive visible watermarking scheme," *Multimedia, IEEE*, vol. 13, no. 2, pp. 60 – 66, april-june 2006.

[3] Y. Hu and S. Kwong, "An image fusion based visible watermarking algorithm," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 3, may 2003, pp. III–794 – III–797 vol.3.

[4] S.-K. Yip, O. Au, C.-W. Ho, and H.-M. Wong, "Lossless visible watermarking," in *Multimedia and Expo, 2006 IEEE International Conference on*, july 2006, pp. 853 –856.

[5] S. Mohanty, R. Sheth, A. Pinto, and M. Chandy, "Cryptmark: A novel secure invisible watermarking technique for color images," in *Consumer Electronics, 2007. ISCE 2007. IEEE International Symposium on*, june 2007, pp. 1 –6.

[6] S. Bandyopadhyay, D. Bhattacharyya, and P. Das, "Hybrid digital-embedding using invisible watermarking," in *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, june 2008, pp. 1881 –1885.

[7] M. Yeung and F. Mintzer, "An invisible watermarking technique for image verification," in *Image Processing, 1997. Proceedings., International Conference on*, vol. 2, oct 1997, pp. 680 –683 vol.2.

[8] A. Huggett and C. Stubbings, "Invisible watermarking for digital video applications and challenges," in *Secure Images and Image Authentication (Ref. No. 2000/039), IEE Seminar on*, 2000, pp. 9/1 –9/6.

[9] T.-Y. Chen, D.-J. Wang, T.-H. Chen, and Y.-L. Lin, "A compression-resistant invisible watermarking scheme for h.264," in *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP '09. Fifth International Conference on*, sept. 2009, pp. 17 –20.

[10] "Pdf reference, sixth edition, version 1.7." [Online]. Available: http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/pdf_reference_1-7.pdf

[11] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, February 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology*, ser. Lecture Notes in Computer Science, G. Blakley and D. Chaum, Eds. Springer Berlin / Heidelberg, 1985, vol. 196, pp. 10–18.

[13] J. Hoffstein, N. Howgrave-graham, J. Pipher, J. H. Silverman, and W. Whyte, "Ntrusign: Digital signatures using the ntru lattice," in *City University of Hong Kong*. Springer-Verlag, 2002.

[14] "Digital signature standard." [Online]. Available: http://www.itl.nist.gov/fipspubs/fip186.htm

[15] X. Wang, Y. Yin, and H. Yu, "Finding collisions in the full sha-1," in *Advances in Cryptology - CRYPTO 2005*, ser. Lecture Notes in Computer Science, V. Shoup, Ed. Springer Berlin / Heidelberg, 2005, vol. 3621, pp. 17–36.

[16] "Digital signatures in a pdf." [Online]. Available: http://learn.adobe.com/wiki/download/attachments/52658564/Acrobat_DigitalSignatures_in_PDF.pdf?version=1.