UNIVERSITY OF TARTU
Institute of Computer Science
Cyber Security Curriculum

Gema Fernández Bascuñana

Method for Effective PDF Files Manipulation Detection

Master's Thesis (30 ECTS)

Supervisors:
Pavel Laptev
Inna Ivask
Raimundas Matulevičius

Tartu 2017

Method for Effective PDF Files Manipulation Detection

Abstract:

The aim of this thesis is to ease the process of detecting manipulations in PDF files by addressing its source code, before having to use other methods such as image processing or text-line examination. It is intended to be a previous step to tackle, which can save a lot of time to examiners and provide them with more proof of manipulations regarding digital file evidence. The result is the construction of a solid and effective method for PDF file investigation and analysis to determine its integrity. To achieve this goal, a study of PDF file anatomy will be conducted firstly, in order to become familiar with the structure and composition of this file format. Afterwards, a series of manipulations performed directly against the file source code will deepen in its secrets and vulnerabilities, and will therefore help in setting the foundations for the method. Finally, a study on the most common types of file manipulations will lead to a set of layouts to which compare the files under investigation and thus, test its veracity, complemented with a quest for specialised open source tools to accomplish this task; a set of validation experiments will complete the work, evaluating the obtained results and stating future lines of work in this field.

Keywords:

Digital forensics, PDF manipulation, computer forensics, file forensics, manipulation detection

CERCS: P170, Computer science, numerical analysis, systems, control

PDF Dokumentide Võltsimistunnuste Tuvastamise Metoodika

Lühikokkuvõte:

Käesoleva magistritöö eesmärgiks on lihtsustada PDF failides tehtud muudatuste tuvastamise protsessi kasutades faili lähtekoodi enne, kui liigutakse edasi teiste meetodite juurde nagu näiteks pilditöötlus. Lähtekoodi analüüs on mõeldud esimeseks sammuks, mis võimaldab säästa palju uurijate aega ning pakkuda rohkem tõestusmaterjali muudatuste tegemise kohta asitõendiks oleva digitaalse faili kohta. Magistritöö tulemusel valmib põhjalik ja efektiivne metoodika PDF failide terviklikkuse uurimiseks ja analüüsimiseks. Püstitatud eesmärgi saavutamiseks õpitakse kõigepealt tundma PDF faili ehitust mõistmaks faili struktuuri ja komponente. Seejärel tehakse ridamisi muudatusi faili lähtekoodis, mis võimaldab süveneda faili varjatud külgedesse ja leida haavatavaid kohti ning millest saadav informatsioon on abiks metoodika aluste paika panemisel. Failide enamlevinud muutmise tüüpide uurimisel saadakse kogum andmeid, millede suhtes hakatakse võrdlema uurimise all olevaid faile ning seeläbi testitakse faili tõepärasust. Lisaks otsitakse vabavaralisi tarkvarasid, millega antud ülesannet lahendada. Töö lõpetatakse kontrollkatsetega, sealhulgas hinnatakse saadud tulemusi ja märgitakse ära tuleviku tegevussuunad antud valdkonnas.

Võtmesõnad:

Digitaalne kohtuekspertiis, PDF failide võltsimine, arvutikriminalistika, failiuuring, autentsuse kontroll

CERCS: P170, Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1 Introduction

Nowadays, the great majority of information exists in digital form, both at professional and personal level, and in terms of information creation, storage and communication. Moreover, everyday less of these activities are made in a printed/paper form, and new generations are already digitally-born, so basic knowledge on how to take advantage of new technologies to put almost completely aside printed and tangible procedures is assumed.

When specifically addressing documentation, not only information is already digitally created, but existing printed/handwritten documents are going under an archiving and back-up process; they are being scanned and converted into digital form to make storage and/or communication easier and more convenient. They are also duplicated and stored in different locations to avoid data loss. One crucial issue regarding this matter is the integrity of the document when performing such actions; the same way paper documents can be forged, digital files are also vulnerable to manipulation. Therefore, the unalterable character of file formats becomes an important issue to consider when deciding how to store/exchange a file.

Portable Document Format is currently considered one of the most ‘secure’ formats against changes made to the document [1], and consequently it is the most trusted and used one when storing and exchanging files. This happens mainly due to the unchanging character of PDF, to which documents can be transformed regardless of the software or operating system that originated or will open them. Nevertheless, it is rather foolish to believe such (or any) format is safe from alterations just because it does not change among viewers.

This fact is most relevant when referring to digital evidence. Everyday more, PDF files are being presented as evidence in trials, and admissibility of these files is clearly conditioned to the veracity of its origin, content and most important, integrity. Unfortunately, PDFs are currently widely manipulated, both in terms of malware embedding and document forgery, and consequently, it is incredibly important for forensic professionals to keep pace and, if possible, to have a head-start on the race that is ensuring the integrity of files.

In order to help in this matter, a study the field of PDF file manipulation detection can suppose an actual contribution to digital forensics world, and make a difference in the way PDF files are analysed for efficient manipulation detection. For this matter, this paper encompasses the whole process for the creation of an effective method aimed towards detecting manipulated PDF files by means of investigating its source code.

Firstly, this process is contextualized, addressing digitisation while focusing on digital evidence, and specifies the forensic analysis best practices and the rules of admissibility they must comply with. Straightaway, we will focus on our format of interest, PDF, strolling through its history and types, and finishing the way in PDF file anatomy; this part is essential, as it will form a basis from which to knit the threads of our method, for which a deep knowledge of PDF syntax and structure is crucial. Furthermore, and in the interest of mastering PDF internals, manual manipulations will be performed; even though these forgeries are not usual in manipulation cases, they will illustrate the secrets and vulnerabilities of this format, providing us with great understanding of the logic followed by PDF files formation. Having already dived deep into PDF file manipulation, the main outcome of this work will be this clear method on how to effectively investigate, analyse and ensure the integrity of PDF files, detecting manipulations and forgeries.

Finally, with the aim of proving and determining the effectiveness of such method, a validation process will be carried out with materials provided by EKEI[1].

[1] Eesti Kohtuekspertiisi Instituut

2 Background

2.1 Digitised documents

We can define digitisation as the process of converting analogue content into digital form, this is, a sequence of ones and zeros that conform a code which is readable by computers [2]. This process has been carried on since the proliferation of computer networks, and it continues until today, even when almost all documents are digitally created. Among its multiple benefits, some are particularly worth stressing, such as accessibility, availability, conservation, space saving or linking, networking and duplicating possibilities [3]

Specifically document digitisation means a great improvement regarding not only efficiency and the advantages mentioned above, but also regarding economic growth; digitisation has had an actual impact on reducing unemployment or even on improving quality of life [4].

The most common way of digitisation nowadays corresponds to scanning physical documents which, of course, creates new challenges when facing criminal activities regarding documents; if it was not harsh enough to detect forgeries in paper contracts, agreements or cheques, digitisation has also brought digital felonious activities with it, which are just a way of adapting crime to the new era. Forgeries and manipulations of documents are in vogue, and ensuring their integrity is crucial in, for example, court cases.

After having stated the relevance and currency of this kind of digital files and before jumping to the specific file format, general concepts about digitised documents and its proper handling must be addressed.

Digital signatures

Nowadays, in an everyday more paperless world, traditional signatures are becoming obsolete; they just do not fit in our current digital lifestyle and daily processes, as it is completely inefficient, slow and delaying having to print digital documents, sign them and scan them again to return them where they came from. Digital signatures not only solve this problem, but also improve the traditional mechanism of hand-written signatures.

A digital signature consists of a pair of keys belonging to a single person or entity: a private key, known only by the owner, and a public one, available for anyone. The digital signature corresponds to a hash of the file encrypted with this private key along with a timestamp of the signature; this is then all added to the document, and for its verification, the hash is calculated again, the signature is decrypted with the public signature and the result hash from this operation is then compared with the one previously calculated: if they match, the integrity of the file is proved. This protocol is called PKI[2], shown in Figure 1, and in order for it to have validity, a Certificate Authority must be the issuer of the key-pair [5].

This process has proven to be secure and highly efficient, and it is being slowly adopted as a general practice. Unfortunately, there is still a high number of documents presented as digital evidence in court whose integrity must be proven, because hand-written signatures are still being widely used. Digital signatures must prevail in the future over traditional signature methods, as this will provoke considerable savings in matters of time and natural resources, and will raise efficiency in a very notable way. Until this is achieved, we must find ways how to handle and deal with digitised documents that have been traditionally signed, and set procedures and mechanisms to prove these documents integrity and validity, in order to help the transition towards an entire digital world without paper bureaucracy.

[2] Public Key Infrastructure

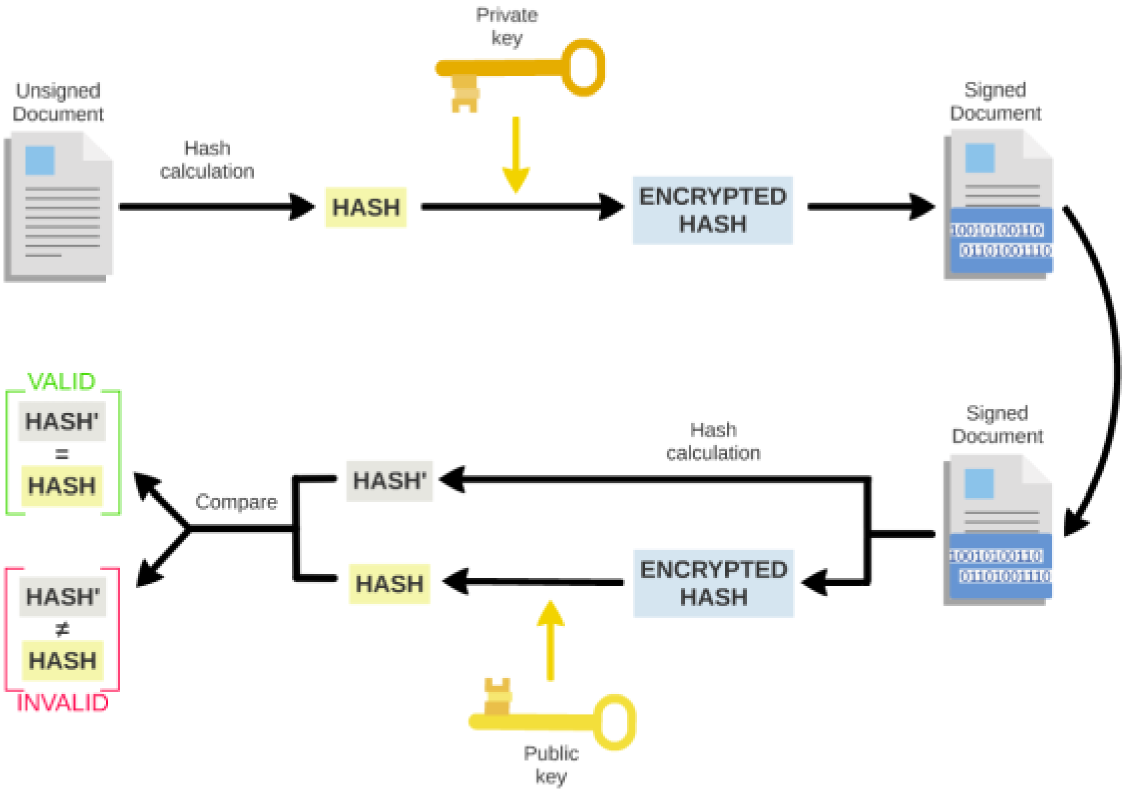


Figure 1. PKI protocol [5]

Forensic analysis best practices

As we are going to handle potential digital evidence, it is important to address which are the best practices to do so. The Scientific Working Group on Digital Evidence[3] specifies a very explicative and clear recommendations when collecting, handling and analysing digital evidence [6]; our great interest resides on the analysis part, as the others are out of the scope of this thesis. According to them, examiners:

- Should be appropriately trained.
- Should review and study any kind of documentation provided to them by the petitioner, so the necessary processes to address the examination can be determined, as well as the legal authority.
- Should avoid conducting the actual examination in the original evidence, but using instead a forensic copy or image.
- Should use appropriate standards and control during the examination process.
- Should perform the examination analysis always logically and systematically complying with the organisational policy.

Regarding the documentation, which is a crucial part of the process, it should contain enough details for another examiner to understand exactly what has been done, repeat the process and obtain the results independently; this is, the documentation allows a process to become repeatable. Moreover, a report with the findings should be delivered at the end. This report should:

- Present all information in a readable and understandable format for non-technical people, and the examiner should be able to properly explain it in its whole.

[3] SWGDE

- Include any pertinent information, regarding the acquisition and/or evidence handling.
- Address the petitioner needs.
- Include and document the scope/purpose of the examination as well as any additional reports related to the examination.
- Provide with a detailed description of the evidence examined, as well as the date when the examination was performed and the examiner's name.
- Be reviewed by a competent person according to organisational policy.

Admissibility in court

Main reason why PDF files and documents in general are being checked for forgery and manipulations is because they are digital evidence which could be used in a trial. To be able to act as such trial evidence, documents and all the procedure involved in their acquisition and examination must comply with certain admissibility rules.

The U.S. Department of Justice has issued a Guide for Law Enforcement [7], which can enlighten us regarding these matters. When addressing evidence examination for court matters, as we have already discussed in best practices, the examiner must be properly trained and the examination should not be conducted on the original evidence, but on a copy.

There is a difference between extraction and analysis; the former includes the recovery of the data from the device and the latter consists in the examination and interpretation of the data. For the purpose of this thesis we should focus on the analysis only.

The analysis of the data has as objective determining its relevance for the case, and different approaches might be taken depending on the requests made:

- *Timeframe analysis*: it can help to link the events occurred in a system with the events regarding the case, understanding and building a timeline. For this matter timestamps are crucial, in our case, regarding the PDF file; creation, modification and accessed time and date should be checked in order to place the history of the file within the timeline of the case.
- *File and data hiding analysis*: files and data in general might be hidden in the system, so an exhaustive search must be done in order to reveal them. Regarding to our concerns, PDF files can also hide data, highlighted as a remarkable aspect for the analysis. Compressed/encoded data should be specially considered, as well as the use of steganography. Moreover, metadata will help us to put the file into context; who is the author, when it was created/modified/accessed, which operating system was used, etc.
- *Ownership and possession*: thanks to the metadata we can find out relevant information such as the author of the file, or if it was modified/manipulated, when was it done and under which conditions (operating system, specific software…).

We can see that identifying the origin and the conditions of the file is very important for its admissibility in court. From the case-law relative to digital evidence and how it is managed [8] we can learn how to proceed when addressing digital evidence for examination, and it turns out that demonstrating that the file is truthful and remains original, results to be decisive in many cases.

2.2 Portable Document Format

Definition

Portable Document Format is, as its own name indicates, a file format intended to maintain the structure and appearance of the documents when presenting, exchanging or simply viewing them, regardless of the hardware, operating system or program involved in their creation or presentation [9]. PDF files have a composed fixed layout, which includes text, graphics, and all information that allows to display it. Interoperability among these files was the prime objective when creating this file format, which is why now it is extensively used.

History

This format was created by the American company Adobe Systems Incorporated, which specialises in software related to presenting and editing different types of files (documents, multimedia, images, videos….). One of the greatest achievements of this company was the creation of PostScript language, used mainly by laser printers.

PDF has its origins in project Camelot, leaded by John Warnock, Adobe's cofounder, which sought for a universal way to present and communicate printed information electronically [10]. Finally, in the year 1992, Adobe presents PDF and just one year later Acrobat, main PDF file viewer, editor and manager, was released.

"*What industries badly need is a universal way to communicate documents across a wide variety of machine configurations, operating systems and communication networks. These documents should be viewable on any display and should be printable on any modern printers. If this problem can be solved, then the fundamental way people work will change*" [10]

And it certainly did; in 1993, Adobe Reader joint with PDF specification were released free of charge, which made a crucial difference as you could freely download and install this tool and view any PDF file you would like, no matter the lower layers. What we know now as PDF format differs considerably from what it was back then, as several improvements and additions were being included in following versions of the format [See Appendix I]. Some of them were the creation of a web browser plugin, which increased its popularity over the internet, the possibility for encryption, a very relevant feature regarding security and integrity assurance, or the easy integration with Microsoft Office software [11], which allows you to transform your otherwise editable and non-consistently presentable file to PDF. Another important introduction of functionality was the ability to introduce JavaScript code, fact that in the current times is mischievously exploited to embed malicious code by attackers who wish the opener of the file to execute malware in a system.

Later in 2001, with the release of Acrobat 5 and PDF version 1.4, a significant change was introduced; until then, metadata could only be addressed within the object /Info, but then they introduced the concept of XMP (Extensible Metadata Platform), which broadened the possibilities for metadata inclusion in PDF files. Another upgrade worth mention is the enhancement of the cross-reference table, which was included in version 1.5 in 2003; they added cross-reference encoded streams, which would speed up considerably the object retrieval within the file.

But it was not until early 2008 when the big step was made. Although PDF 1.7 was not a particularly innovative version release, as it included mainly security and commenting support, with this version the Portable Document Format finally became an official ISO standard, ISO 32000-1:2008. Therefore, this was a very important milestone for developers who wished to understand, learn from and experiment with PDF file internals. Even if file format is free, Adobe has not released as free all their technologies, as some remain proprietary.

Finally, a new version is planned to be released by mid 2017 (PDF 2.0) as well as new specifications (ISO 32000-2:2017).

Types of PDF files

Depending on the use you are going to give to a certain PDF file, there are 8 different types from which you can choose the one that suits you better [12]. Two of them are out of the standard and satisfy field specific needs (like PDF Healthcare), and the other six are:

- *PDF*: this is the general standard, which is meant for office use, sharing and viewing online and if you are looking for standard quality.
- *PDF/A*: this type is thought for long-term file storage, with a restricted set of features such as JavaScript, audio and video content and encryption.
- *PDF/E*: this type is meant to be used for architects and engineers, as it copes with large-format drawings or multimedia.
- *PDF/X*: this one is thought for print and creative professionals or graphic designers, due to its high quality and print-readiness. Another type based on components of PDF/X type is *PDF/VT*, with the main addition of allowing customisation of data within the files.
- *PDF/UA*: Universal Access standard is specially created for people with disabilities, and contains technology that assists users through reading and navigation.

We are mainly going to work with PDF general type and some PDF/A types, as they are the most common in court cases and trials as digital evidence.

2.3 PDF anatomy

In order to be able to immerse ourselves into the depths of PDF file manipulation, first step to address is the investigation of the anatomy of these files. This will give us a basic idea of how PDF files are constructed, structured and secured. All the information in this section has been retrieved from an extensive study of the PDF standard, ISO 32000:2008 [13]

Data Types

With the aim of achieving a better understanding of the format, a quick overview of the syntax and data types of interest for our needs that are used in PDF files is given.

Boolean, *number* and *null* types are used as they normally are; boolean data values correspond to true and false, number objects can be real or integers, and null objects do not correspond to any existent object, being represented with ‘null’.

When you wish to insert a group of bits and confine them altogether, *strings* can be used, of which there are two types; literal, which are delimited by parentheses, can contain a group of characters, while hexadecimal are delimited by angle brackets and contain a set of hexadecimal digits represented as ASCII[4] values, each value defined by two digits. Here we can see equivalent strings represented in both forms:

- *Literal*: `(examplestring)`
- *Hexadecimal*: `<65 78 61 6d 70 6c 65 73 74 72 69 6e 67>`

[4] American Standard Code for Information Interchange

However, the maximum string size is 65535 bytes, so if need more space *streams* should be used. These are used to represent large portions of the document, such as images, and consist of a *dictionary* (explained later in this same chapter) with some normalized entries (such as length or filter) and the actual stream delimited by the tags ‘stream’ and ‘endstream’.

```
<<
  /Length 2 0 R /Filter /DCTDecode
>>
stream
[…]
endstream
```

Different from strings, *names* are a group of characters that identify a specific object; even if two strings contain the same bits, they remain as two different objects, if two names contain the same bits, they refer to the same object. Names are preceded of a slash ‘/’, and the maximum length are 127 bytes, for example: `/Type`

Next object type are *arrays*, which correspond to a set of ordered elements delimited by brackets, although they can only be unidimensional. While usual array types require the elements to be from the same kind, in PDF syntax an array can be composed of elements of any kind (even other arrays), for example: `[12 example -92 /exname (exstring)]`

One relevant data type are *dictionaries*, which are tables that associate element pairs, which correspond to ‘key’ and ‘value’, and have a maximum of 4096 entries. While the key must always of name type, the value can belong to any other data type. Moreover, the key must be unique, not being able to correspond to any other key of any other pair of elements. The importance of this object type resides on the frequency with what they are used in PDF file; they basically conform the main blocks of the documents, joining and relating different objects to conform a bigger element (such as a page). One example of a dictionary could be:

```
<<
  /key1 111
  /key2 false
>>
```

Last object type corresponds to indirect objects; these are linked to a specific object in the document, so other objects can refer to it. To make this reference possible, they are assigned both a ‘object number’ and a ‘generation number’, both positive integers. Object numbers are not necessarily assigned in a sequential order and the generation numbers are always 0 when the object has just been created; this value is incremented if the object is updated. Indirect objects are delimited by the labels ‘obj’ and ‘endobj’ as shown in the example:

```
23 0 obj
  <65 78 61>
endobj
```

In order to refer to them we need an indirect reference, which consists of object and generation numbers followed by ‘R’: `23 0 R`

File Structure

Every PDF file has a common structure, which can be divided in four main areas, shown in Figure 2. When reading PDF from the source code, the correct way to do so is from bottom to top. We can see that PDF file is composed of four main objects:

- *Header*: here we can find only one line, stating that this is a PDF file and the used version, as an example here we have a header with the last version of PDF:

```
%PDF-1.7
```

- *Body*: here we can find the content of the document (images, text, multimedia…); by means of indirect objects, they represent the actual data content that will be shown. Any object can be part of the body.

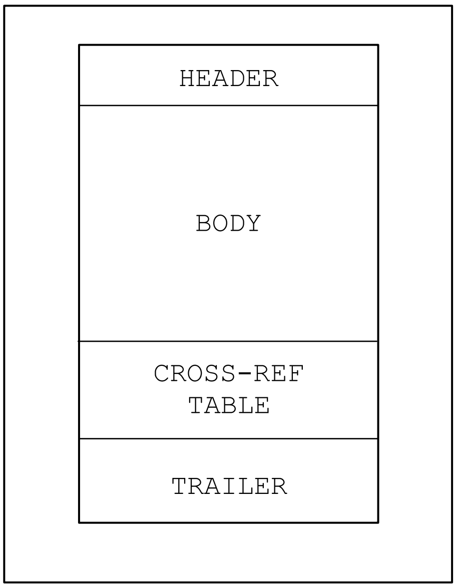
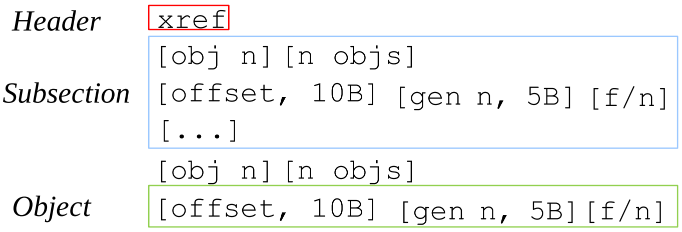


Figure 2. Basic structure of a PDF file

- *Cross-reference table*: this important part of the file structure hosts references and information about all indirect objects on how to locate and access them, so there would be no need of reading the whole file to find a specific object. There might be more than one.
  As we can see in Figure 3, the cross-reference table always starts with the label 'xref'. It is divided in subsections, which contain one or more objects, represented with one line each. Subsections start with a pair of numbers: first one, is the object number (`obj n`), which corresponds to the first object in the subsection, having the following (if there is more than one) a sequential object number. The second number in the pair is the number of object that subsection contains (`n objs`).
  In the example, we can see that in the first subsection there is only one object (object number 0), while in the second subsection there are two (object numbers 10 and 11).
  Furthermore, each object is divided in 3 parts, which mean different things depending on the value of the last part:


Header
xref
Subsection
[obj n][n objs]
[offset, 10B] [gen n, 5B] [f/n]
[...]
[obj n][n objs]
Object
[offset, 10B] [gen n, 5B][f/n]

```
xref
0 1
0000000011 65535 f
10 2
0000023569 00002 n
0000000000 00001 f
```

Figure 3. cross-reference table structure and example

- If the last letter is 'n', it means this object is in use. The first 10 bytes then correspond to the offset of this object from the beginning of the file, and the following 5 bytes are the generation number, which is 0 if the object has not been updated, and increases 1 every time it is freed.

    So, for example, object number 10 (first one in second subsection) is currently used, and it is the second time is being reused, same as saying is used for the third time (as it has been freed twice).

    - If the last letter is ‘f’, it means this object has been freed, and even if it is present, it should not be used. In this case, the first 10 bytes point to the next free object in the table, and the generation number is increased by 1 every time the object changes from ‘in use’ to ‘free’. The first object in the first subsection is free, and it points to object 11 which indeed is the next free object. As there are no more free objects left, object 11 points to the object 0 again. About generation numbers, first object must always have a generation number of 65535, and object 11 has just been freed, so the value is 0001.

  There will be as many subsections as needed in the file, so initially only one subsection will exist, and the followings will be appended containing the new existent objects.

- *Trailer*: it is the last part of the file, and so we would need to address it first when understanding PDF source code. It helps us find both the cross-reference table and other specific relevant objects in the file.

Table 1. Objects in trailer.

Key	Type	Value
/Size	Integer, required	It specifies the number of existent entries in the cross-reference table
/Root	Dictionary, required	Points to the object referencing the Document Catalogue of the file
/Encrypt	Dictionary, required if encrypted	Refers to the document’s encryption dictionary
/Prev	Integer	Offset to the previous cross-reference table, if more than one exists
/Info	Dictionary	References the document’s information dictionary
/ID	Array, required if /Encrypt is present	Consisting of two unencrypted byte-strings that conform a file identifier that allows to check if accessing the right file without decrypting it

An example of trailer would look like:

```
trailer
<<
  /key1 value1 %Objects in the trailer
  /key2 value2
>>
startxref
1111 %Offset to xref table
%%EOF
```

If we interpret the trailer bottom-up, its last line corresponds to the last line of the file itself, and it contains the label '%%EOF'. Previous two lines are the reference to the starting of the cross-reference table: it consists of the label 'startxref' and next the offset in bytes from the beginning of the file until the start of the table. Before that, we encounter a dictionary containing the names of relevant objects and its corresponding values. These specific objects are specified in Table 1.

Incremental updates

The structure of the PDF file has been thought in a way that making modifications does not imply rewriting the whole document. This is achieved by means of the incremental updates, preserving the original state of the file and appending any changes made to the end. While the original structure and objects remain intact, new body, cross-reference table and trailer are appended at the end. There can be more than one update, so more than one of these blocks can exist. These updates only contain entries referring to the objects that have been either created, changed or deleted; in this last case, the object remains in the file but its state is changed to 'free' (f). Added trailers must have the field /Prev indicating the previous cross-reference table and their own %%EOF label.

Metadata

The general concept of metadata can be defined as data about data, this is, information that describes and defines certain data. Regarding PDF files (and files in general), it addresses information about the file that has nothing to do with the content, but with the context of its creation and modification. In order to include such information within the file, there are two main techniques that are currently used.

The first one is the /Info *object*. It is declared in the trailer as an indirect reference in the trailer `(/Info 1 0 R)`, and it is composed of a document information dictionary, which contains the metadata of the file. All attributes inclusion is optional, and they are specified in the form of /Name and corresponding Value. The possible attributes that can be used are */Title, /Author, /Subject, /Keywords, /Creator, /Producer, /CreationDate, /ModDate* and */Trapped*. One sample */Info* object would look as shown here:

```
1 0 obj
<<
      /Title (Example)
      /Creator (TeX)
      /Author (GEMA)
      /Producer (pdfTeX-1.4)
      /CreationDate (D:20170219133028-03'00')
      /ModDate (D: 20170219133601-03'00')
      /Keywords (sample)
>>
endobj
```

The most remarkable attributes for us have been included in the example. Special attention must be drawn to the creation and modification dates, whose format corresponds to:

`D:YYYYMMDDHHmmSS(deviation with UT`[5]`)`

The second one corresponds to XMP object. The Extensible Metadata Platform is an XML[6]-based format file labelling technique [14] which allows to include metadata in an organised

[5] Universal Time

[6] Extensible Markup Language

standardized form, and is more resistant to changes than */Info* object method. Properties about the file are organised in schemas, some of them are predefined, but they can be customized as well. An XMP packet is embedded in a PDF object, that can be indirectly referred. All of this is extensively explained in XMP Specification, Storage in Files [15]. The structure of an XMP packet includes:

- *Header*: it contains information about the packet itself. It has two attributes that must always be included, 'begin' and 'id', and this is the way it looks:
  - `<?xpacket begin=" " id="W5M0MpCehiHzreSzNTczkc9d"?>`

  Attribute 'begin' value shall be always an "Unicode zero-width nonbreaking space character U+FEFF" [15], which acts as a byte-order maker. The attribute 'id' must always have the value 'W5M0MpCehiHzreSzNTczkc9d'.
- *XMP data*: the content of the packet is included here, encoded as specified in the 'begin' value of the header. The attributes specified in here may vary, and can follow both a predefined or customized schema. Regarding PDF files, one of the most followed schemas is Dublin Core (dc) Metadata Element Set [16]. When using it, the attributes 'URI' and 'prefix' are fixed, and most common properties are, for example, date, creator, description or title.
- *Padding*: between 2-4KB of padding are included in the packet, in case it needs to be extended in the future, so it will not affect the rest of the file.
- *Trailer*: it indicates the end of the packet. It has one only attribute, 'end', whose values can be either 'r'(read-only) or 'w'(writeable) and it looks like this:
  - `<?xpacket end='w'?>`

When embedding an XMP packet in an PDF file object, its XML code is included within a */Metadata* object; this packet is specified as a stream, due to its length and density. Normally some space is left between the end of the XML code and the end of the stream, in case some information is added afterwards, as stated. Here is an example with just one dc attribute:

```
1 0 obj
<< /Type /Metadata /Subtype /XML /Length 100 >>
      stream
            <?xpacket begin='' id='W5M0MpCehiHzreSzNTczkc9d'?>
                  <dc:creator>
                         <rdf:Seq>
                            <rdf:li>GEMA</rdf:li>
                         </rdf:Seq>
                  </dc:creator>
            <?xpacket end='w'?>
      endstream
endobj
```

The existence of such object is indirectly referred in /Root object, in the Document Catalogue of the file `(/Metadata 1 0 R).`

Document Structure

The document structure of a PDF file consists in a series of objects hierarchically organised within the body. These objects, most of them of dictionary type, are sorted out in page objects, which are organised in a page tree, specified in the document catalogue, as shown in Figure 4. In order to reach the *document catalogue* dictionary object, as specified before, you can check the /Root value in the trailer of the file. This catalogue contains references to

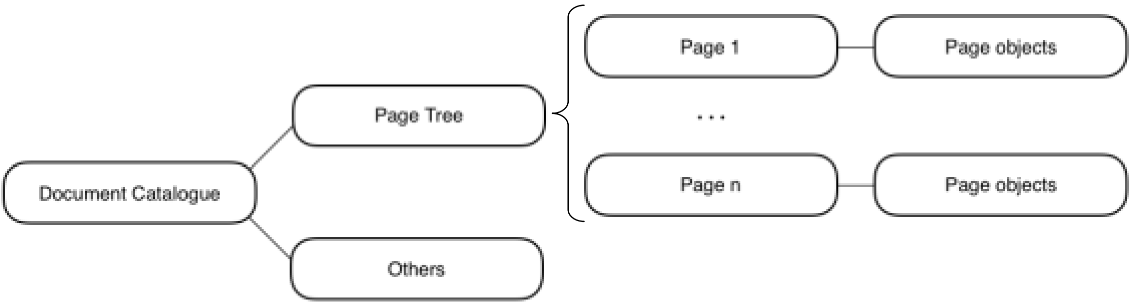


Figure 4. Simplified document structure schema

other objects which provide information about the document contents, as well as information about how these contents should be displayed.

This catalogue references several objects, but the most relevant to us is the *page tree*; it is a structure in which all page objects are organised and can be easily accessed. The page tree can contain two types of objects, page tree nodes and page objects:

- *Page tree nodes*, also called intermediate nodes. They do not directly correspond with a specific structure in the document (sections or chapters), but organise pages instead. The most basic construction of this type would be one only page tree node from which only one page would hang; moreover, if more than one page exists they can hang either from the same page tree node or from different ones. Finally, one page tree node can hang from another page tree node. Some attributes should appear in this dictionary, which are:
    - /Type: with the value ‘Pages’.
    - /Parent: except on the root nodes.
    - /Kids: an array specifying the indirect references to the page objects immediately hanging from it.
    - /Count: the number of descendant page objects this node has.
- *Page objects*, also called leaf nodes, correspond with the different pages in the document. They specify different characteristics and information about the page they refer to, in the form of a dictionary. Some elements in the dictionary are necessary, as the /Type, which must be ‘Page’ and /Parent, but others can be omitted. If so, some attributes can be defined as inheritable, in which case the page object will inherit the value specified in the parent page node.

Another relevant object referenced by the document catalogue is the *name dictionary*, which basically states the correspondence between objects and names, as sometimes it is more convenient to refer to them with names instead of indirect references.

Compression in PDF files

When referring to the content shown for the user in the PDF file, we mainly talk about stream data, as they are normally long sequences of bits. However, PDF has mechanisms so these big chunks of data do not occupy that much space, and it is done by means of compression. Consequently, at the time the document is conformed, data can be encoded in a stream for mainly two reasons: converting it to ASCII so it is easier to transfer, compressing it to save space or even both. In stream objects, /Filter attribute specifies the technique(s) used for decoding the streams when presenting the information.

Regarding ASCII decoders, PDF uses one filter for data encoded in hexadecimal form of ASCII (ASCIIHexDecode), and another for ASCII base-85 (ASCII85Decode). Both techniques retrieve the original binary data, which might correspond to an image or to already compressed data, and are lossless, so no data is lost in the process.

Regarding compression filters, PDF uses several of them, depending on the data being compressed and the specific requirements of each case:

- *Both text and images:* LWZ[7] and Flate methods are both adaptive and variable-length, and the difference resides in Flate (a zip compression method) using after LWZ an adaptive Huffman coding. They both can compress binary or ASCII data, and they produce a binary output. Flate is more used, as it reduces more the size of the data, although it has a slower performance. The basic principle in which they both operate is approaching the representation of the data by a differencing process; in an image, for example, it is shorter to describe the differences between one sample and the previous one than describing all samples. They also can use predictor functions to speed up these processes. Flate filter is the one used generally against text streams in PDF files, and even sometimes as a second filter for images.
- *Black and white images:* RunLength, CCITT[8] fax and JBIG2[9] decoders perform lossless compressions mainly oriented towards monochrome image compression (1 bit per pixel). RunLength decoder basically looks for sequences of equal bytes and translates them into number of repetitions, so the efficiency depends highly on the homogeneity of the bytes. CCITT and JBIG2 both work for bitmap image data, although JBIG2 has almost overtaken CCITT for presenting better results.
- *Colour images:* DCT[10] and JPX[11] decoders deal with grey scale and colour images. While the former decodes images being encoded with JPEG[12] techniques, the latter takes the ones being encoded with JPEG2000 specification, the evolution of JPEG, which defines JPX format. These encoding techniques are *lossy*, so the image obtained after decompression will not be the same as the original one.

When referring to images within the PDF file, we refer to images inserted as */XObject*, which behaves like a regular data object, not to inline images, which are included within a stream, as those will be treated just like the whole stream.

Embedded malware in PDF files

With the release of PDF version 1.3, JS[13] actions could be performed within PDF files [See Appendix I], which opened a door for malware authors to take advantage of this file format to perform criminal actions. Currently, PDF supports JS code inclusion and some JS functionalities in the form of APIs [17], which can be used to embed malicious code within a PDF document. Moreover, Flash objects allow to include shell code and perform this way malicious actions.

These malicious PDFs are normally distributed my means of email mass-spamming or including them in websites, with the objective of entering to end-point computers and infect

[7] Lempel-Ziv-Welch

[8] Comité Consultatif International Téléphonique et Télégraphique (International Coordinating Committee for Telephony and Telegraphy). Currently known as International Telecommunications Union (ITU).

[9] Joint Bi-Level Image Experts Group, member group of ISO (International Organization for Standardization)

[10] Discrete cosine transform

[11] Joint Photographic Experts

[12] Joint Photographic Experts Group

[13] JavaScript

them; however, they need mechanisms to bypass the security measures existent to protect systems from malware. A very common way for them to evade these security measures is to hide the malicious code by means of compression: as explained in previous section, there are many different filters available in PDF files, and they can even be applied in a cascade form, so it is even harder to decompress them. This serves the attackers to hide the malware until the PDF reader interprets those encoded streams and executes the code. For this, several tools possess functionalities to detect embedded malware before it is executed, such as Peepdf, which includes various functionalities against malicious code, like detecting JS and shellcode inside the PDF file or PDF Stream Dumper, which in addition allows you to separately address a specific stream for decoding and interacting with it [18].

This is just scratching the surface of malware embedding in PDF files, but it is such a wide topic that a separate study should be done about it, so we will leave it out of scope for our purposes.

Tools for PDF file inspection

In order to get to know the field, tools for PDF investigation must be addressed, as they have been used to perform the investigations; they all are open source and available for free, which, as a defender of *copyleft* and open source software, is an essential aspect in the selection criteria. Each tool has contributed in different tasks, and together they form a very thorough set for PDF analysis.

- *iText RUPS*: This software developer toolkit allows not only to enhance PDF functionalities, but also to integrate them within applications and processes [19]. Although it is available as a proprietary software owned by iText Group NV, it is also distributed as an open source library in Java under AGPLv3[14]. iText offers lots of functionalities to manage, create and interact with PDF files, and even automatise processes like archiving and documenting.
  Regarding our purposes, the most interesting utility this tool offers is 'Reading and Updating PDF Syntax (RUPS), which allows, by means of a GUI[15], to inspect, explore and visualize PDF objects, their hierarchy and relations within a file. It allows you to expand the object hierarchy from the root and see the bonds among the different objects present in the cross-reference table in an intuitive and pleasant way. Moreover, it also supports stream decoding, images visualisation and even attributes modification. One relevant detail to take into account when using this tool is the addition of some elements by iText RUPS into a PDF file that is being explored with it:
  - A comment as a signature of the software is added
  - If the type of metadata is */Info* object, it adds iText as an extra */Producer* of the file
  - If the type of metadata is XMP, it creates an */Info* object with the modification date and the */Producer* value set to iText.

  So, if no modifications are made and you do not save your work afterwards, these elements will not remain.
- *Qpdf*: This Perl based software has both manipulation, creation and investigation capabilities for PDF files [20]; it can linearize, encrypt, decrypt or select specific pages from a PDF file among other actions.

[14] Affero General Public License version 3

[15] Graphical User Interface

The very first basic thing it does is structurally organise the PDF source code, so it becomes more easily readable and understood. It has also decoding and encoding capabilities, which can be very useful when dealing with filters in PDF files. Another very interesting feature of this tool is QDF mode, which is a completely valid form of PDF file to which a regular file is converted in order to be more easily manipulated with a text editor; objects are organised in numerical order, all streams are decoded and normalised, and comments with details are added to objects so they can be identified easily.

- *Peepdf*: This Python open-source tool is mainly oriented towards detecting malicious files, for which provides JS and shellcode analysis features. For our interests, it allows you to explore and interact with PDF files content, from viewing metadata to decoding streams [21]. Used from the command line, it provides a quick way to visualize and investigate the objects of a file separately and to obtain details from the file, apart from creating and modifying PDFs

  - `python peepdf.py [Target_filename] –i`

  Isolating each object and working with them separately allows you to focus and clarify your work, and that is the main benefit Peepdf gives us.

- *Xpdf*: This tool is not just an open-source PDF file viewer, but has also several functionalities that simplify your life when dealing with manipulated PDF files, such as metadata retrieval or image and text extraction from the file [22]. It consists of a group of binary executables that once added to your path can be used against any PDF file to retrieve information or resources from it:

  - `pdfimages`
  - `pdftotext`
  - `pdfdetach`

  For our purposes, these executables are incredibly valuable, like *pdfimages* which is extremely useful to notice hidden ones, or *pdftotext,* which can show the existent text in a file in a very quick and easy way, which will facilitate further investigation.

- *PDF Stream Dumper*: This software in mainly oriented towards detection of malicious PDF documents, and deals mainly with shellcode, obfuscated JS and low level headers [18]. It detects them and lets you interact and experiment with them, as well as modify the file itself. It gives you all these possibilities through a very intuitive GUI, by which you can upload files and investigate their components. For our purposes, this tool provides us a quick way to locate and identify the streams within the file, decode them and even modify them.

2.4 Related work

When looking for information to address this thesis, a lot of previous work regarding scanned PDFs was concretely found. The initial idea resides in the digital images that had been tampered with, which due to the increasing functionality of image editing software, has become rather frequent and allows even non-professional people to easily manipulate images [23]. If we narrow the research towards digital images that have been previously scanned, a variety of methods have been proposed to identify forged images. The most successful ones bet on identifying differences among scanner sources, being able to determine if more than one scanner source trace can be found within the same image, which would mean that it has forged. One first method uses statistical features of imaging sensor pattern noise as a fingerprint for the scanner [23], which has been proved to be effective when used along with other existing methods. Another efficient method consists in taking a closer look to the refined noise, constructing reference patterns to identify the source scanner [24], whose results show that the scanner model is effectively identified regardless of training image types. Lastly, another method identifies the forged regions using dust or scratch positions and device imperfection pattern of the scanned images [25], and in addition, results are then compared to the two previously mentioned methods, concluding that all three methods are effective, but their accuracy depends greatly on the block size and that all the methods commit errors when trying to determine the forged regions.

Departing from this start, some wondered if it would be useful and effective to apply the same principles of image processing and/or source scanner identification to detect forgery in digital scanned documents instead of images. Following the line of identifying the source scanner, one method takes advantage of the fact that different scanners have different techniques to digitise documents, which results in a difference of quality in the edges of the letters in the text [26]; these variations of texture can be quantified, so they extract all 'e' letters from the text (which is the most common letter in English language), group them and divide them in pixel groups, from which they can extract the relevant features vectors and classify them by means of a SVM[16] classifier, which is the one deciding if any tamper has been committed. Similar to this, next proposed method addresses forgeries made to documents by printing and copying, and tries to identify the source printer to detect the manipulations [27]. It aims to distinguish documents produced by laser printers, inkjet printers, and electrostatic copiers, which are commonly used for document creation. Just as the previous discussed method, characters are extracted and interesting features are identified and organised by means of a SVM classifier, being this way able to detect different sources within a same document. In this case, the features addressed are noise energy, character contour roughness and average gradient on the character edge region.

But, what if the manipulations are done also digitally, and not by means of copying and scanning? A recent research focuses on using image processing tools to examine computer-manipulated documents [28]; they firstly manipulate several digital documents by means of Paint and Adobe Photoshop software tools, both deleting and adding text, and afterwards they investigate and examine the manipulated documents using Picasa and Adobe Photoshop software. They basically look for disturbances, such as irregular spacing, discrepancies in the font or the size of the text and uniformities in the background. The results of this study were described as encouraging and suggest ways to help examiners when trying to detect manipulations of digital documents. Nevertheless, this method could be bypassed with the copy-move forgery method, which consists of copying a region of the same document (a letter, for instance) and pasting it in another place within the document, so the result would be format consistent. Another recently proposed method aims to counteract this technique, combining copy-move forgery detection methods (CMFD) with optical character recognition methods (OCR) [29]. Firstly, a series of documents were tampered with different parameters, they were tested, also varying some testing parameters such as the block sizes, CMFD methods or metrics within the methods. The process of the analysis is divided in two branches which feed from each other; OCR process segments the document at one point, separating the background from the actual text. The background is examined separately, and the text is analysed to detect characters and classify them. This result becomes then an input for the CMFD process, which will extract the necessary features and analyse them. Finally, combining the final results from background and text examination, the decision about the authenticity and integrity of the document is made.

[16] Support Vector Machine

Another study on this field made by the Islamic University in Gaza proposes a new forgery detection method based on several features of the pixels in scanned documents [30]. The reason of proposing a new alternative in this paper is the lower efficiency that previous methods show when dealing with Arabic language, as it works in a completely different way. To solve this problem, they focus on the difference in the edge gradient and intensity of the characters to detect manipulations.

Moreover, when focusing exclusively on the text characteristics to detect if any forgeries have been performed against the document, text-line examination comes into picture. A research following this line of thinking poses that, in order to detect whether a file has been tampered or not, text-line rotation and alignment suppose a very important source of information [31]. The proposal presented in this paper corresponds to an automated approach for verifying documents based on these features (text-line rotation and alignment), demonstrating moreover its usefulness regarding document security.

All these methods, although having been proved effective and have produced good and useful results, require a lot of preparation and time to be applied, as they use image-processing techniques or text-line examination to do so, and need to work on the actual document they want to analyse. My idea is taking a different approach, and instead of applying image-processing / text-line techniques, addressing the document source code and metadata. We are going to examine what it is there and to find traces of modification and manipulation in PDF documents, as it is thought to be inalterable regardless of the software and hardware by which it is created and viewed.

3 Practical work context

3.1 Problem statement

Once the background has been covered, a need for stating the problem to address in this thesis arises; the main aim of this work is to ease the process of detecting manipulations in PDF files, as actual methods are arduous, like image processing or text-line examination which are the most used ones. As an alternative, the technique employed in this study is investigating directly the source code; in section *2.3 PDF anatomy* a deep study has shown us the structure and formation of this file type, as well as revealed some parts where traces of manipulation can be found. The idea of this thesis is that addressing directly the source code of the files, not only time and resources can be saved, but more proof of manipulations can be found, which is always necessary. Finally, the reason of selecting this format for the study is due to the general belief of PDF being the most secure one [1], as mentioned before.

3.2 Research questions

Now the problem statement has been defined, we need to divide it in approachable issues to address them separately.

Firstly, and after having performed a profound theoretical study of PDF internals, a need for practical interaction is needed to find secrets, vulnerabilities and alternative places where traces of manipulation may lay apart from the ones defined in the PDF specifications. So, the first question to answer is '*What vulnerabilities PDF file presents when directly attacking to the source code and where do they reside?*'.

Once we have mastered PDF internals we need to tackle the real cases of manipulation which, as mentioned above, do not correspond to manual manipulations in the source code, but to modifications made with editing software, careless of the traces left in the source code. So, the next questions to answer are *'Where to look when trying to find manipulation traces? Which are the PDF key objects for traces to reside?*'

Finally, and combining the knowledge about PDF structure and composition, as well as the vulnerabilities, we need to figure out a method to effective and efficiently detect manipulations made to PDF files. So, the final question to answer is '*How traces of manipulations made with editing software can be effectively detected?*'

3.3 Scope and limitations

As this is a limited study for a thesis project, the scope needs to be narrowed to our capabilities and objectives. This work is aimed to be a starting point in PDF manipulation detection by means of source code investigation, so the most typical and obvious manipulation cases have been addressed, which are the ones the examiners usually encounter when dealing with court cases. Just single-page PDF files, manipulated with regular editing software (Adobe Photoshop in most of the cases) and not manually manipulated (addressing the source code) have been tackled.

Regarding the manipulation part, and in terms of defining some kind of threat model, only the most typical profile has been addressed. As the files we are covering are presented in trials as digital evidence, the manipulator does not need to have advanced technical skills, but just the basic ones to know how to use editing software in a non-professional way (text, image and metadata deletions or additions). Therefore, the cost of these manipulations is presumed to be low in money (just the price of the used software) and medium in time investment (depending on the skilfulness of the specific manipulator).

4 Manipulation experiments

In this section, and in order to answer the research question ‘*What vulnerabilities PDF files present when directly attacking to the source code and where do they reside?*’, a series of manual manipulation experiments will be conducted by attacking the source code of the PDF files. The aim is to make this forgeries and manipulations in such a way that they remain undetectable, so even if this objective is not achieved, on the path, the traces that are left will be discovered, and that will be helpful to detect such manipulations afterwards, answering the next research question *’Where to look when trying to find manipulation traces? Which are the PDF key objects for traces to reside?*’. In any case, these experiments will be extremely useful to get familiar with PDF files source code, as being able to understand its structure and behaviour supposes an advantageous knowledge when investigating for manipulations.

For the manipulation experiments, sample documents have been created; first, they were printed, then scanned again as separate PDF files. Afterwards, two of them were manually signed with fake signatures and scanned again. This way, 8 sample documents were obtained [See Appendix II] which have the following features:

- *Sample1.pdf* and *Sample2.pdf*: digitally created files, with text recognition.
- *Watermark_txt.pdf* and *Watermark_only.pdf*: digitally created files with watermarks (first one also includes a little piece of text).
- *sc_Sample1.pdf* and *sc_Sample2.pdf*: scanned files, without text recognition (the whole page is a single image).
- *sig_sc_Sample.pdf* and *sig_sc_Sample2.pdf*: signed scanned files without text recognition (the whole page is a single image).

4.1 Cross-reference table manipulation

When carrying out any modification in PDF files, a readjustment of the cross-reference (xref) table must be done; as it locates the different objects by means of their byte-offset from the beginning of the file, if any modification is made that implies modification in the number of total bytes, object’s 10-byte offset must be recalculated and updated in the xref table. We will refer to this adjustment in following manipulation experiments.

4.2 Metadata manipulation

Metadata in PDF files is a crucial source of information when investigating them; key knowledge can be extracted from it, such as the author of the file (which would normally correspond to the machine user that originated it), the creation and modification dates (which can put the file in context and know if it has been modified at some point) or the tools involved in their conformance (maybe a PDF editor has processed the file at some point of its existence). For all this, integrity of the metadata fields is extremely important when examining a PDF file. In the process, we are going to use three tools, Notepad++, HxD and Exiftool, the first one in order to modify the text itself, as it is more clearly presented to us, the second one, a hexadecimal interpreter, to afterwards adjust the xref table and the third one to check if anomalies are detected within the process. To ensure the correct presentation of the file is maintained, we will use Adobe Acrobat Reader to view the file. As we want to examine both types of metadata, we are going to use Sample1.pdf and sc_Sample1.pdf, each one having one of the two types.

Changing contents of data

This experiment consists in manipulating the different fields in the Metadata object, both /Info and XMP in an unnoticeable way. Software that allows to modify PDF metadata leaves traces, such as an incremental update, so we are going to attack directly the source code of the PDF file. Starting with */Info* type of metadata, these are the steps followed:

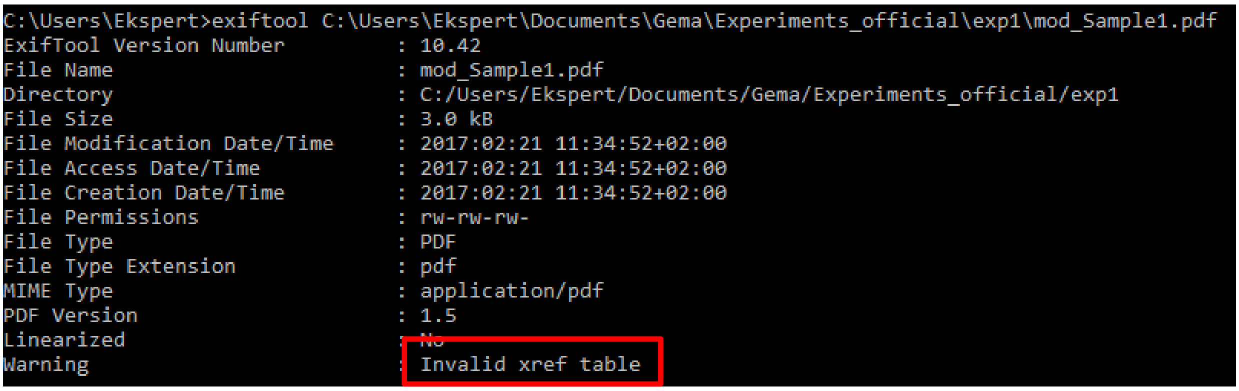
1. Locate the metadata, for that, we first move to the trailer at the end of the document, where /Info object number will be specified. Once inside /Info object, we just have to replace the content of each attribute at our pleasure. In Figure 5, an example of modification of all the existent fields is shown.

```
1 0 obj
<<
  /Title (Sample Document)
  /Creator (Microsoft® Office Word)
  /Author (Ekspert)
  /Producer (Microsoft® Office Word)
  /CreationDate (D:20170301001122)
  /ModDate (D:20170402001122)
  /Keywords (Sample)
>>
endobj
```

```
1 0 obj
<<
  /Title (Modified Document)
  /Creator (Microsoft: Print to PDF)
  /Author (AnonymousExpert)
  /Producer (Microsoft: Print to PDF)
  /CreationDate (D:20001111111111)
  /ModDate (D:20010101010101)
  /Keywords (Wordkey)
>>
endobj
```

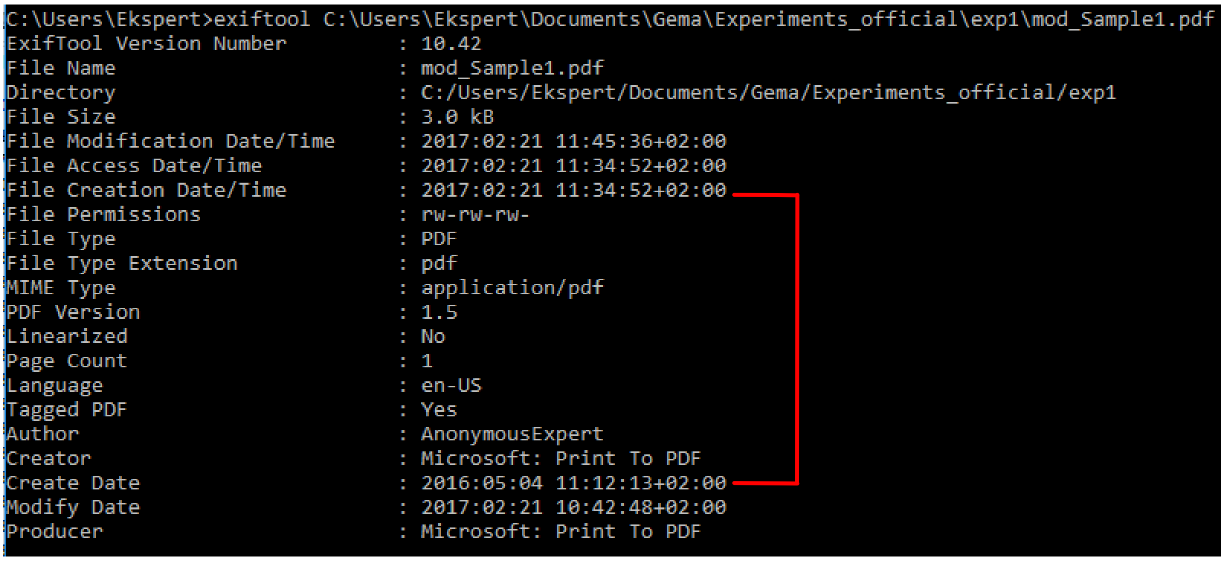
Figure 5. /Info content modification

2. At this point, the file is most certainly readable by the PDF viewer, but if we check the result file with Exiftool, we can see that the metadata cannot be interpreted and that the xref table is wrong, as shown in Figure 6
3. In order to fix that, HxD is used to correct the xref table. Every object placed after the one we have modified within the document must be verified, because those will be the ones with a wrong byte-offset, as bytes been added/extracted. The beginning of the xref table must be checked too, as the *startxref* value must be adjusted too.
4. Once we have all the new objects' offsets, we change them in the xref table. Then, Exiftool recognises the metadata and verifies that the document is correct.
5. If there is more than one xref table, or any xref streams, extra attributes must be modified, such as */Prev* (which indicates the offset to the previous xref table) or */XrefSteam* (which indicates the offset to the xref stream).



```
C:\Users\Ekspert>exiftool C:\Users\Ekspert\Documents\Gema\Experiments_official\exp1\mod_Sample1.pdf
ExifTool Version Number         : 10.42
File Name                       : mod_Sample1.pdf
Directory                       : C:/Users/Ekspert/Documents/Gema/Experiments_official/exp1
File Size                       : 3.0 kB
File Modification Date/Time     : 2017:02:21 11:34:52+02:00
File Access Date/Time           : 2017:02:21 11:34:52+02:00
File Creation Date/Time         : 2017:02:21 11:34:52+02:00
File Permissions                : rw-rw-rw-
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.5
Linearized                      : No
Warning                         : Invalid xref table
```

Figure 6. Exiftool output before modifications



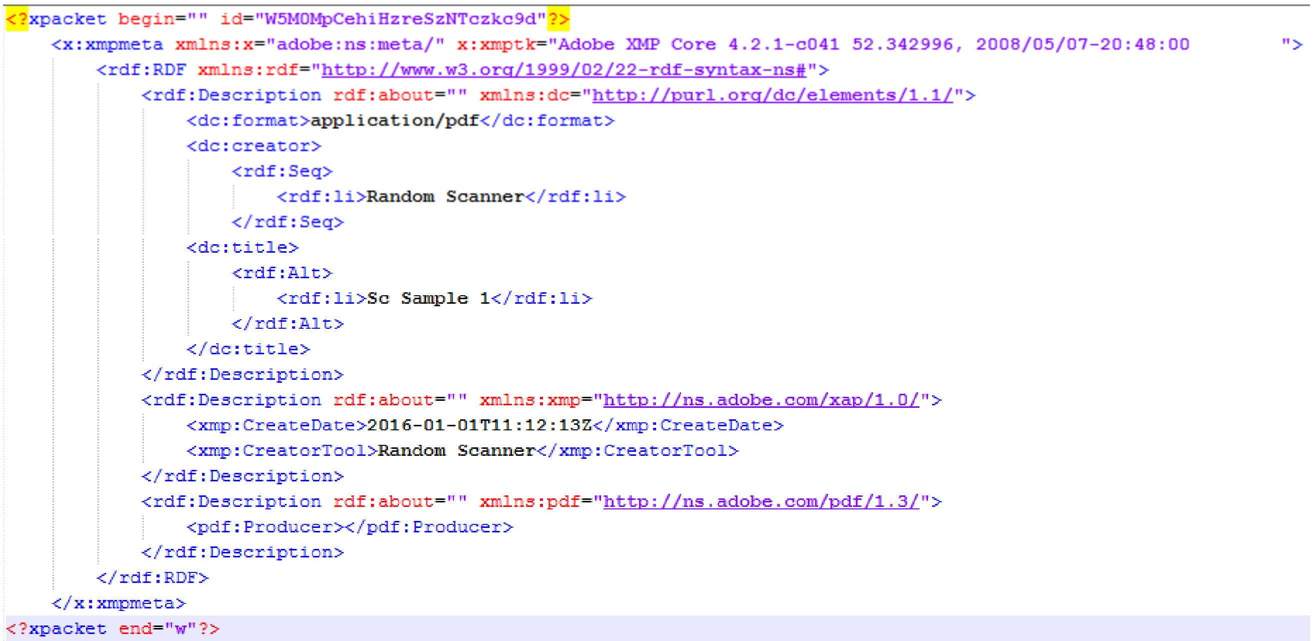
```
C:\Users\Ekspert>exiftool C:\Users\Ekspert\Documents\Gema\Experiments_official\exp1\mod_Sample1.pdf
ExifTool Version Number         : 10.42
File Name                       : mod_Sample1.pdf
Directory                       : C:/Users/Ekspert/Documents/Gema/Experiments_official/exp1
File Size                       : 3.0 kB
File Modification Date/Time     : 2017:02:21 11:45:36+02:00
File Access Date/Time           : 2017:02:21 11:34:52+02:00
File Creation Date/Time         : 2017:02:21 11:34:52+02:00
File Permissions                : rw-rw-rw-
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.5
Linearized                      : No
Page Count                      : 1
Language                        : en-US
Tagged PDF                      : Yes
Author                          : AnonymousExpert
Creator                         : Microsoft: Print To PDF
Create Date                     : 2016:05:04 11:12:13+02:00
Modify Date                     : 2017:02:21 10:42:48+02:00
Producer                        : Microsoft: Print To PDF
```

Figure 7. Exiftool output after modifications

Nevertheless, if we pay attention to the indicated data in Figure 7, ‘Create date’ and ‘File Creation Date/Time’ differ. This is because the latter refers to the file in the specific system in which is being investigated, not to his origin.

Regarding XMP metadata, these are the steps to follow:

1. Locate metadata: we first go to the */Root*, where */Metadata* indirect reference resides.



```
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
    <x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 4.2.1-c041 52.342996, 2008/05/07-20:48:00        ">
        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
            <rdf:Description rdf:about="" xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:format>application/pdf</dc:format>
                <dc:creator>
                    <rdf:Seq>
                        <rdf:li>Random Scanner</rdf:li>
                    </rdf:Seq>
                <dc:title>
                    <rdf:Alt>
                        <rdf:li>Sc Sample 1</rdf:li>
                    </rdf:Alt>
                </dc:title>
            </rdf:Description>
            <rdf:Description rdf:about="" xmlns:xmp="http://ns.adobe.com/xap/1.0/">
                <xmp:CreateDate>2016-01-01T11:12:13Z</xmp:CreateDate>
                <xmp:CreatorTool>Random Scanner</xmp:CreatorTool>
            </rdf:Description>
            <rdf:Description rdf:about="" xmlns:pdf="http://ns.adobe.com/pdf/1.3/">
                <pdf:Producer></pdf:Producer>
            </rdf:Description>
        </rdf:RDF>
    </x:xmpmeta>
<?xpacket end="w"?>
```

Figure 8. XMP after modifications

2. Once in */Metadata* object, information can be edited and even added at our pleasure; not only the text present in the fields can be modified, but fields can be also added, like done with the field *title* in Figure 8.
3. Then, as done before, the xref table is adjusted and then the result is checked with Exiftool, which gives us a positive result with the information we wanted.

Changing the type of metadata

In this case, the objective is changing the metadata type of a file without damaging the file itself. The reason why this is important is because when the documents are scanned with some scanners, they generate an XMP type of metadata, and maybe when a modification is done, the type of metadata changes to */Info* (or vice versa); so being able to systematically interchange them is a very important phase to address.

First, the different objects and references each type of Metadata leaves should be located:

- */Info*: `/Info 1 0 R` indirect reference in trailer, and */Info* object itself
- *XMP*: `/Metadata 1 0 R` indirect reference in */Root*, and /*Metadata* object itself.

Then, XMP type of metadata will be replaced with */Info* type. The followed steps are:

1. Delete */Metadata* object content to replace it with the */Info* object content that we would like to be present.
2. Insert new content in the object, with the */Info* metadata layout, including all the fields we want.

```
5 0 obj                            5 0 obj
<< /Type /Metadata                 <<
/Subtype /XML                      /Author(AnonymousExpert)
/Length 929 >>                     /Creator(Microsoft: Print to PDF)
stream                             /CreationDate(D:20170101111213+02’00’)
<?xpacket begin=””                 /ModDate(D:20170221104248+02’00’)
id=“W5M0MpCehiHzreSzNTczkc9d”?>    /Producer(Microsoft: Print to PDF)
[…] endstream                      >>
endobj                             endobj
```

Figure 9. Object content replacement

3. Delete the */Metadata* entry from */Root* object and insert in the trailer one */Info* indirect reference to the object that we have modified.
4. Mend the cross-reference table.

When checked with Exiftool, we can see the metadata is recognised and the file is correct.

Finally, the other way around would be replacing */Info* type of metadata with XMP. The followed steps are:

1. Locate the */Info* object and delete its content to use it as */Metadata* object.
2. Insert new content in the object, with the XMP layout.
3. Delete the */Info* entry from the trailer and insert a */Metadata* entry in */Root* object.
4. Mend the cross-reference table.

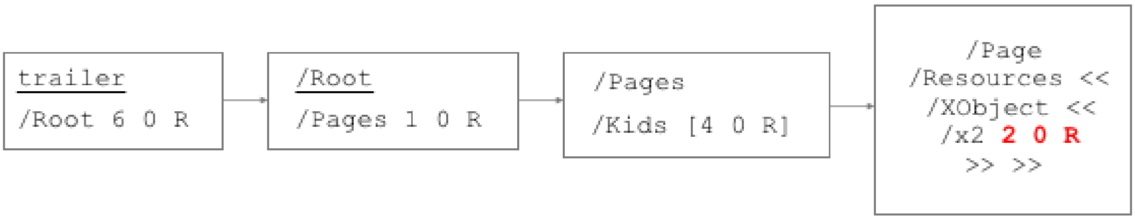
When checked with Exiftool, we can confirm the correctness of the file.

4.3 Image manipulation

In scanned files, scanners can either produce PDF files using OCR, so text is recognised, or not using it, so each page is represented as one whole image. In this section, non-OCR scanners will be address, so in order to modify the content, the image itself has to be modified. For that, several file manipulations involving the image objects of the file (*/XObject*) will be performed. As example, sc_Sample2.pdf will be used as reference file, and sig_sc_Sample1.pdf as source of pieces we want to insert in the previous file.

Extracting page images

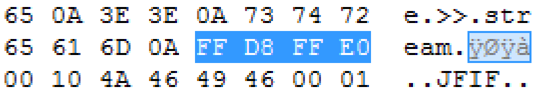
First step to manipulate the image itself is successfully extract it. For that, we need to navigate through the file (sig_sc_Sample1.pdf) until locating the page image object, as shown in Figure 10. We start in the trailer, which will lead us to the document catalogue (*/Root*), where there will be a reference to the */Pages* object. When */Pages* is located, a reference to its */Kids* should be included, that will take us to the actual */Page* object; over there, the indirect reference to the */XObject* should appear.



```
trailer
/Root 6 0 R

/Root
/Pages 1 0 R

/Pages
/Kids [4 0 R]

/Page
/Resources <<
/XObject <<
/x2 2 0 R
>> >>
```

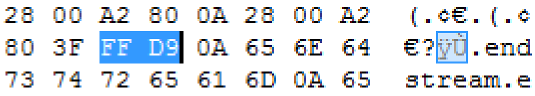
Figure 10. Steps to retrieve the image object

The image is represented as a stream of binary characters, as it is encoded in JPEG. HxD is used to find the JFIF[17] signature that locates the beginning of the image, in hexadecimal 'FF D8 FF E0'. Then, the trailer signature must be also found, which is 'FF D9'.



```
65 0A 3E 3E 0A 73 74 72  e.>>.str
65 61 6D 0A FF D8 FF E0  eam.ÿØÿà
00 10 4A 46 49 46 00 01  ..JFIF..
```

Figure 11. JFIF signature



```
28 00 A2 80 0A 28 00 A2  (.¢€.(.¢
80 3F FF D9 0A 65 6E 64  €?ÿÙ.end
73 74 72 65 61 6D 0A 65  stream.e
```

Figure 12. JFIF trailer signature

Finally, the content between those two signatures (including them) is selected and saved as a separate file with '.jpeg' extension (page_img.jpeg). This way, the image is obtained, and its validity can be checked by opening it with any viewer. We notice that the image corresponds to what we see in the PDF page, but rotated 90º clockwise direction. This is an important detail to have into account when inserting an image of our own.

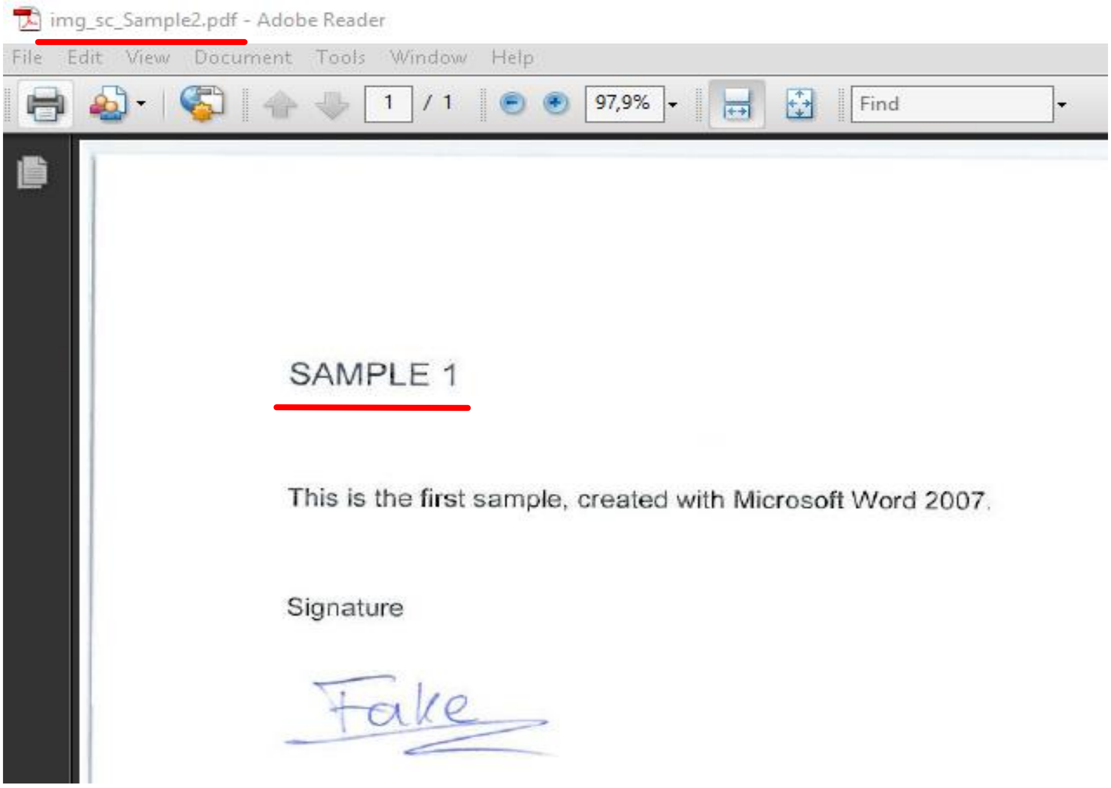
Inserting page images

Now that we know how to extract images from the PDF file manually, we would like to insert a page image of our own. For this experiment, the already extracted image from sig_sc_Sample1.pdf file (page_img.jpeg) will be used. It will be inserted into the page image object of the file sc_Sample2.pdf; as done in the previous section, the page image object is located first. Now, all we need to do is replacing the current image binary content with our own binary content. For this, HxD is used; first the existent image binary content is deleted and then binary content of page_img.jpeg is copied and pasted within the stream space.

Now, all we need to do is adjusting the cross-reference table to mend the created unbalances. After this, the file is checked with Exiftool and we confirm that everything is alright, and then we view the PDF with Acrobat Reader, and we can see that the replacement has been successful in Figure 13.

An important detail to take into account is that when we want to replace one image with other, both of them must have the same size, because the */XObject* builds a space for a specific image, and if this image does not comply with the characteristics described in the created frame, it would look deformed when displayed.

[17] JPEG File Interchange Format


img_sc_Sample2.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
97,9%
Find

SAMPLE 1

This is the first sample, created with Microsoft Word 2007.

Signature

Fake

Figure 13. Page image inserted in a different file

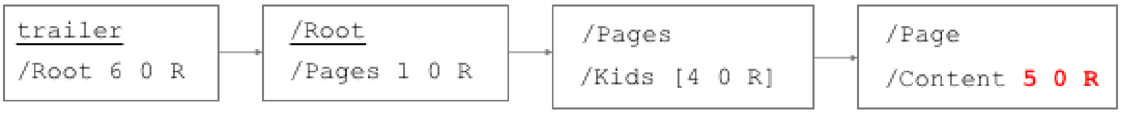
4.4 Text manipulation

When addressing a file that has recognised text on it, thanks to OCR for example, this means, not as a set of pixels but as an independent object with can be recognised as text by the viewer software, there are several complications while manipulating it.

- *Encoding*: in most cases, the stream within the object in which text content is specified is encoded, and it can only be retrieved applying a FlateDecode filter (the most usual filter to use in text objects).
- *Format*: when you manage to decode the content stream, most of the encountered data is not the text to be displayed itself, but commands on how to present it, regarding font, distribution in the file…

This means that modifications that can be done to the text of the PDF are minimal, and do not always work properly. One simple manipulation could be exchanging numbers (put a *1* when before there was a *2*), and that is easily performed:

1. Locate the text object, and for that, the following steps must be followed:



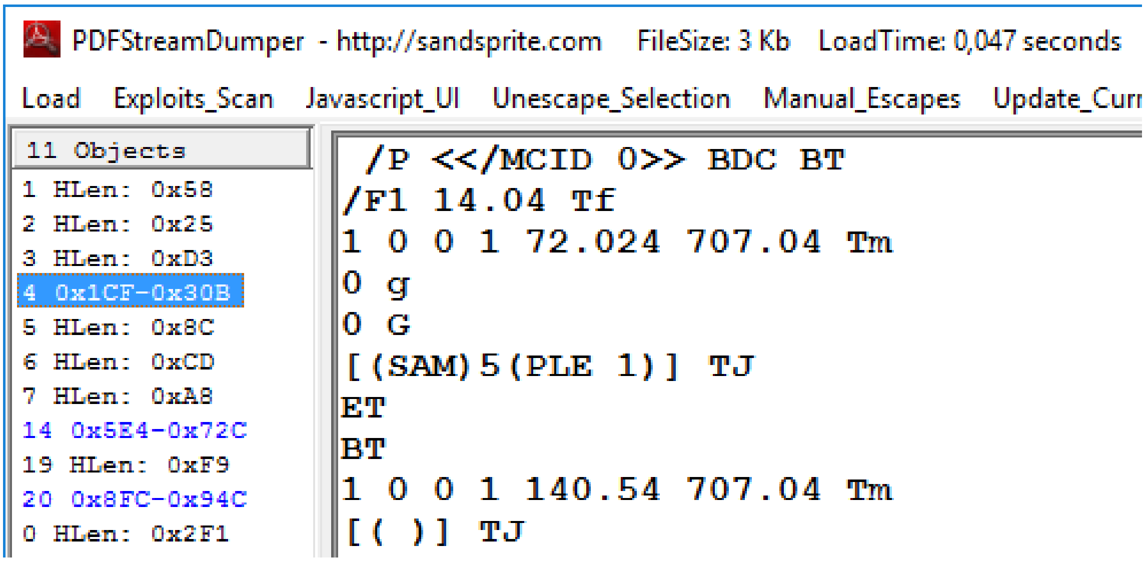
```
trailer          /Root            /Pages           /Page
/Root 6 0 R      /Pages 1 0 R     /Kids [4 0 R]    /Content 5 0 R
```

Figure 14. Steps to locate text object

2. Then, within the object we will learn which filter we should apply in order to decode it and the length of the encoded stream.

```
5 0 obj
<<
     /Length 955
     /Filter /FlateDecode
>>
```

3. After the dictionary, the stream is placed. To be able to modify the text, it needs to be decoded first, for which PDF Stream Dumper tool is used, which is mostly utilised for embedded malware detection.


PDFStreamDumper - http://sandsprite.com FileSize: 3 Kb LoadTime: 0,047 seconds
Load
Exploits_Scan
Javascript_UI
Unescape_Selection
Manual_Escapes
Update_Curr

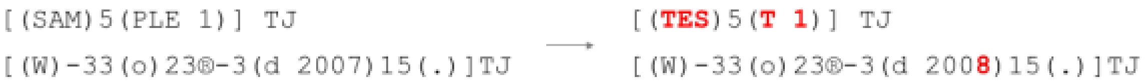
```
11 Objects
1 HLen: 0x58
2 HLen: 0x25
3 HLen: 0xD3
4 0x1CF-0x30B
5 HLen: 0x8C
6 HLen: 0xCD
7 HLen: 0xA8
14 0x5E4-0x72C
19 HLen: 0xF9
20 0x8FC-0x94C
0 HLen: 0x2F1
```

```
 /P <</MCID 0>> BDC BT
/F1 14.04 Tf
1 0 0 1 72.024 707.04 Tm
0 g
0 G
[(SAM)5(PLE 1)] TJ
ET
BT
1 0 0 1 140.54 707.04 Tm
[( )] TJ
```

Figure 15. Decoded text with PDF Stream Dumper

4. The file Sample1.pdf will be analysed as example, shown in Figure 15. PDF Stream Dumper allows to modify the plain text and to update the stream within the file, provided that these manipulations are minimal, as they could break the file. It performs all necessary adjustments in the file to maintain its correctness.

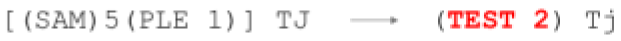
Another relatively simple manipulations are word exchange (not necessarily with the same amount of characters, but with minimal variations) or text deletion. In order to keep the appearance of the document a lot of details must be specified, so the usual type of attribute to indicate the text is ‘TJ’. The easiest way to modify this kind of text representation is changing the existent characters without modifying the structure, as shown in Figure 16.



```
[(SAM)5(PLE 1)] TJ                              [(TES)5(T 1)] TJ
                                          →
[(W)-33(o)23®-3(d 2007)15(.)]TJ                 [(W)-33(o)23®-3(d 2008)15(.)]TJ
```

Figure 16. Modifications made within ‘TJ’ attribute

Another option is exchanging the labels, and place a ‘Tj’ object where before there was a ‘TJ’ one, as done in Figure 17; this does not always work, and it does when the text we want to introduce is short, otherwise character overlapping may be present.



```
[(SAM)5(PLE 1)] TJ  →  (TEST 2) Tj
```

Figure 17. Modifications made by replacing 'TJ' with 'Tj' attribute

Watermarking

The resource of file watermarking is normally oriented towards its copyright protection, i.e. to avoid manipulation and/or illegal appropriation. Regarding PDF files and from a technical point of view, a watermark can be translated into an extra text object added to the file; consequently, to our concerns, it will be manipulated as text. The process is similar to text modification; if it is added after PDF file creation, an incremental update is performed, so it is rather easy to modify/delete it following these steps:

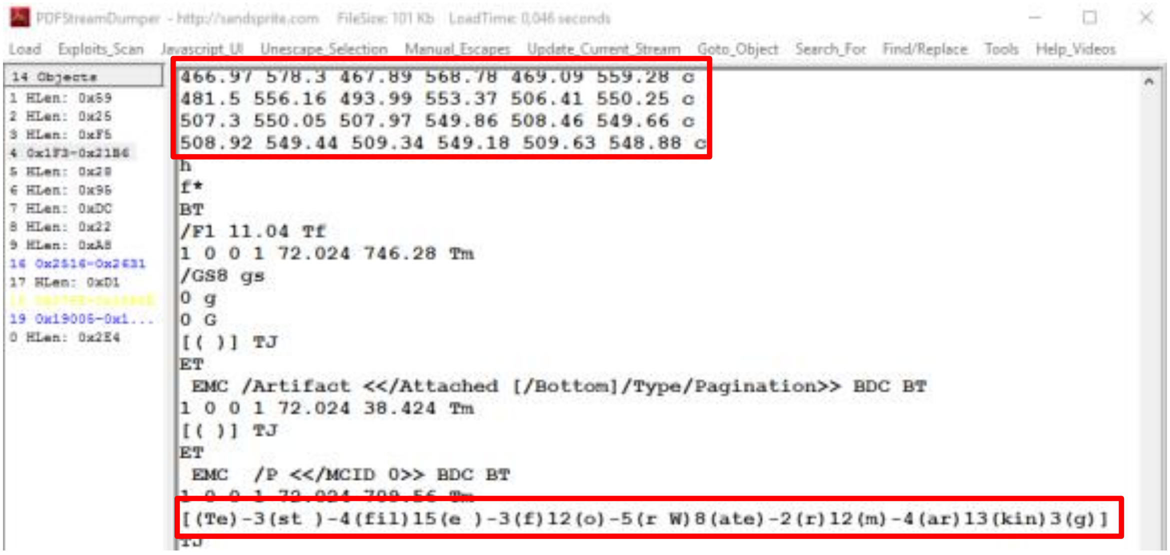
1. Locate the object, for what we move to the incremental update (normally at the bottom of the source code) and follow the same steps than in Figure 14.
2. If the content of the object is stored on plain text, it can be modified directly there, keeping in mind that extreme modifications may break down the text object. If encoded, same steps than in the previous section can be followed, using PDF Stream Dumper tool to decode and modify it.
3. Finally, the cross-reference table is adjusted as usual.

If what we want is to delete the watermark, the following traces should be removed:

1. Delete the reference in the */Content* attribute of the */Page* object.
2. Delete the text object.
3. Delete the added cross-reference table and trailer.
4. If necessary, adjust the cross-reference table to the new values.

However, when it comes to watermarks added at the same time as the creation of the document, there is no such incremental update; the most common case is digitally created documents, and the result is new text object (or even adding it to an existent one). So, in this case, the watermark is represented as a series of coordinates (highlighted upper part of Figure 18), that help drawing the image of the watermark within the text object (lower part of the image), being rather difficult to modify. Nevertheless, the deletion process of this watermark is rather easy:

- Within the text object, the part related to the watermark is deleted and the cross-reference table is adjusted (which in this case PDF Stream Dumper does automatically). Now if we open the PDF file with a viewer, the watermark is gone.



```
14 Objects
1 HLen: 0x69
2 HLen: 0x25
3 HLen: 0xF5
4 0x1F3-0x21B6
5 HLen: 0x28
6 HLen: 0x96
7 HLen: 0xDC
8 HLen: 0x22
9 HLen: 0xA8
16 0x2516-0x2631
17 HLen: 0xD1
19 0x19006-0x1...
0 HLen: 0x2E4
```

```
466.97 578.3 467.89 568.78 469.09 559.28 c
481.5 556.16 493.99 553.37 506.41 550.25 c
507.3 550.05 507.97 549.86 508.46 549.66 c
508.92 549.44 509.34 549.18 509.63 548.88 c
h
f*
BT
/F1 11.04 Tf
1 0 0 1 72.024 746.28 Tm
/GS8 gs
0 g
0 G
[( )] TJ
ET
 EMC /Artifact <</Attached [/Bottom]/Type/Pagination>> BDC BT
1 0 0 1 72.024 38.424 Tm
[( )] TJ
ET
 EMC  /P <</MCID 0>> BDC BT
1 0 0 1 72.024 709.56 Tm
[(Te)-3(st )-4(fil)15(e )-3(f)12(o)-5(r W)8(ate)-2(r)12(m)-4(ar)13(kin)3(g)]
TJ
```

Figure 18. Text object with watermark and text

5 Manipulation detection method

So far, we have dived through the insides of PDF files, firstly discovering their anatomy and structure and then twisting and taking advantage of it by manually manipulating their content and metadata, with the objective of getting a deeper knowledge and pushing the limits of the file. At this point, the research question '*How traces of manipulations made with editing software can be effectively detected?'* should be answered. Therefore, traditionally manipulated files can be investigated to detect traces left by external tools that do the code adjusting themselves, normally careless of leaving traces behind, instead of attacking directly to the source code. For this, a reference layout of the objects and structure of a non-manipulated PDF file will be constructed, to be able to compare the target file and learn the differences between them following a series of steps; this way, these differences can be studied and the integrity of the target file can be determined.

There are several assumptions to consider when addressing this method, regarding the context and the information we have before starting the investigation. This method is being developed with the purpose of helping the investigators in their daily tasks, targeted to the admissibility of the documents in court; this implies the possible knowledge of several information regarding the file beforehand, like for example:

- *Category of the document*: if the document was digitally created or digitised, and if OCR was performed upon it. These characteristics can be either known as extra information about the case or learned when first accessing the document.
- *Presumed or suspected basic metadata*: in court cases, the hesitation about a document being manipulated or genuine arises from the affirmation of one part that it is forged and the other acknowledging the contrary. In this case, the origin of the document must be contrasted, comparing the presumed date, time of creation/modification with the reflected on the metadata. Same happens with author, title, creation/modification software and all metadata fields existent in the document.

Therefore, if such background information about the files is not available (like in this study), hypothesis should be presented. Nevertheless, when put into practice this information will be available, so it is not a concerning matter.

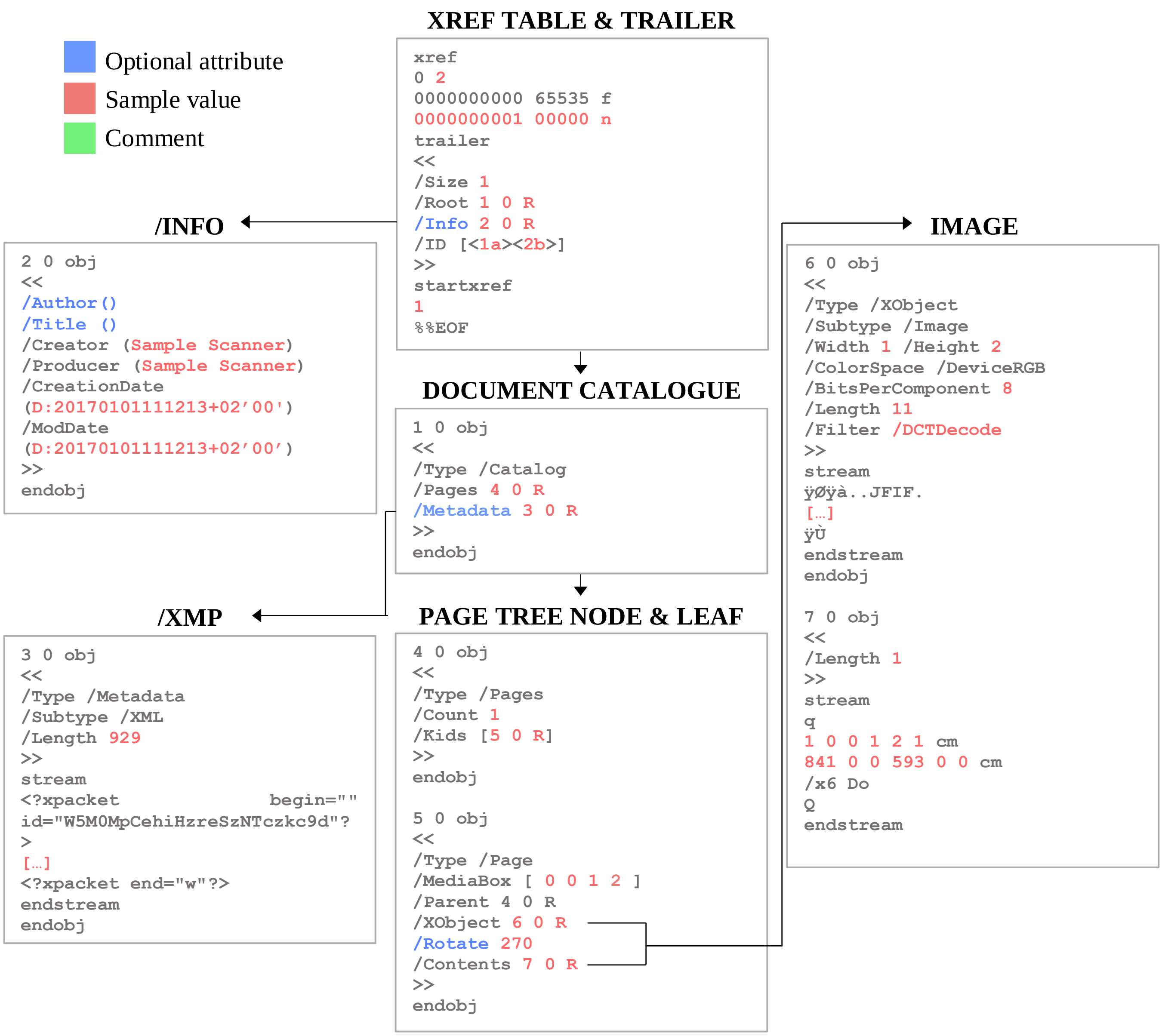
5.1 Investigation layout

In order to detect if a file has been manipulated, it would be extremely useful to have a reference layout of the objects and structure of a non-manipulated PDF file to be able to compare the target file of the analysis and learn the differences between them; this way, these differences can be studied and the integrity of the target file can be determined.

Scanned PDF files without OCR

The first schema corresponds to files that have been scanned without performing OCR. In Figure 19, we can observe the set of objects that appear on such files, as a reference on what you should be expecting when investigating the file under suspicion. The objects appear in the order they should be investigated, starting with the cross-reference table and trailer until the specific stream and image objects.

- *Cross-reference table & trailer*: first thing needed to be noticed is the absence of incremental updates in any case, as it would most certainly involve a modification. As these files are directly created in PDF, there should be only one cross-reference table. In the trailer, */Size*, */Root* and */ID* objects are a must, and then, depending on the type of metadata, there will be or not a reference to the */Info* object.



🟦 Optional attribute
🟥 Sample value
🟩 Comment

XREF TABLE & TRAILER

```
xref
0 2
0000000000 65535 f
0000000001 00000 n
trailer
<<
/Size 1
/Root 1 0 R
/Info 2 0 R
/ID [<1a><2b>]
>>
startxref
1
%%EOF
```

/INFO

```
2 0 obj
<<
/Author()
/Title ()
/Creator (Sample Scanner)
/Producer (Sample Scanner)
/CreationDate
(D:20170101111213+02'00')
/ModDate
(D:20170101111213+02'00')
>>
endobj
```

DOCUMENT CATALOGUE

```
1 0 obj
<<
/Type /Catalog
/Pages 4 0 R
/Metadata 3 0 R
>>
endobj
```

/XMP

```
3 0 obj
<<
/Type /Metadata
/Subtype /XML
/Length 929
>>
stream
<?xpacket              begin=""
id="W5M0MpCehiHzreSzNTczkc9d"?
>
[…]
<?xpacket end="w"?>
endstream
endobj
```

PAGE TREE NODE & LEAF

```
4 0 obj
<<
/Type /Pages
/Count 1
/Kids [5 0 R]
>>
endobj

5 0 obj
<<
/Type /Page
/MediaBox [ 0 0 1 2 ]
/Parent 4 0 R
/XObject 6 0 R
/Rotate 270
/Contents 7 0 R
>>
endobj
```

IMAGE

```
6 0 obj
<<
/Type /XObject
/Subtype /Image
/Width 1 /Height 2
/ColorSpace /DeviceRGB
/BitsPerComponent 8
/Length 11
/Filter /DCTDecode
>>
stream
ÿØÿà..JFIF.
[…]
ÿÙ
endstream
endobj

7 0 obj
<<
/Length 1
>>
stream
q
1 0 0 1 2 1 cm
841 0 0 593 0 0 cm
/x6 Do
Q
endstream
```

Figure 19. Investigation layout for scanned files without OCR

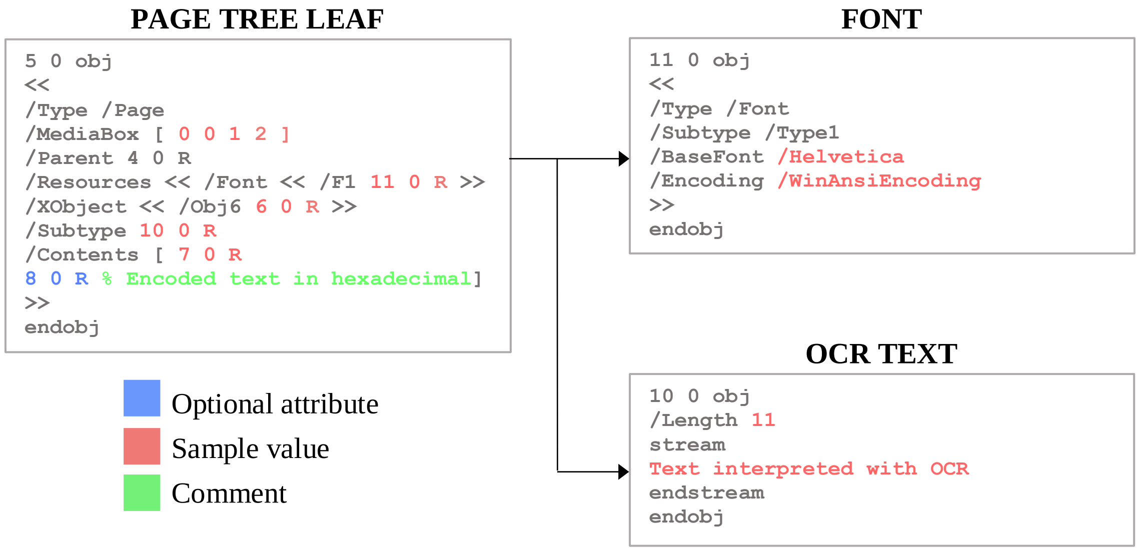
- *Document catalogue*: here in the */Root* object, you will be able to find a reference to the Page Node and, depending on the type of metadata, a reference to XMP.
- *Metadata*: depending on the brand and mode of operation of the scanner, it will produce either a */Info* or an XMP type of metadata. In both cases, the */Creator* or even the */Producer* will be marked somehow with some label related to the specific machine that performed the scanning. Also, the creation and modification dates **must** be identical, otherwise it would mean that the file has been altered somehow.
- *Page tree node & leaf*: the first one (*/Type /Pages*) gathers the references to all the page leafs, this is, actual pages within the file. Once in the page tree leaf (*/Type /Page*), we will find a reference to the node (parent), and the attribute */Mediabox*, which defines the boundaries within which the page will be displayed. But the most important element is the reference to the image, which can be represented either as reference to an */XObject*, or within the attribute */Resources*. Either way, the reference will lead us to the actual image object. Another existent reference is */Contents*, which in this case, in the absence of text.

▪ *Image*: because these files do not recognise text, they interpret the whole file as a set of images representing each of its pages (one object per image per page).
In this object, we will find the type, subtype and several other attributes specifying other details of the image, including the filter needed for decoding the stream that represents the image itself. Besides, the object referenced by the attribute */Content* in the page leaf point towards a descriptive object, which specifies details about position parameters of the image. These attributes tend to be similar among documents, as all the images are complete pages of a file, so the position is almost equal (same with the attributes width and height in the */XObject*).

Scanning PDF files with OCR

In this case, the scanner performs OCR while scanning, so the output file includes the text that has been recognised in an object within the file, as shown in Figure 20.

The only difference with the previous type is the existence of an extra object, indirectly referenced in the page leaf object as */Subtype*; this object contains (encoded or decoded) the interpretation of the recognised text. Moreover, in */Resources* attribute we can see a reference to a */Font* object, referring to the font recognised in the text.



PAGE TREE LEAF

```
5 0 obj
<<
/Type /Page
/MediaBox [ 0 0 1 2 ]
/Parent 4 0 R
/Resources << /Font << /F1 11 0 R >>
/XObject << /Obj6 6 0 R >>
/Subtype 10 0 R
/Contents [ 7 0 R
8 0 R % Encoded text in hexadecimal]
>>
endobj
```

FONT

```
11 0 obj
<<
/Type /Font
/Subtype /Type1
/BaseFont /Helvetica
/Encoding /WinAnsiEncoding
>>
endobj
```

OCR TEXT

```
10 0 obj
/Length 11
stream
Text interpreted with OCR
endstream
endobj
```

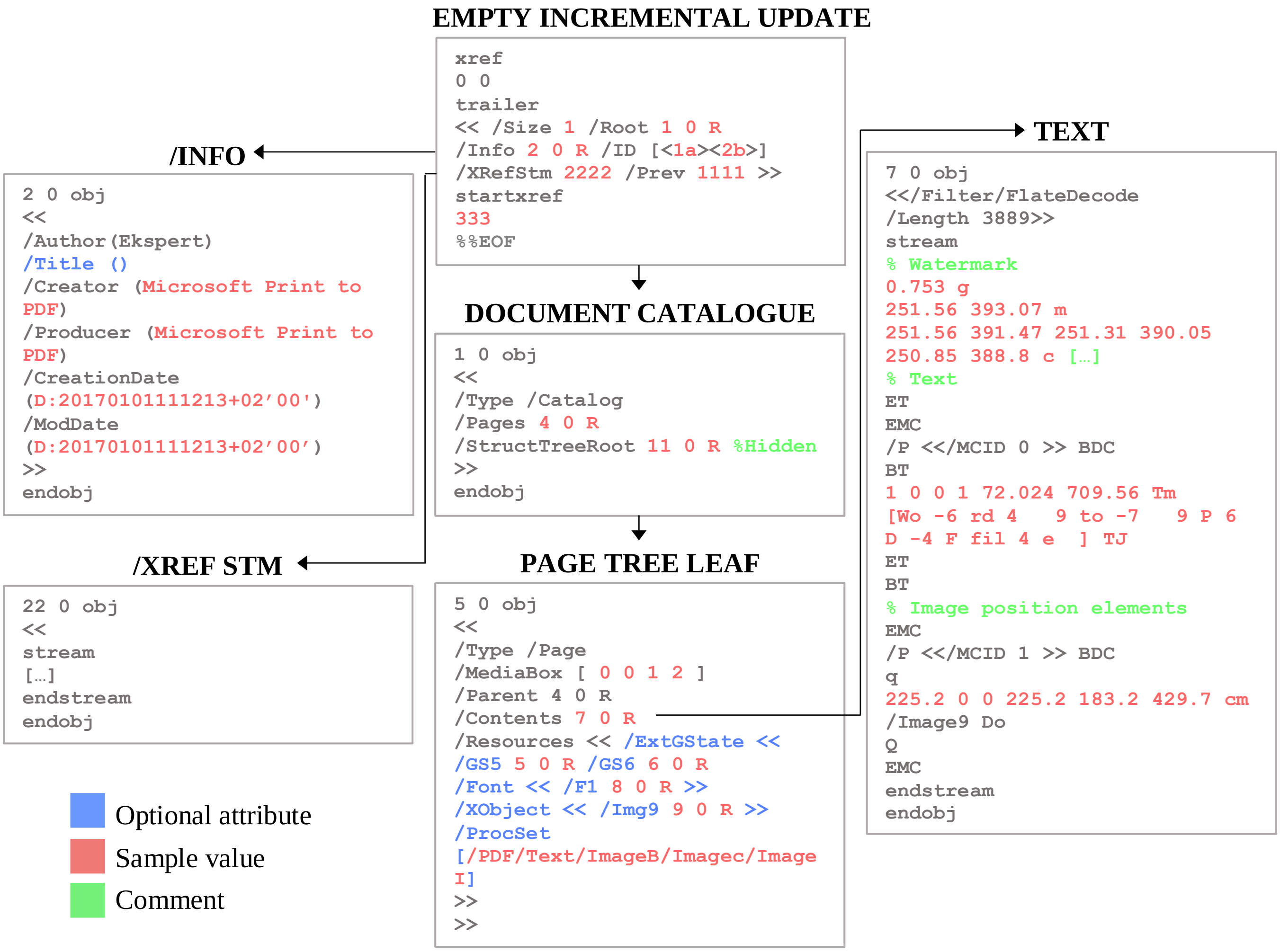
🟦 Optional attribute
🟥 Sample value
🟩 Comment

Figure 20. Additional objects in OCR PDF files

Digital files converted to PDF

In this section, we refer to those files created with a text editor and then converted to PDF. This kind of PDF files are harder to normalise, as many applications perform this conversion. Nevertheless, most of them create the same structure and normally add the same objects and attributes. The main differences regarding previous file types are:

- *Incremental update*: a priori, there should be no incremental update, being this the first version of this PDF file. However, we might find extra cross-reference table, but as you can see in Figure 21, it ought to be empty. This is due to the existence of a cross-reference object stream, referenced by */XRefStem* in the trailer. This attribute is encoded, and exists due to reasons of compatibility between versions of PDF file. Together with an object stream, they contain information about the hierarchy of the file, and depending on the supported version of the PDF reader, regular table or the cross-reference stream [13] will be used.



EMPTY INCREMENTAL UPDATE

```
xref
0 0
trailer
<< /Size 1 /Root 1 0 R
/Info 2 0 R /ID [<1a><2b>]
/XRefStm 2222 /Prev 1111 >>
startxref
333
%%EOF
```

/INFO

```
2 0 obj
<<
/Author(Ekspert)
/Title ()
/Creator (Microsoft Print to
PDF)
/Producer (Microsoft Print to
PDF)
/CreationDate
(D:20170101111213+02'00')
/ModDate
(D:20170101111213+02'00')
>>
endobj
```

DOCUMENT CATALOGUE

```
1 0 obj
<<
/Type /Catalog
/Pages 4 0 R
/StructTreeRoot 11 0 R %Hidden
>>
endobj
```

/XREF STM

```
22 0 obj
<<
stream
[…]
endstream
endobj
```

PAGE TREE LEAF

```
5 0 obj
<<
/Type /Page
/MediaBox [ 0 0 1 2 ]
/Parent 4 0 R
/Contents 7 0 R
/Resources << /ExtGState <<
/GS5 5 0 R /GS6 6 0 R
/Font << /F1 8 0 R >>
/XObject << /Img9 9 0 R >>
/ProcSet
[/PDF/Text/ImageB/Imagec/Image
I]
>>
>>
```

TEXT

```
7 0 obj
<</Filter/FlateDecode
/Length 3889>>
stream
% Watermark
0.753 g
251.56 393.07 m
251.56 391.47 251.31 390.05
250.85 388.8 c […]
% Text
ET
EMC
/P <</MCID 0 >> BDC
BT
1 0 0 1 72.024 709.56 Tm
[Wo -6 rd 4   9 to -7   9 P 6
D -4 F fil 4 e  ] TJ
ET
BT
% Image position elements
EMC
/P <</MCID 1 >> BDC
q
225.2 0 0 225.2 183.2 429.7 cm
/Image9 Do
Q
EMC
endstream
endobj
```

🟦 Optional attribute
🟥 Sample value
🟩 Comment

Figure 21. Additional objects in files created by 'Print to PDF'

This is called a hybrid-reference file, and files converted to PDF normally belong to this type, to make them interoperable among versions. For this same reason, in the document catalogue (*/Root*) there is a reference to */StructTreeRoot*, an object that cannot be found in the source code and that appears as 'free' in the regular cross-reference table; it exists in the encoded stream of the object stream.

- *Resources*: when we reach the page leaf, we can find many different combinations of attributes within the */Resources* dictionary. Considering a file that has at least one image, some text and a watermark, the most common ones are shown in Figure 21. Most of them refer to the appearance of the page, like */ExtGState*, which refers to how graphics are rendered to the screen [32], or the procedure sets (*/ProcSet*), which specifies the types of objects you will encounter within the file (text, colour/black&white images…). Although we will not address them, there can be more attributes, depending on the contents of the file (such as multimedia or forms).
- *Contents*: finally, unlike in the previous types, page objects include images, text, watermarking and even multimedia (not covered in this study), while the scanned ones had just a single image to represent each page (a 'picture' of the page). Within the text object referenced in */Contents* the text shown in the page and watermark texts can be found. One clear indication that a PDF file has been manipulated is finding text both in the object referred in */Contents*, and in an */XObject* (which would correspond to a text box), as it may mean it has been added afterwards.

5.2 Investigation method

Once we have set normalized layouts to which we can compare our files under investigation, we proceed to describe a series of steps, including possible tools to use for the tasks, and references to these layouts to draw conclusions about the integrity of the file.

As stated before, we assume that a context will be provided along with the digital evidence to investigate; therefore, before the actual investigation, some checks will be done beforehand, among them opening the file with a viewer to write down our firsts impressions and ideas.

Metadata investigation

As repeatedly pointed out in this document, metadata is the first element we should face when investigating a file, as most interesting information for an examiner resides there. As stated before, contrasting the presumed metadata with the actual one is the main objective in this part of the investigation.

- First thing to do will be looking at the metadata shown by the viewer; this may or may not be accurate, as more than one metadata objects can co-exist in the document, and the viewer most certainly will only interpret one of them. Here we will get a fair idea of what it can be learned without analysing the file in depth and we can compare it with future findings we may discover.
- The first point of interest is the existence of an *incremental update* and its content; if the file is a scanned document, there should not be one. If the file has been originated by converting it to PDF, then there might be one, with the characteristics explained in the section *Digital files converted to PDF* (empty cross-reference table and references */Prev* and */XRefStm* in the trailer), but its existence is not necessary. In order to check it, we can use any text editor (Notepad++ for example).
- For investigating the content of the metadata, we can use several tools to retrieve it, but Exiftool will provide you with the most detailed and accurate information, being able to contrast the presumed information with the actual one.
    - `exiftool [target_file]`

Content extraction

Once the metadata has been covered, the actual content of the file can be addressed. When opened with a viewer, we can draw some conclusions about the file, if we did not know them beforehand: whether it is a scanned file or a digitally created one, whether it has OCR performed or what objects apparently form the file (text, images…).

Afterwards, some content extractions are performed so we can learn what components exist in the file and discover hidden elements if any. In order to do so, Xpdf can be used, which allows to extract separately text, images and attached files if existent, and save them in different files using the executables provided by the software with the following commands:

- `pdfimages [target_filename] –j [output_base_filename]`

The option '-j' indicates that the format of the output files will be JPG, as the default one is PBM[18] for monochrome images and PPM[19] for non-monochrome ones. The value 'output_base_filename' corresponds to the root of the filename the extracted images will

[18] Portable Bit Map

[19] Portable Pixel Map

have; for example, if we give it the value ‘img’, the images will be named ‘img0001’, ‘img0002’, etc.

- `pdftotext [target_filename] [output_filename]`
- `pdfdetach [target_filename] –list`

The option –list indicates to only show the list of attached files in the screen instead of saving them in files; this is done to check if there are any before trying to obtain them.

Now that all the content has been extracted, it can be compared to what we can see with the viewers, so hidden files, for example, can be discovered. Another matter we can notice at this stage are manipulations in scanned files; if there are single images that do not correspond to a whole page image, then they must be added afterwards. We can see that some conclusions can be drawn already.

Layout comparison and conclusions drawing

Finally, and basing our following steps on the discoveries in the previous stages, the specific objects that exist in the target file will be investigated in order to verify its integrity. For that, a series of steps to be followed have been defined, shown in Figure 22, to combine all the information found in previous steps and draw conclusions. As stated before, either the kind of PDF file we are dealing with is known in advance (it is information from the court case) or it can be deduced from the elements and characteristics found from previous phases. Either way, we will try to confirm our hypothesis and determine if the file has been manipulated or not. Now the flowchart’s content can be clarified:

- Firstly, the target file should be compared with the layout, matching all objects present in the reference with the ones in the target. For this, iText RUPS tool can be used, which will allow us to see the objects in a structured and clear way, as well as to connect them hierarchically to facilitate the process of investigation. If the objects in the target file match the ones in the layout, we have successfully established our reference of study.
  If they do not, we should check if there the target file matches any other layout; if it does, we now have the reference set, and if it does not, we will have an unidentified construction and the elements will be investigated separately.
- Next, we take a look at the objects in the file. From previous steps, we have the images and text we have extracted with Xpdf, so firstly they should be mapped with the objects we have already identified. In case of having a reference layout we must, moreover, locate any existent object in the target file that does not appear in the layout. These objects can be just suitable additions (our layouts only cover the basic elements that should be present), but they might also be out of place, as for example an extra image in a scanned document apart from the one corresponding to the whole page. For this, we can use the options from Peepdf and Qpdf to show and decode a specific object to isolate it from the rest of the document and clarify the content.
  - `PPDF> object 1`
  - `qpdf [target_file] –show-object=1`
- In any case, after having identified the objects of interest (suspicious ones), they will be evaluated together with the metadata extracted from previous steps, in order to look for traces of manipulation, (references to editing software, timestamps…)

As we can see, the conclusions of this part of the investigation are either that the file is correct a priori, or that it is manipulated.

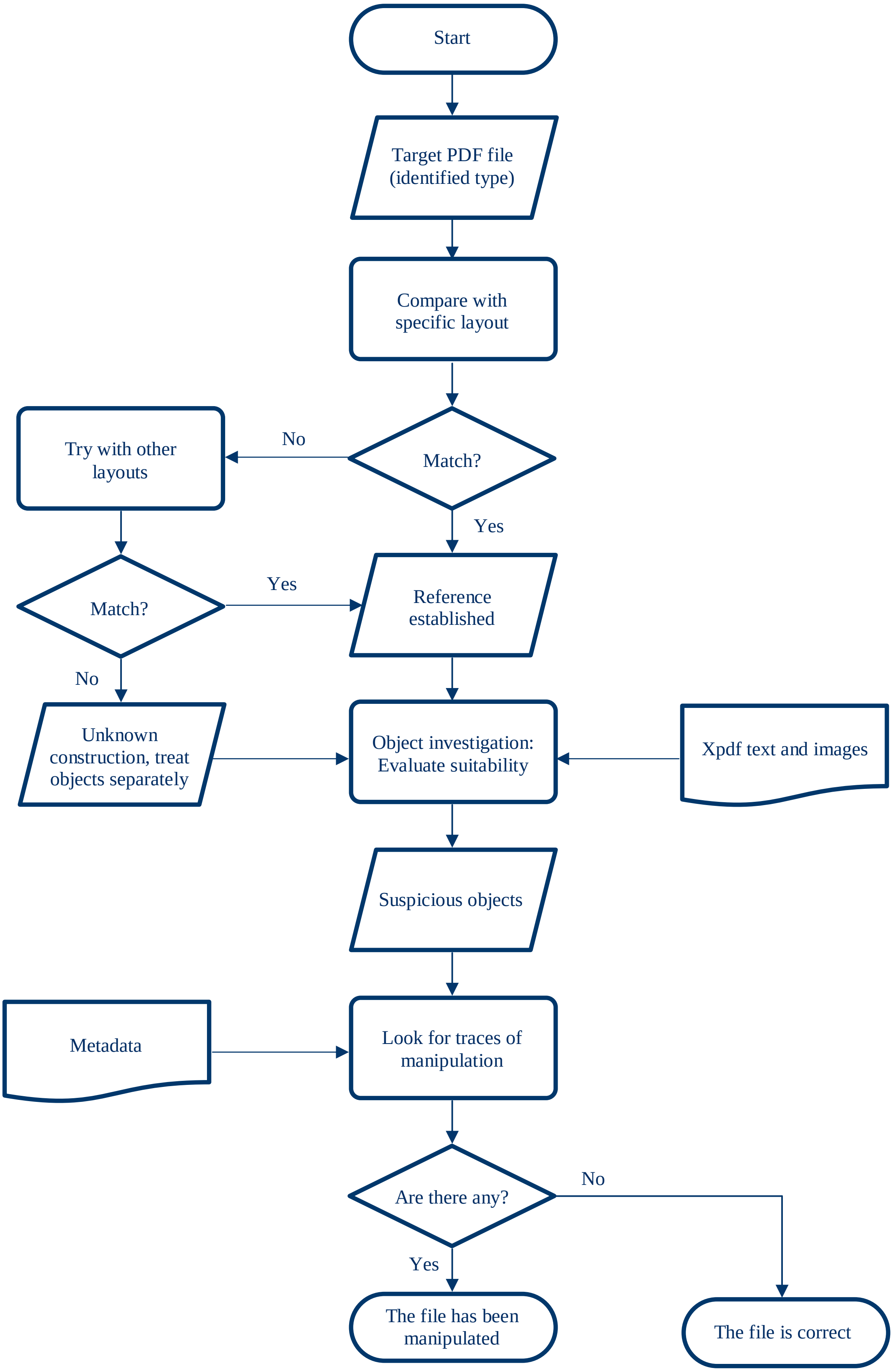

Start
Target PDF file (identified type)
Compare with specific layout
Match?
No
Try with other layouts
Yes
Match?
Yes
Reference established
No
Unknown construction, treat objects separately
Object investigation: Evaluate suitability
Xpdf text and images
Suspicious objects
Metadata
Look for traces of manipulation
Are there any?
No
Yes
The file has been manipulated
The file is correct

Figure 22. Method flowchart

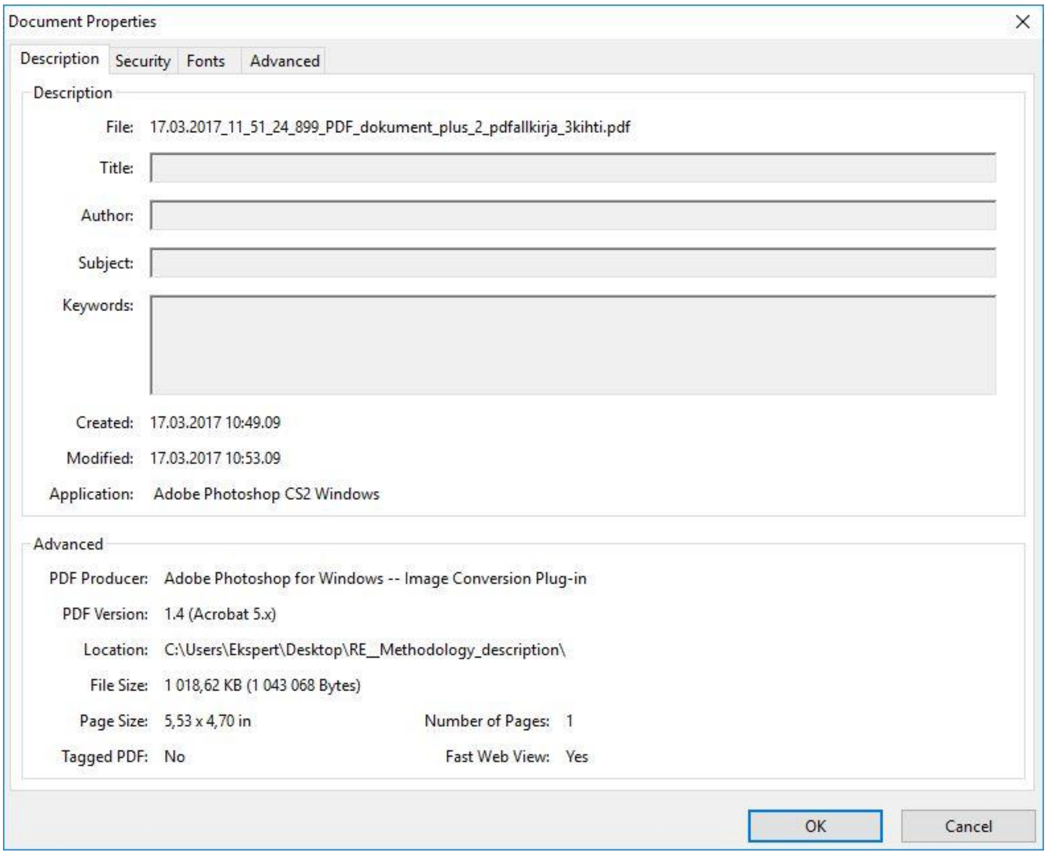
6 Validation tests

It is now time to prove the effectiveness of the method. After having described and detailed it, we are going to put it through some validation tests that will determine its effectiveness. To perform these tests, 9 PDF files provided by Eesti Kohtuekspertiisi Instituut (EKEI) from Inna Ivask, document expert, were used; these files are manipulated copying the most usual procedures used to manipulate digital evidence, whose details and actual files can be found in Appendix IV. When the documents were delivered, whether all of them were manipulated or not, nor which manipulations had been performed was unknown for the validating part. Therefore, first these documents were subjected to the method, and only afterwards the results were evaluated against the forgeries which had been performed, so conclusions on its effectiveness were drawn.

In the main document only the first, fourth and ninth test procedures would be presented, as they are the most illustrative, and then a summary table and the results evaluation will be posed. Moreover, to normalize the nomenclature, the files will be named as ‘Evidence N’, being ‘N’ their position in alphabetical sorting order, and a mapping with the original file names can be found in Appendix III.

6.1 Test 1: Evidence 1

Firstly, we look at how to document is shown and what we can find there, so Evidence 1 is opened with a PDF reader, in this case, Adobe Acrobat Reader. What can be captured from a fast check of the file is that, although it looks like a digitally created document, it does not recognise text or images (everything is recognised as the same image) and it does not have a typical form of rectangular page (unusual shape). If we take a look at the ‘Properties’ shown in Figure 23, this is what we can see some facts that may be catch our eye:



Description

File: 17.03.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf

Title:

Author:

Subject:

Keywords:

Created: 17.03.2017 10:49.09

Modified: 17.03.2017 10:53.09

Application: Adobe Photoshop CS2 Windows

Advanced

PDF Producer: Adobe Photoshop for Windows -- Image Conversion Plug-in

PDF Version: 1.4 (Acrobat 5.x)

Location: C:\Users\Ekspert\Desktop\RE_Methodology_description\

File Size: 1 018,62 KB (1 043 068 Bytes)

Page Size: 5,53 x 4,70 in Number of Pages: 1

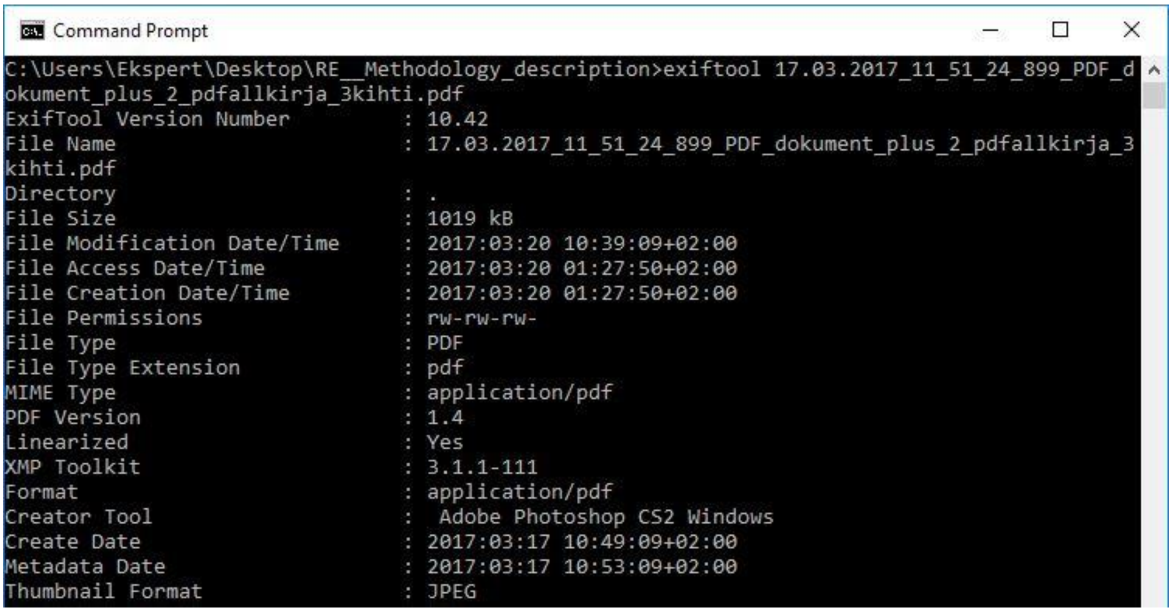
Tagged PDF: No Fast Web View: Yes

Figure 23. Properties shown by Adobe Acrobat Reader

- The size of the file is large for a one-page PDF file (1MB)
- Adobe Photoshop is the creator software
- Creation and modification dates do not match

Step 1: Metadata

Firstly, we check whether there are incremental updates or not; we use Notepad++ and find two cross-reference tables, one regular and one incremental update. Moreover, the extra cross-reference table is not empty, but has 20 objects on it. Afterwards, we check the trailer, where there is an */Info* object, but when we move to the document catalogue, we discover a */Metadata* object too. If we explore both metadata objects, we see the information in both is consistent, and Exiftool shows us all this information in an organised way (Figure 24).



```
C:\Users\Ekspert\Desktop\RE__Methodology_description>exiftool 17.03.2017_11_51_24_899_PDF_d
okument_plus_2_pdfallkirja_3kihti.pdf
ExifTool Version Number         : 10.42
File Name                       : 17.03.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3
kihti.pdf
Directory                       : .
File Size                       : 1019 kB
File Modification Date/Time     : 2017:03:20 10:39:09+02:00
File Access Date/Time           : 2017:03:20 01:27:50+02:00
File Creation Date/Time         : 2017:03:20 01:27:50+02:00
File Permissions                : rw-rw-rw-
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.4
Linearized                      : Yes
XMP Toolkit                     : 3.1.1-111
Format                          : application/pdf
Creator Tool                    :  Adobe Photoshop CS2 Windows
Create Date                     : 2017:03:17 10:49:09+02:00
Metadata Date                   : 2017:03:17 10:53:09+02:00
Thumbnail Format                : JPEG
```

Figure 24. Exiftool main output

Although we can see only the starting part of exiftool output, most relevant details are present. The information we can retrieve from its output is similar to the what we got from ‘Properties’, but in a most detailed way:

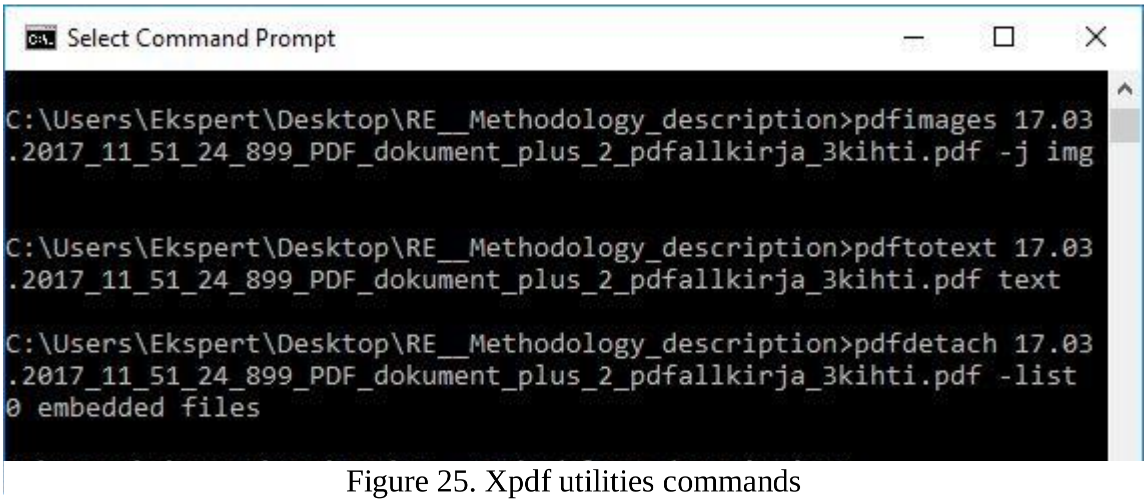
- The creator and producer software of the document is Adobe Photoshop.
- Creation and modification dates do not match, being separated by 4 minutes.

We also obtain a lot of metadata about colour profiles, patterns, thumbnails and dimensioning, which is not usual in PDF file metadata, but it looks instead more like the kind of metadata you could find on an image, for example.

Step 2: Content extraction

Now we proceed to extract all images, text and attachments following the procedures detailed in the method. For this, as stated previously, the utilities *pdfimages*, *pdftotext* and *pdfdetach* that Xpdf provides are used. The issued commands are shown in Figure 25. There are no attached files and not text found on the PDF file. However, *pdfimages* provides us with two different images:

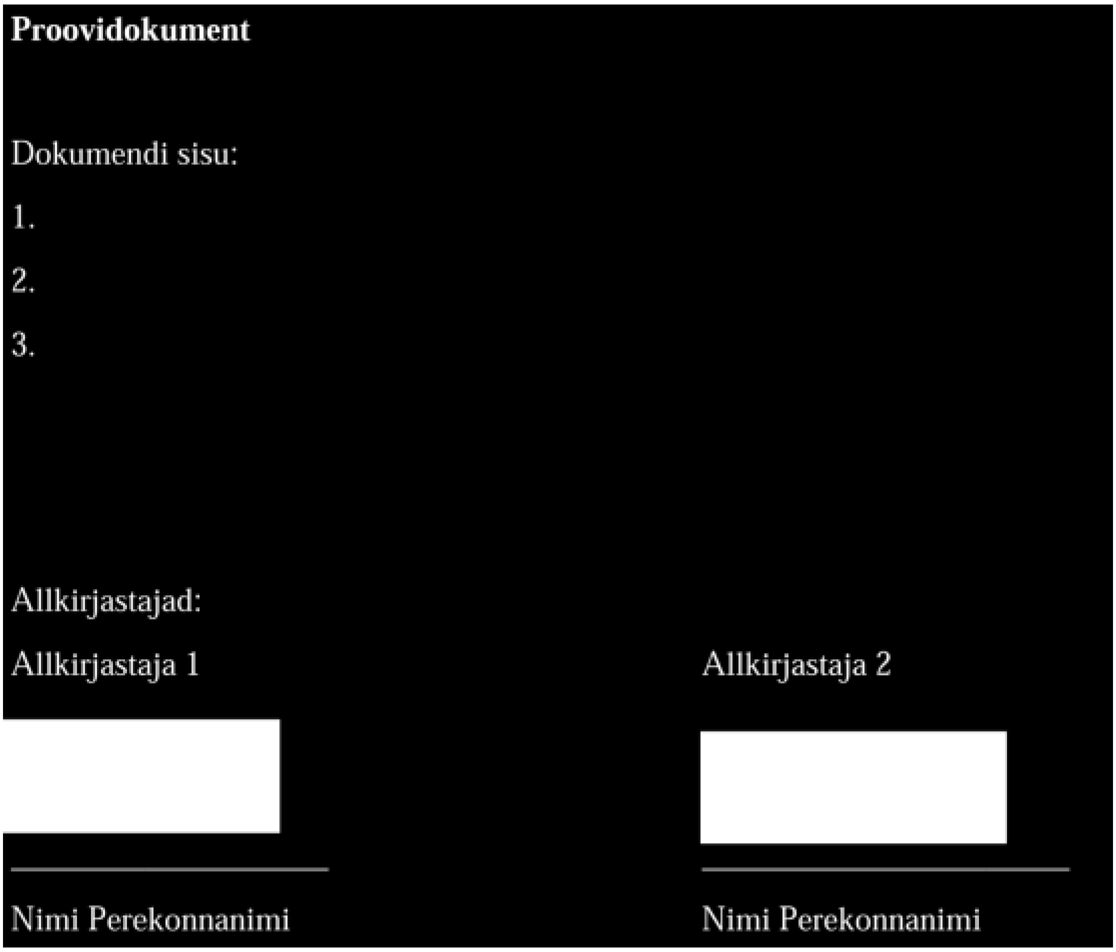
- First image corresponds to the ‘page image’, the image representing the whole content of the only page in the file, and its format is, as expected, JPG.
- Even though both should be in JPG format (as indicated with the option –j in the command), the second image remains in PPM format.



```
C:\Users\Ekspert\Desktop\RE__Methodology_description>pdfimages 17.03
.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf -j img


C:\Users\Ekspert\Desktop\RE__Methodology_description>pdftotext 17.03
.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf text

C:\Users\Ekspert\Desktop\RE__Methodology_description>pdfdetach 17.03
.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf -list
0 embedded files
```

Figure 25. Xpdf utilities commands

This means that the original image included in the PDF file had PPM format, an unusual one when it comes to PDF documents. It could not be opened with regular image viewers, that only accept JPG, JPEG and PNG image formats, so GIMP software was used, and, as shown in Figure 26, we can see that it looks like the page image but with the colours inverted and without the signatures that appear on the original. This looks a lot like the layer masks used in image editor software, and it make us think that two images of signatures might have been added afterwards.



Proovidokument

Dokumendi sisu:

1.

2.

3.

Allkirjastajad:

Allkirjastaja 1

Allkirjastaja 2

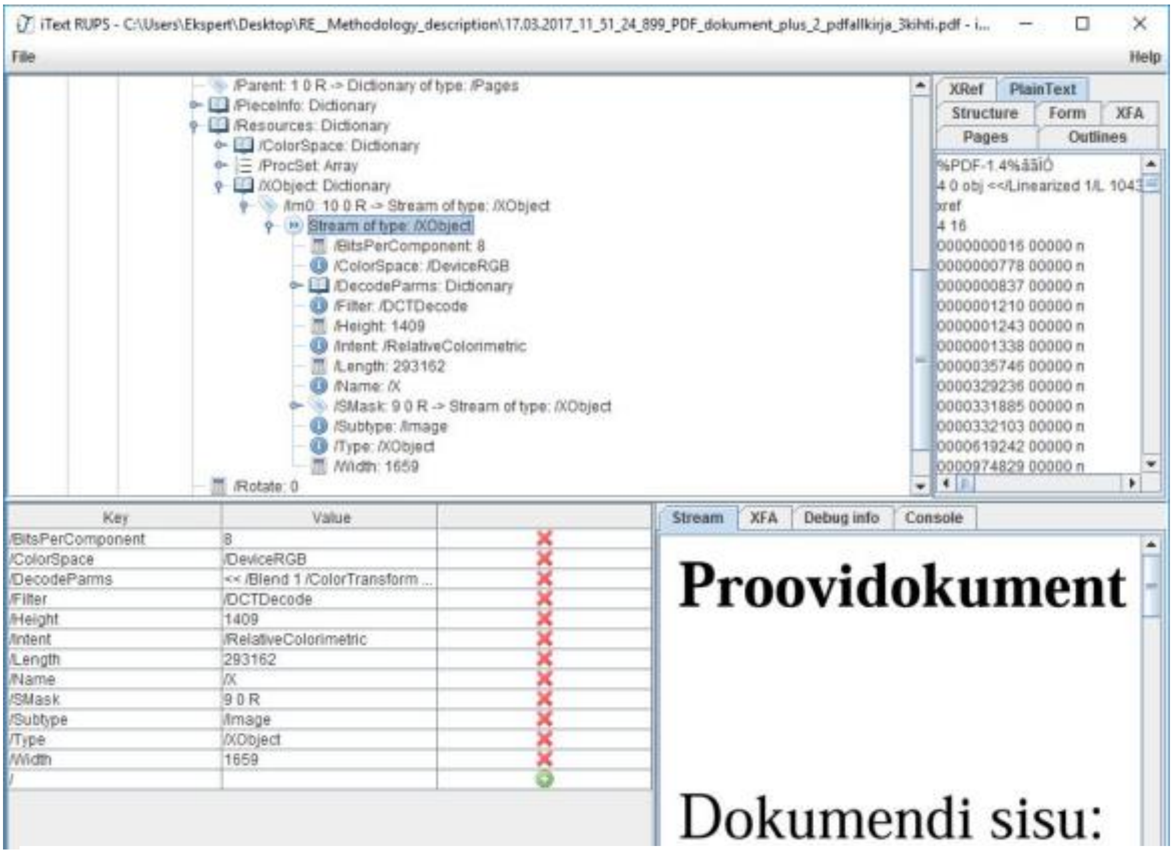
Nimi Perekonnanimi

Nimi Perekonnanimi

Figure 26. PPM image extracted from file

Step 3: Layout comparison and investigation

Finally, we address the source code of the file and start investigating the objects. Following the guiding flowchart, firstly we need to establish a reference layout. Except for the incremental update, at first sight it would look like a scanned file (no text recognised and no empty cross-reference table), so we will take it as a reference.



iText RUPS - C:\Users\Ekspert\Desktop\RE_Methodology_description\17.03.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf - i...

```
/Parent: 1 0 R -> Dictionary of type: /Pages
/PieceInfo: Dictionary
/Resources: Dictionary
  /ColorSpace: Dictionary
  /ProcSet: Array
  /XObject: Dictionary
    /Im0: 10 0 R -> Stream of type: /XObject
      Stream of type: /XObject
        /BitsPerComponent: 8
        /ColorSpace: /DeviceRGB
        /DecodeParms: Dictionary
        /Filter: /DCTDecode
        /Height: 1409
        /Intent: /RelativeColorimetric
        /Length: 293162
        /Name: /X
        /SMask: 9 0 R -> Stream of type: /XObject
        /Subtype: /Image
        /Type: /XObject
        /Width: 1659
/Rotate: 0
```

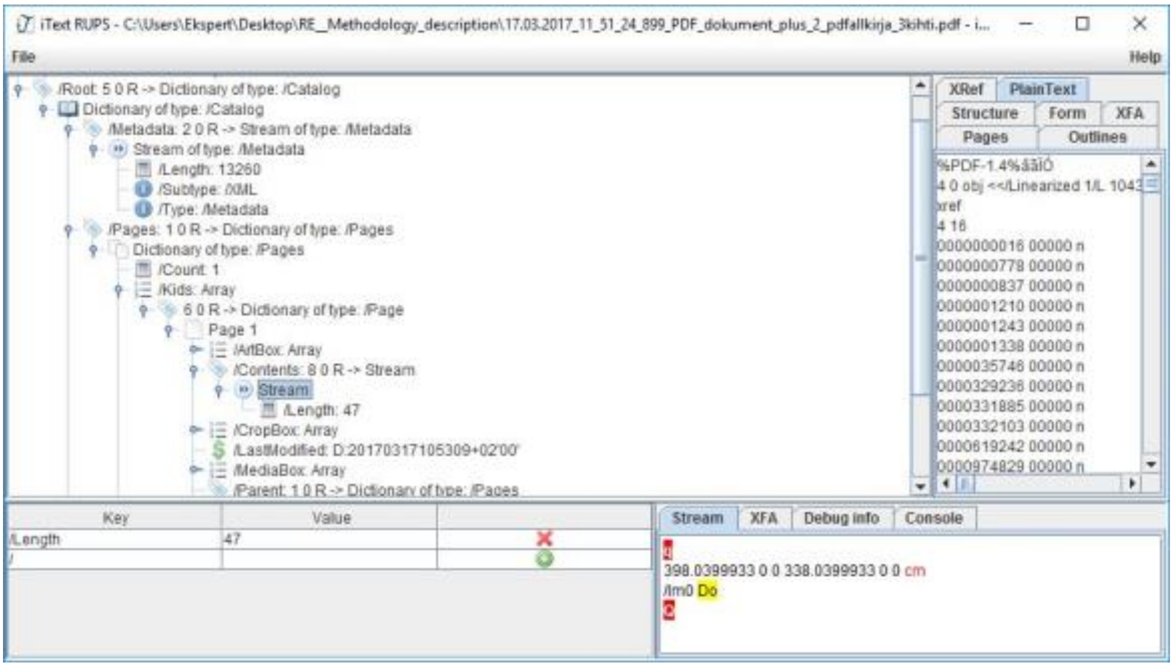
```
%PDF-1.4%âãÏÓ
4 0 obj <</Linearized 1/L 1043
xref
4 16
0000000016 00000 n
0000000778 00000 n
0000000837 00000 n
0000001210 00000 n
0000001243 00000 n
0000001338 00000 n
0000035746 00000 n
0000329236 00000 n
0000331885 00000 n
0000332103 00000 n
0000619242 00000 n
0000974829 00000 n
```

Key	Value
/BitsPerComponent	8
/ColorSpace	/DeviceRGB
/DecodeParms	<< /Blend 1 /ColorTransform ...
/Filter	/DCTDecode
/Height	1409
/Intent	/RelativeColorimetric
/Length	293162
/Name	/X
/SMask	9 0 R
/Subtype	/Image
/Type	/XObject
/Width	1659
/	

Proovidokument

Dokumendi sisu:

Figure 27. Page image shown by iText RUPS



iText RUPS - C:\Users\Ekspert\Desktop\RE_Methodology_description\17.03.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf - i...

```
/Root: 5 0 R -> Dictionary of type: /Catalog
  Dictionary of type: /Catalog
    /Metadata: 2 0 R -> Stream of type: /Metadata
      Stream of type: /Metadata
        /Length: 13260
        /Subtype: /XML
        /Type: /Metadata
    /Pages: 1 0 R -> Dictionary of type: /Pages
      Dictionary of type: /Pages
        /Count: 1
        /Kids: Array
          6 0 R -> Dictionary of type: /Page
            Page 1
              /ArtBox: Array
              /Contents: 8 0 R -> Stream
                Stream
                  /Length: 47
              /CropBox: Array
              /LastModified: D:20170317105309+02'00'
              /MediaBox: Array
              /Parent: 1 0 R -> Dictionary of type: /Pages
```

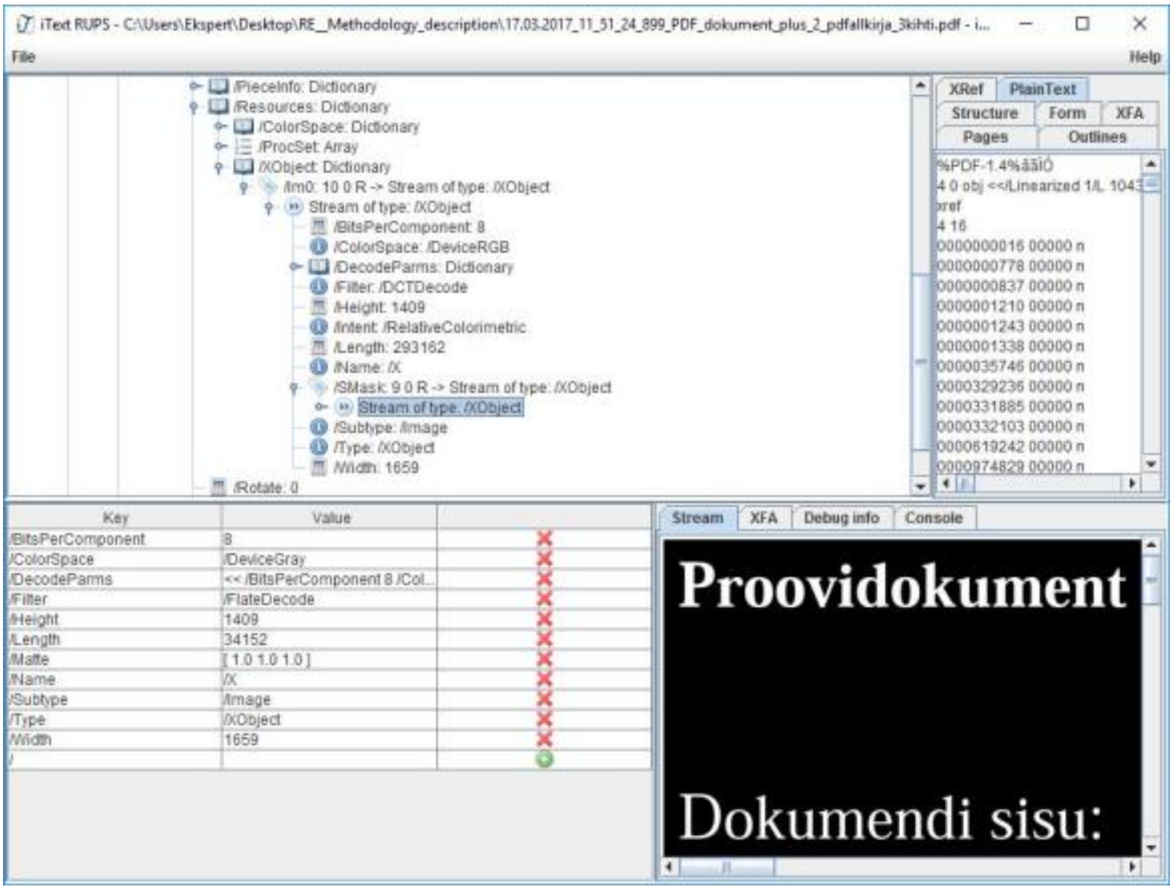
```
%PDF-1.4%âãÏÓ
4 0 obj <</Linearized 1/L 1043
xref
4 16
0000000016 00000 n
0000000778 00000 n
0000000837 00000 n
0000001210 00000 n
0000001243 00000 n
0000001338 00000 n
0000035746 00000 n
0000329236 00000 n
0000331885 00000 n
0000332103 00000 n
0000619242 00000 n
0000974829 00000 n
```

Key	Value
/Length	47
/	

```
q
398.0399933 0 0 338.0399933 0 0 cm
/Im0 Do
Q
```

Figure 28. Contents stream shown by iText RUPS

We identify the main objects present in the layout, which are the page image (Figure 27) and its positioning details (Figure 28); both references reside in the page leaf (in */Resources* and in */Content* respectively). Moreover, one of the images found in the content extraction coincides with the image in the */XObject* within the page leaf (page image).


iText RUPS - C:\Users\Ekspert\Desktop\RE_Methodology_description\17.03.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf - i...
File
Help
XRef
PlainText
Structure
Form
XFA
Pages
Outlines
Stream
XFA
Debug info
Console
Proovidokument
Dokumendi sisu:

```
/PieceInfo: Dictionary
/Resources: Dictionary
  /ColorSpace: Dictionary
  /ProcSet: Array
  /XObject: Dictionary
    /Im0: 10 0 R -> Stream of type: /XObject
      Stream of type: /XObject
        /BitsPerComponent: 8
        /ColorSpace: /DeviceRGB
        /DecodeParms: Dictionary
        /Filter: /DCTDecode
        /Height: 1409
        /Intent: /RelativeColorimetric
        /Length: 293162
        /Name: /X
        /SMask: 9 0 R -> Stream of type: /XObject
          Stream of type: /XObject
        /Subtype: /Image
        /Type: /XObject
        /Width: 1659
/Rotate: 0
```

```
%PDF-1.4%âãÏÓ
4 0 obj <</Linearized 1/L 1043
xref
4 16
0000000016 00000 n
0000000778 00000 n
0000000837 00000 n
0000001210 00000 n
0000001243 00000 n
0000001338 00000 n
0000035746 00000 n
0000329236 00000 n
0000331885 00000 n
0000332103 00000 n
0000619242 00000 n
0000974829 00000 n
```

Key	Value	
/BitsPerComponent	8	✗
/ColorSpace	/DeviceGray	✗
/DecodeParms	<< /BitsPerComponent 8 /Col...	✗
/Filter	/FlateDecode	✗
/Height	1409	✗
/Length	34152	✗
/Matte	[1.0 1.0 1.0]	✗
/Name	/X	✗
/Subtype	/Image	✗
/Type	/XObject	✗
/Width	1659	✗
/		⊕

Figure 29. Mask image shown by iText RUPS

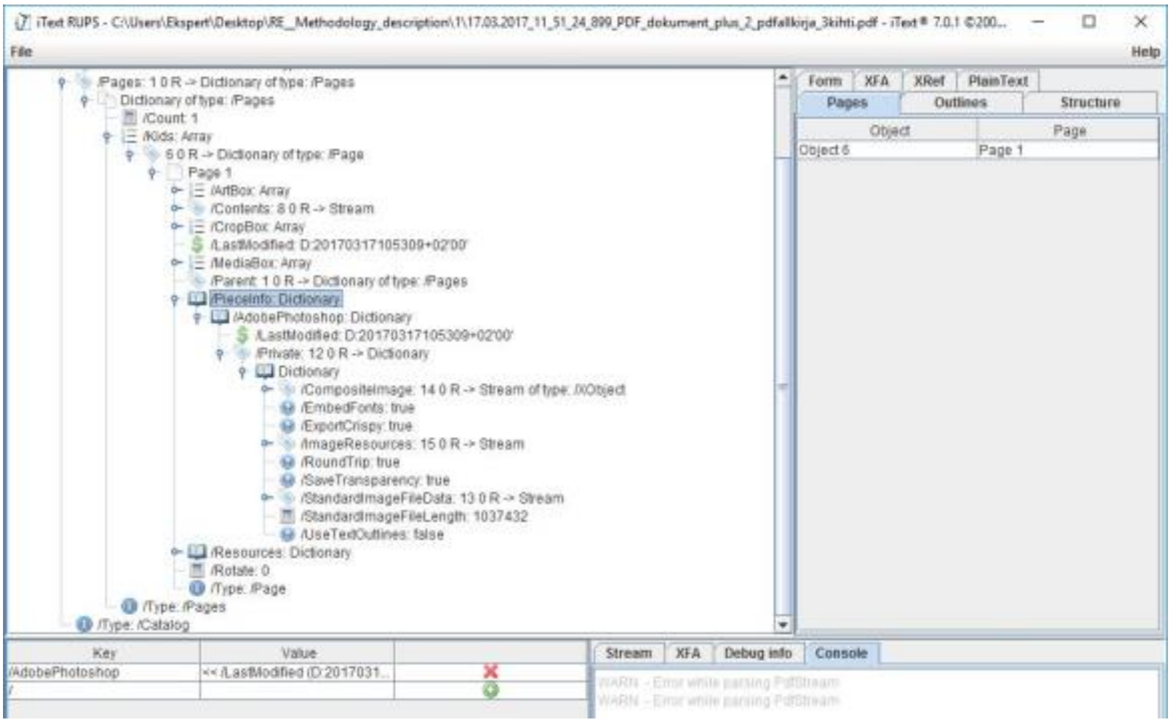
Once the reference is established, we look for extra objects, and we find a suspicious image within the attributes of the */XObject* of the page image (Figure 29). It matches with the other image we found in the content extraction shown in Figure 26.

Moreover, an extra attribute in the page leaf exists, called */PieceInfo*, which corresponds to an attribute added by the creator and which includes a */Private* dictionary, shown in Figure 30, with private product data about the creator/producer software, in this case, Adobe Photoshop.

Now it is time to add all the clues we found along the investigations:

- Excessive size for a one-page PDF file
- Adobe Photoshop is the creator/producer of the file
- Creation and modification dates do not match (4 minutes difference)
- Looks like a scanned PDF file, but it has a suspicious image apart from the page image which lacks the signatures that appear on the file.
- Continuous traces of Adobe Photoshop actions to the file

Finally, we can conclude that this image has been manipulated with Adobe Photoshop software, and the manipulation consists in adding the two signatures present in the document with the help of this software.


iText RUPS - C:\Users\Ekspert\Desktop\RE_Methodology_description\1\17.03.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf - iText® 7.0.1 ©200...
File
Help
Form
XFA
XRef
PlainText
Pages
Outlines
Structure
Stream
XFA
Debug info
Console

```
/Pages: 1 0 R -> Dictionary of type: /Pages
  Dictionary of type: /Pages
    /Count: 1
    /Kids: Array
      6 0 R -> Dictionary of type: /Page
        Page 1
          /ArtBox: Array
          /Contents: 8 0 R -> Stream
          /CropBox: Array
          /LastModified: D:20170317105309+02'00'
          /MediaBox: Array
          /Parent: 1 0 R -> Dictionary of type: /Pages
          /PieceInfo: Dictionary
            /AdobePhotoshop: Dictionary
              /LastModified: D:20170317105309+02'00'
              /Private: 12 0 R -> Dictionary
                Dictionary
                  /CompositeImage: 14 0 R -> Stream of type: /XObject
                  /EmbedFonts: true
                  /ExportCrispy: true
                  /ImageResources: 15 0 R -> Stream
                  /RoundTrip: true
                  /SaveTransparency: true
                  /StandardImageFileData: 13 0 R -> Stream
                  /StandardImageFileLength: 1037432
                  /UseTextOutlines: false
          /Resources: Dictionary
          /Rotate: 0
          /Type: /Page
    /Type: /Pages
/Type: /Catalog
```

Object	Page
Object 6	Page 1

Key	Value	
/AdobePhotoshop	<< /LastModified (D:2017031...	✗
/		⊕

```
WARN - Error while parsing PdfStream
WARN - Error while parsing PdfStream
```

Figure 30. /PieceInfo and /Private dictionaries

6.2 Test 4: Evidence 4

This file investigation differed considerably with the previous one, that is why it was also included in the main document. When we look at Evidence 4 with Adobe Acrobat Reader, we notice that text and images are recognised and selectable. Moreover, the existent image corresponds to a manually written signature, which is the first reason of suspicion.

Step 1: Metadata

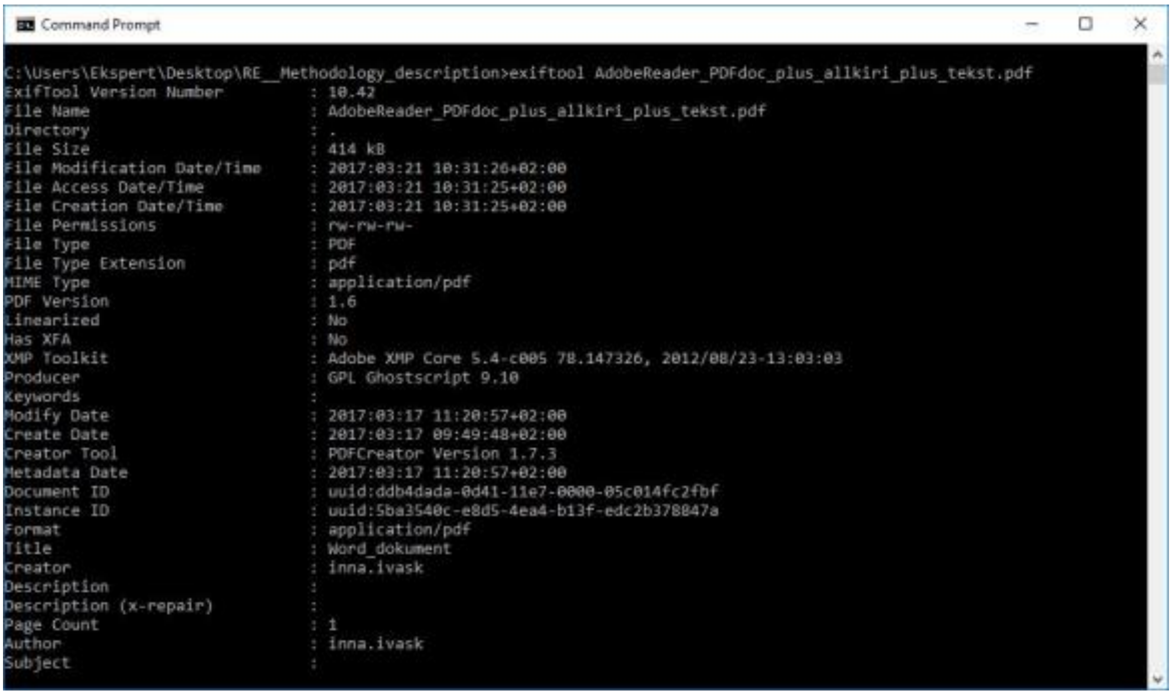
When we look at the metadata with Exiftool, shown in Figure 31, we can observe the following:

- Creation and modification dates differ one and a half hour.
- Fields like 'Author' and 'Title' are present.
- Creator tool corresponds to PDFCreator, which means that this file was digitally created with another software and then converted to PDF.
- Producer tool corresponds to GPL[20] Ghostscript, which is an interpreter for both PostScript language and PDF code. It is used by many tools that create and modify PDF files to translate the modifications made to a correctly formed PDF document, as it is certainly the case of PDFCreator,

Moreover, when we investigate the source code, we realise that there are no incremental updates, so it still matches with the reference layout 'Digital file converted to PDF'.

In the source code, we find also two different metadata objects, one */Info* object and one */Metadata object*. However, they both contain compatible information, and it coincides with what Exiftool shows us:

[20] General Public License



```
C:\Users\Ekspert\Desktop\RE__Methodology_description>exiftool AdobeReader_PDFdoc_plus_allkiri_plus_tekst.pdf
ExifTool Version Number         : 10.42
File Name                       : AdobeReader_PDFdoc_plus_allkiri_plus_tekst.pdf
Directory                       : .
File Size                       : 414 kB
File Modification Date/Time     : 2017:03:21 10:31:26+02:00
File Access Date/Time           : 2017:03:21 10:31:25+02:00
File Creation Date/Time         : 2017:03:21 10:31:25+02:00
File Permissions                : rw-rw-rw-
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.6
Linearized                      : No
Has XFA                         : No
XMP Toolkit                     : Adobe XMP Core 5.4-c005 78.147326, 2012/08/23-13:03:03
Producer                        : GPL Ghostscript 9.10
Keywords                        :
Modify Date                     : 2017:03:17 11:20:57+02:00
Create Date                     : 2017:03:17 09:49:48+02:00
Creator Tool                    : PDFCreator Version 1.7.3
Metadata Date                   : 2017:03:17 11:20:57+02:00
Document ID                     : uuid:ddb4dada-0d41-11e7-0000-05c014fc2fbf
Instance ID                     : uuid:5ba3540c-e8d5-4ea4-b13f-edc2b378847a
Format                          : application/pdf
Title                           : Word_dokument
Creator                         : inna.ivask
Description                     :
Description (x-repair)          :
Page Count                      : 1
Author                          : inna.ivask
Subject                         :
```

Figure 31. Exiftool output for Evidence 4

Step 2: Content extraction

When we perform the extraction with Xpdf utilities as done before, we obtain:

- 2 images, which both correspond to the signature present in the file, one as we see it, and another with the colours inverted, which we can suspect it is the mask image.

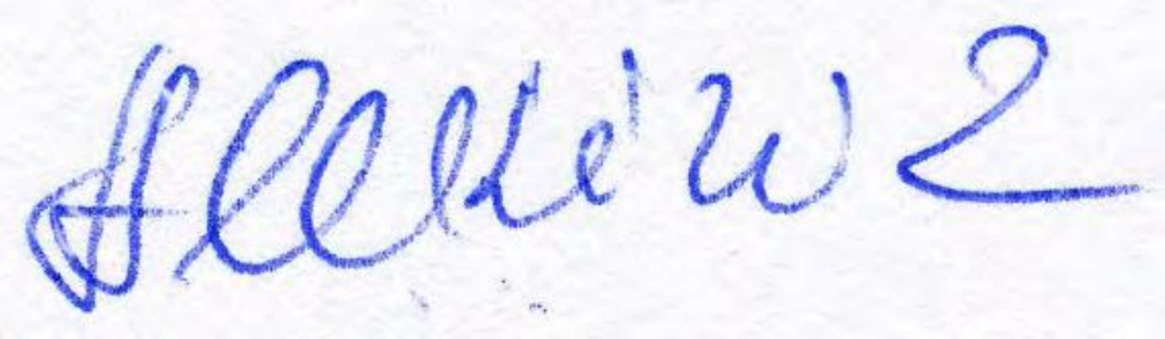


Figure 32. Extracted image 1

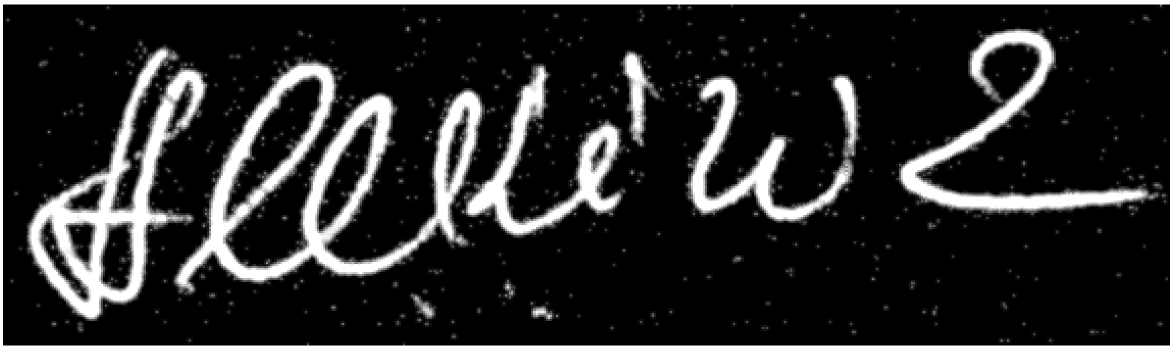
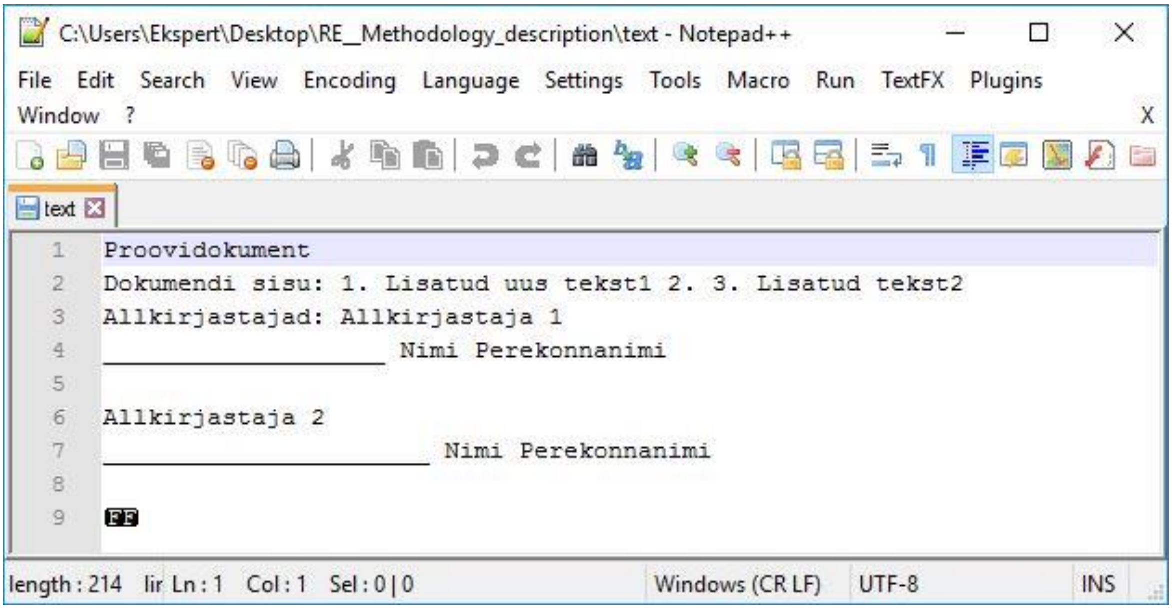


Figure 33. Extracted image 2

- A text document with all the text we could see in the PDF file.

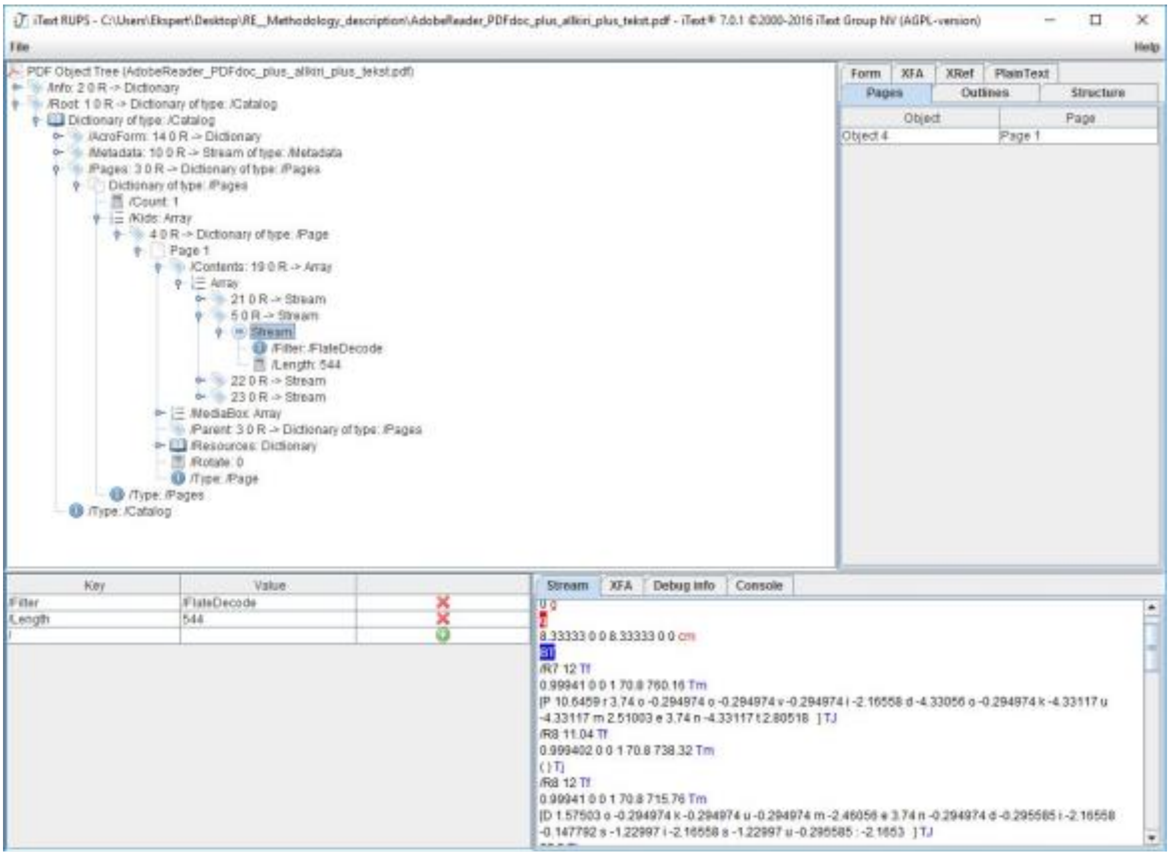


```
Proovidokument
Dokumendi sisu: 1. Lisatud uus tekst1 2. 3. Lisatud tekst2
Allkirjastajad: Allkirjastaja 1
___________________ Nimi Perekonnanimi

Allkirjastaja 2
______________________ Nimi Perekonnanimi

FF
```

Figure 34. Extracted text

Step 3: Layout comparison and investigation

Every clue we found in previous steps of the investigation points towards this file being created by a text editor and then converted to PDF. Now, we look at the objects to confirm that they match with the ones in the reference layout.


iText RUPS - C:\Users\Ekspert\Desktop\RE__Methodology_description\AdobeReader_PDFdoc_plus_allkiri_plus_tekst.pdf - iText® 7.0.1 ©2000-2016 iText Group NV (AGPL-version)
File
Help
Form
XFA
XRef
PlainText
Pages
Outlines
Structure
Stream
XFA
Debug info
Console

```
PDF Object Tree (AdobeReader_PDFdoc_plus_allkiri_plus_tekst.pdf)
  /Info: 2 0 R -> Dictionary
  /Root: 1 0 R -> Dictionary of type: /Catalog
    Dictionary of type: /Catalog
      /AcroForm: 14 0 R -> Dictionary
      /Metadata: 10 0 R -> Stream of type: /Metadata
      /Pages: 3 0 R -> Dictionary of type: /Pages
        Dictionary of type: /Pages
          /Count: 1
          /Kids: Array
            4 0 R -> Dictionary of type: /Page
              Page 1
                /Contents: 19 0 R -> Array
                  Array
                    21 0 R -> Stream
                    5 0 R -> Stream
                      Stream
                        /Filter: /FlateDecode
                        /Length: 544
                    22 0 R -> Stream
                    23 0 R -> Stream
                /MediaBox: Array
                /Parent: 3 0 R -> Dictionary of type: /Pages
                /Resources: Dictionary
                /Rotate: 0
                /Type: /Page
          /Type: /Pages
      /Type: /Catalog
```

Object	Page
Object 4	Page 1

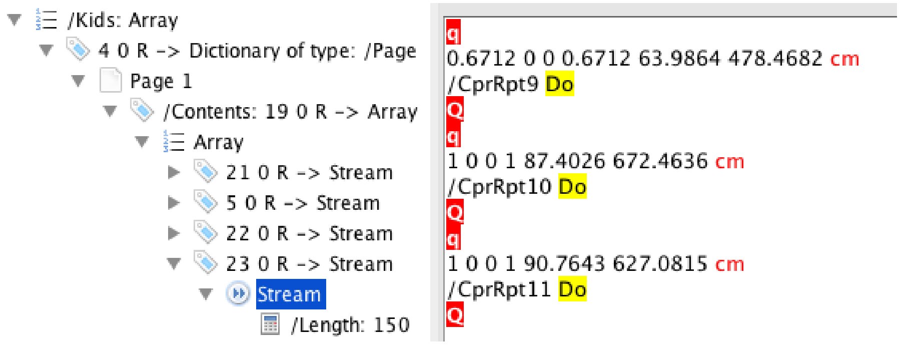
Key	Value
/Filter	/FlateDecode
/Length	544
/	

```
0 g
q
8.33333 0 0 8.33333 0 0 cm
BT
/R7 12 Tf
0.99941 0 0 1 70.8 760.16 Tm
[P 10.6459 r 3.74 o -0.294974 o -0.294974 v -0.294974 i -2.16558 d -4.33056 o -0.294974 k -4.33117 u
-4.33117 m 2.51003 e 3.74 n -4.33117 t 2.80518  ] TJ
/R8 11.04 Tf
0.999402 0 0 1 70.8 738.32 Tm
( ) Tj
/R8 12 Tf
0.99941 0 0 1 70.8 715.76 Tm
[D 1.57503 o -0.294974 k -0.294974 u -0.294974 m -2.46056 e 3.74 n -0.294974 d -0.295585 i -2.16558
-0.147792 s -1.22997 i -2.16558 s -1.22997 u -0.295585 : -2.1653  ] TJ
```

Figure 35. Stream text object

The object */Contents* points towards several text objects:

- Two of them specifying image positioning details.
- One of them specifying the text within the document, shown in Figure 35. However, if we check the text shown in this stream and the text extracted with *pdftotext*, there are two sentences missing:
  - `Lisatud uus text1`
  - `Lisatud uus text2`
- Last one specifying positioning details of three media objects, shown in Figure 36



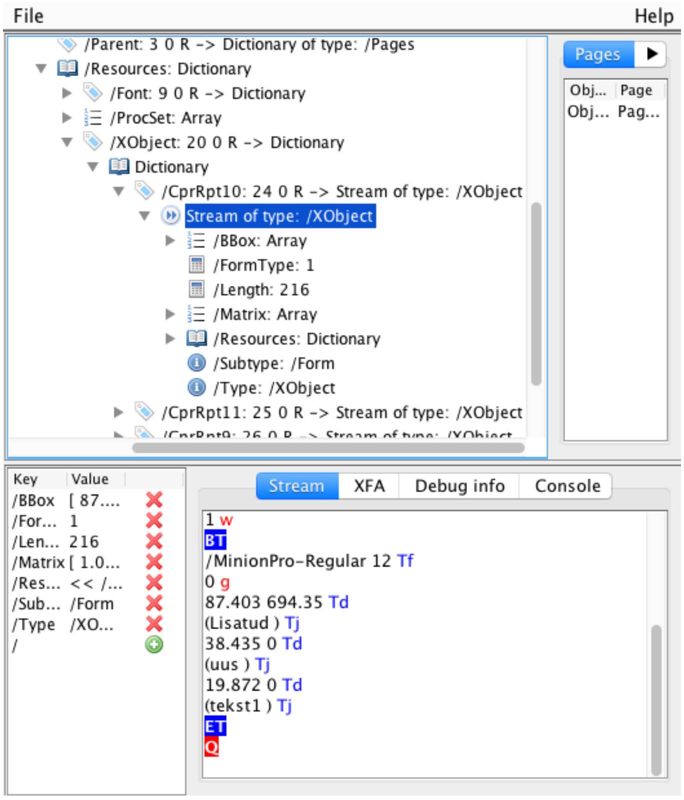
```
▼ /Kids: Array
  ▼ 4 0 R -> Dictionary of type: /Page
    ▼ Page 1
      ▼ /Contents: 19 0 R -> Array
        ▼ Array
          ▶ 21 0 R -> Stream
          ▶ 5 0 R -> Stream
          ▶ 22 0 R -> Stream
          ▼ 23 0 R -> Stream
            ▼ Stream
                /Length: 150
```

```
q
0.6712 0 0 0.6712 63.9864 478.4682 cm
/CprRpt9 Do
Q
q
1 0 0 1 87.4026 672.4636 cm
/CprRpt10 Do
Q
q
1 0 0 1 90.7643 627.0815 cm
/CprRpt11 Do
Q
```

Figure 36. Media objects positioning details found in /Contents

In */Resources* attribute, we find a reference to a */XObject* containing another 3 (*/XObject)*s:

- It turns out that the two first ones */CprRpt9* and */CprRpt10* correspond to two text objects with the sentences that were missing in the main text object, which may indicate that were added afterwards.

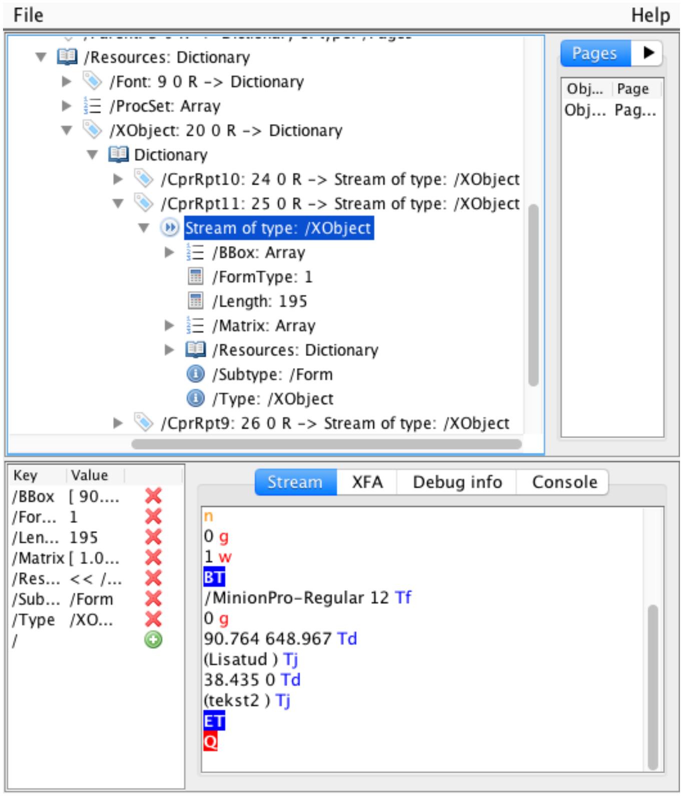


```
    /Parent: 3 0 R -> Dictionary of type: /Pages
▼ /Resources: Dictionary
  ▶ /Font: 9 0 R -> Dictionary
  ▶ /ProcSet: Array
  ▼ /XObject: 20 0 R -> Dictionary
    ▼ Dictionary
      ▼ /CprRpt10: 24 0 R -> Stream of type: /XObject
        ▼ Stream of type: /XObject
          ▶ /BBox: Array
            /FormType: 1
            /Length: 216
          ▶ /Matrix: Array
          ▶ /Resources: Dictionary
            /Subtype: /Form
            /Type: /XObject
      ▶ /CprRpt11: 25 0 R -> Stream of type: /XObject
      ▶ /CprRpt9: 26 0 R -> Stream of type: /XObject
```

Key	Value	
/BBox	[87....	✗
/For...	1	✗
/Len...	216	✗
/Matrix	[1.0...	✗
/Res...	<< /...	✗
/Sub...	/Form	✗
/Type	/XO...	✗
/		⊕

```
1 w
BT
/MinionPro-Regular 12 Tf
0 g
87.403 694.35 Td
(Lisatud ) Tj
38.435 0 Td
(uus ) Tj
19.872 0 Td
(tekst1 ) Tj
ET
Q
```

Figure 37. Text box 1 shown by iText RUPS



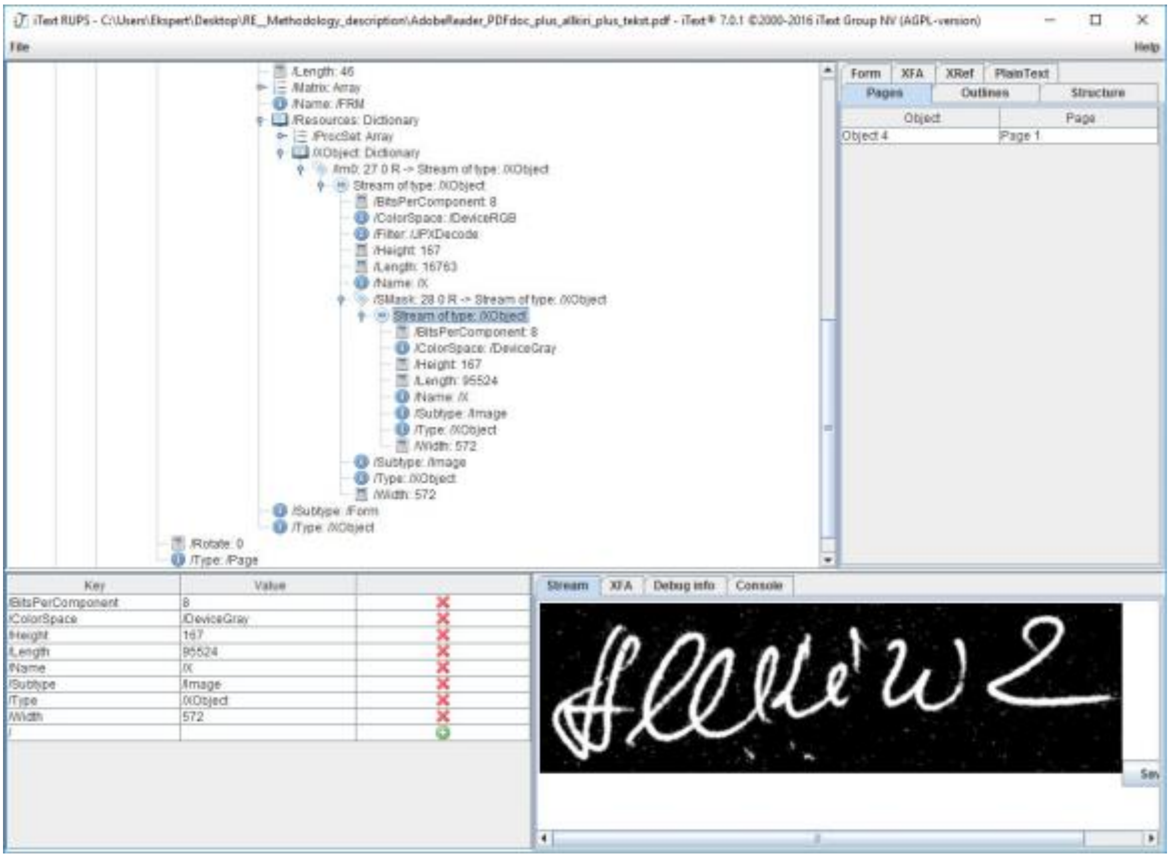
```
▼ /Resources: Dictionary
  ▶ /Font: 9 0 R -> Dictionary
  ▶ /ProcSet: Array
  ▼ /XObject: 20 0 R -> Dictionary
    ▼ Dictionary
      ▶ /CprRpt10: 24 0 R -> Stream of type: /XObject
      ▼ /CprRpt11: 25 0 R -> Stream of type: /XObject
        ▼ Stream of type: /XObject
          ▶ /BBox: Array
            /FormType: 1
            /Length: 195
          ▶ /Matrix: Array
          ▶ /Resources: Dictionary
            /Subtype: /Form
            /Type: /XObject
      ▶ /CprRpt9: 26 0 R -> Stream of type: /XObject
```

Key	Value	
/BBox	[90....	✗
/For...	1	✗
/Len...	195	✗
/Matrix	[1.0...	✗
/Res...	<< /...	✗
/Sub...	/Form	✗
/Type	/XO...	✗
/		⊕

```
n
0 g
1 w
BT
/MinionPro-Regular 12 Tf
0 g
90.764 648.967 Td
(Lisatud ) Tj
38.435 0 Td
(tekst2 ) Tj
ET
Q
```

Figure 38. Text box 2 shown by iText RUPS

- The third one corresponds with the signature image we found in the extraction. Within the */XObject*, an attribute */SMask* appears with the signature image that had the colours inverted, shown in Figure 39. We already saw this in Test 1, and we associated it with the handling of images.


iText RUPS - C:\Users\Ekspert\Desktop\RE_Methodology_description\AdobeReader_PDFdoc_plus_allkiri_plus_tekst.pdf - iText® 7.0.1 ©2000-2016 iText Group NV (AGPL-version)
File
Help
Form
XFA
XRef
PlainText
Pages
Outlines
Structure
Stream
XFA
Debug info
Console
Sav

```
/Length: 46
/Matrix: Array
/Name: /FRM
/Resources: Dictionary
  /ProcSet: Array
  /XObject: Dictionary
    /Im0: 27 0 R -> Stream of type: /XObject
      Stream of type: /XObject
        /BitsPerComponent: 8
        /ColorSpace: /DeviceRGB
        /Filter: /JPXDecode
        /Height: 167
        /Length: 16763
        /Name: /X
        /SMask: 28 0 R -> Stream of type: /XObject
          Stream of type: /XObject
            /BitsPerComponent: 8
            /ColorSpace: /DeviceGray
            /Height: 167
            /Length: 95524
            /Name: /X
            /Subtype: /Image
            /Type: /XObject
            /Width: 572
        /Subtype: /Image
        /Type: /XObject
        /Width: 572
/Subtype: /Form
/Type: /XObject
/Rotate: 0
/Type: /Page
```

Object	Page
Object 4	Page 1

Key	Value	
/BitsPerComponent	8	✗
/ColorSpace	/DeviceGray	✗
/Height	167	✗
/Length	95524	✗
/Name	/X	✗
/Subtype	/Image	✗
/Type	/XObject	✗
/Width	572	✗
/		⊕

Figure 39. /SMask attribute with colour-inverted image

So, if we gather what we have found:

- Different creation and modification dates.
- PDFCreator traces.
- Document traces (author, title…).
- 2 images (signature normal and inverted colours), one residing in the */SMask* attribute of the other.
- 1 text object and 2 text boxes with part of the extracted text.

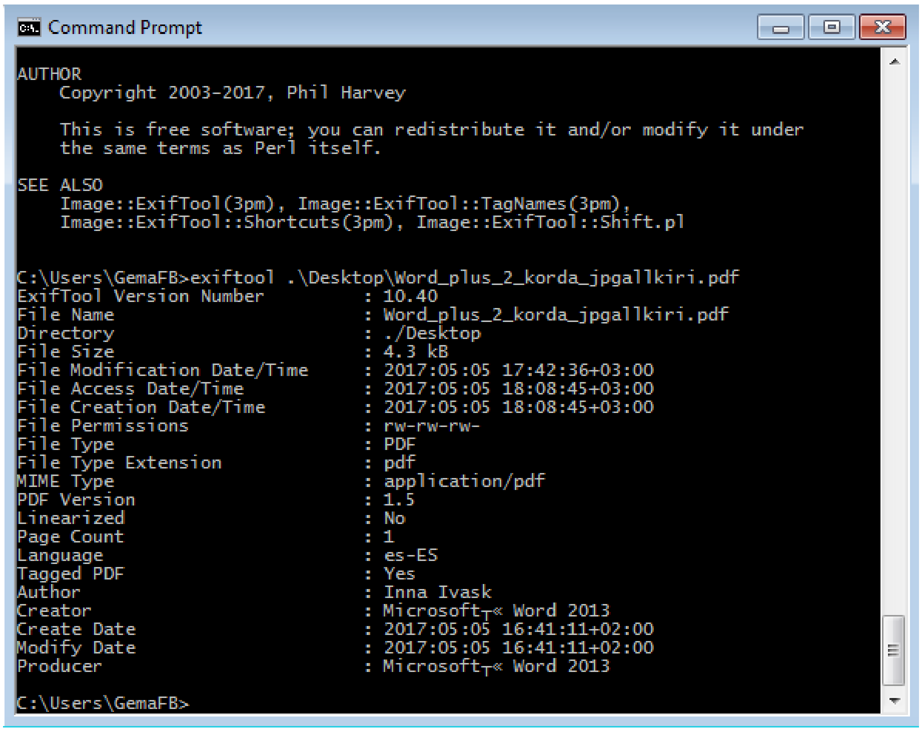
I think we can conclude this file was converted to PDF by PDFCreator, having had previously another format. This previous file was created by a text editor, text was added and the signature was included as an image. Nevertheless, in this case it is suspicious to add text with a text box in a list that could have been filled up adding text normally. Therefore, we can say that this text has been added subsquently to the creation of the PDF file. And in consequence, given the resemblance of the */XObjects* of the added text boxes and the image, we can determinate that the image with the signature was also added afterwards.

6.3 Test 9: Evidence 9

This file investigation differed considerably with the rest, that is why it was also included in the main document. When we take a look at Evidence 9 with Adobe Acrobat Reader, we notice than there are no images, and text is recognised and selectable.

Step 1: Metadata

When we look at the metadata with Exiftool, we can observe the following:



```
AUTHOR
    Copyright 2003-2017, Phil Harvey

    This is free software; you can redistribute it and/or modify it under
    the same terms as Perl itself.

SEE ALSO
    Image::ExifTool(3pm), Image::ExifTool::TagNames(3pm),
    Image::ExifTool::Shortcuts(3pm), Image::ExifTool::Shift.pl


C:\Users\GemaFB>exiftool .\Desktop\Word_plus_2_korda_jpgallkiri.pdf
ExifTool Version Number         : 10.40
File Name                       : Word_plus_2_korda_jpgallkiri.pdf
Directory                       : ./Desktop
File Size                       : 4.3 kB
File Modification Date/Time     : 2017:05:05 17:42:36+03:00
File Access Date/Time           : 2017:05:05 18:08:45+03:00
File Creation Date/Time         : 2017:05:05 18:08:45+03:00
File Permissions                : rw-rw-rw-
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.5
Linearized                      : No
Page Count                      : 1
Language                        : es-ES
Tagged PDF                      : Yes
Author                          : Inna Ivask
Creator                         : Microsoft┬« Word 2013
Create Date                     : 2017:05:05 16:41:11+02:00
Modify Date                     : 2017:05:05 16:41:11+02:00
Producer                        : Microsoft┬« Word 2013

C:\Users\GemaFB>
```

Figure 40. Exiftool output

- Creation and modification dates coincide.
- Fields like 'Author' and 'Title' are present.
- Creator and Producer tool correspond to Microsoft Word 2013, which means that this file was digitally created and then converted to PDF.

Moreover, when we investigate the source code, we realise that there are is an empty incremental update, so it matches with the reference layout 'Digital file converted to PDF'.

In the source code, we find only one */Info* metadata object, containing compatible information with the one observed with Adobe Acrobat Reader and in Exiftool.

Step 2: Content extraction

When we perform the extraction with Xpdf utilities as done before, we obtain:

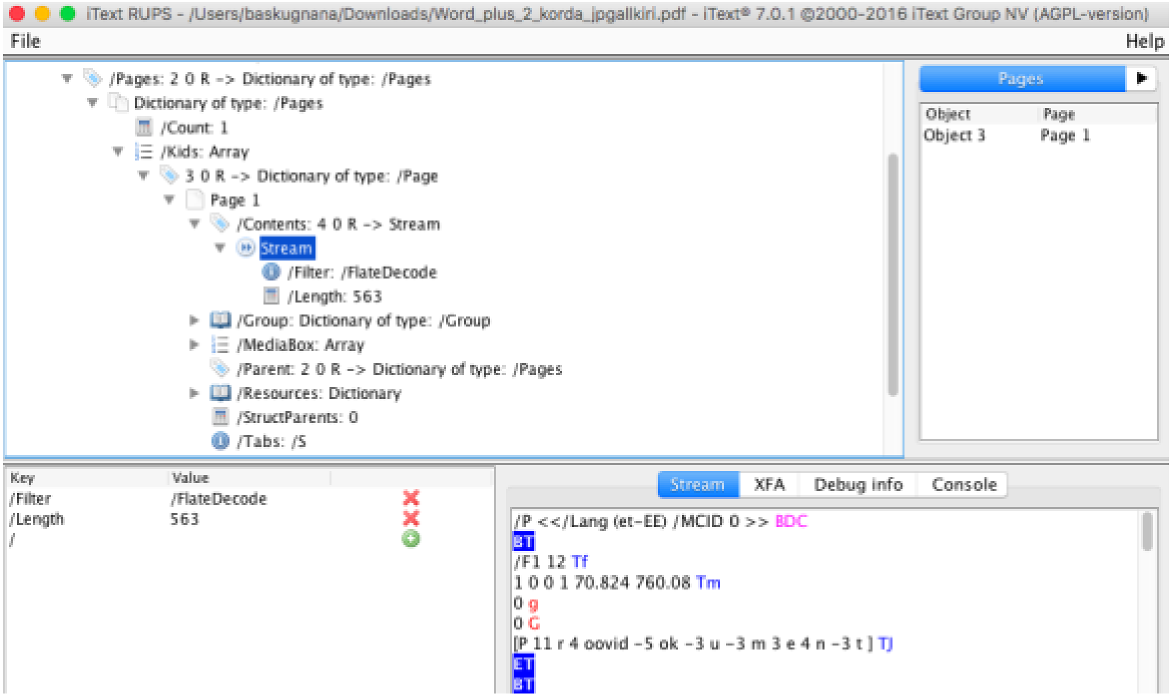
- No images nor embedded files.
- A text document with all the text we can see in the PDF file.

Step 3: Layout comparison and investigation

Every clue we found in previous steps of the investigation points towards this file being created by Microsoft Word 2013 and then converted to PDF. Now, we look at the objects to confirm that they match with the ones in the reference layout, using iText RUPS tool.

The object */Contents*, as shown in Figure 41, points towards one text object, containing all the observable text both in Adobe Acrobat Reader and in the content extraction output file.

In */Resources* attribute, as shown in Figure 42, we find references to two Font objects, which correspond to font types *Times New Roman* and *Times New Roman Bold*, and are inside the group of objects you may find in a non-manipulated PDF file of the kind 'converted to PDF'.



iText RUPS - /Users/baskugnana/Downloads/Word_plus_2_korda_jpgallkiri.pdf - iText® 7.0.1 ©2000-2016 iText Group NV (AGPL-version)

```
▼ /Pages: 2 0 R -> Dictionary of type: /Pages
  ▼ Dictionary of type: /Pages
      /Count: 1
    ▼ /Kids: Array
      ▼ 3 0 R -> Dictionary of type: /Page
        ▼ Page 1
          ▼ /Contents: 4 0 R -> Stream
            ▼ Stream
                /Filter: /FlateDecode
                /Length: 563
          ▶ /Group: Dictionary of type: /Group
          ▶ /MediaBox: Array
            /Parent: 2 0 R -> Dictionary of type: /Pages
          ▶ /Resources: Dictionary
            /StructParents: 0
            /Tabs: /S
```

Object	Page
Object 3	Page 1

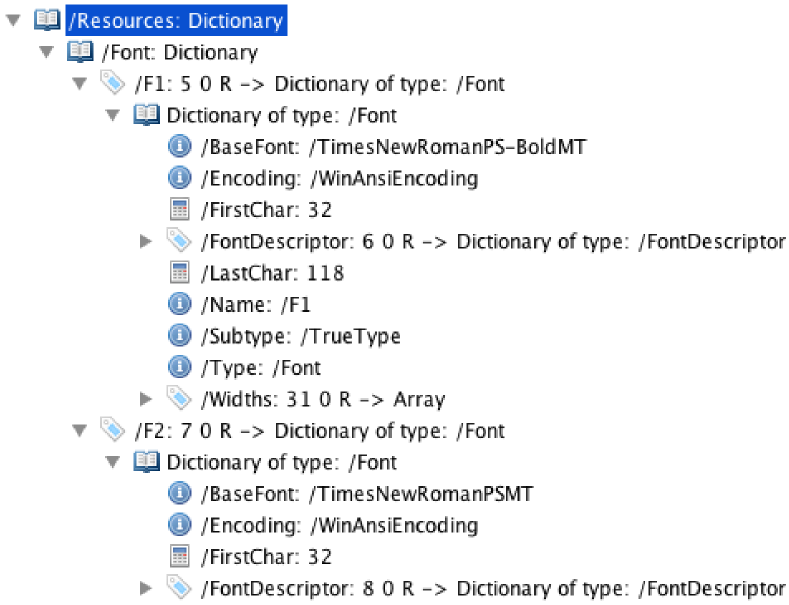
Key	Value
/Filter	/FlateDecode
/Length	563
/	

```
/P <</Lang (et-EE) /MCID 0 >> BDC
BT
/F1 12 Tf
1 0 0 1 70.824 760.08 Tm
0 g
0 G
[P 11 r 4 oovid -5 ok -3 u -3 m 3 e 4 n -3 t ] TJ
ET
BT
```

Figure 41. Text stream shown by iText RUPS

So, if we gather what we have found:

- Same creation and modification dates.
- Microsoft Word 2013 traces.
- Document traces (author, title…).
- 1 text object with all visible text included and 2 font objects.



```
▼ /Resources: Dictionary
  ▼ /Font: Dictionary
    ▼ /F1: 5 0 R -> Dictionary of type: /Font
      ▼ Dictionary of type: /Font
          /BaseFont: /TimesNewRomanPS-BoldMT
          /Encoding: /WinAnsiEncoding
          /FirstChar: 32
        ▶ /FontDescriptor: 6 0 R -> Dictionary of type: /FontDescriptor
          /LastChar: 118
          /Name: /F1
          /Subtype: /TrueType
          /Type: /Font
        ▶ /Widths: 31 0 R -> Array
    ▼ /F2: 7 0 R -> Dictionary of type: /Font
      ▼ Dictionary of type: /Font
          /BaseFont: /TimesNewRomanPSMT
          /Encoding: /WinAnsiEncoding
          /FirstChar: 32
        ▶ /FontDescriptor: 8 0 R -> Dictionary of type: /FontDescriptor
```

Figure 42. Resources object referencing Font objects

I think we can conclude this file was digitally created by Microsoft Word 2013 software, and then converted to PDF with this same software, using the option ‘save to PDF’. There are no traces of manipulation whatsoever, and all the existent objects coincide with the corresponding layout.

6.4 Results compilation and analysis

After applying the created method against typically manipulated files, we are going to present and evaluate the obtained results in order to assess the efficiency of the method. Firstly, a summary of the findings and conclusions is presented, and afterwards a more extended evaluation of this results is elaborated. As expressed before in this same document, validation cases are limited to the usual types of manipulation the experts encounter regarding digital evidence from court cases.

Findings compilation

Once all the evidences have been investigated, a lot of resemblances were found among some of them, so only three of the processes have been included in the main document as illustrative. Now, and in order to evaluate these similarities and to assess the findings, Table 2 shows us a compact and clear summary of the obtained results:

Table 2. Results summary

File and aspect	Metadata	Content	Conclusions
Evidence 1 - Digitally created - Atypical shape - 2 signatures	- Incremental update, non-empty xref table - Large size (1 MB) - Adobe Photoshop - Creation date ≠ Modification date	- 1 page image - 1 mask image without signatures - */PieceInfo*, */Private* dictionaries - No text/attachments - Adobe Photoshop traces	Manipulated PDF file adding the two signatures that are missing in the mask image with Adobe Photoshop software.
Evidence 2 - Scanned document - 2 signatures	- Incremental update, non-empty xref table - Large size (28 MB) - Adobe Photoshop - Creation date ≠ Modification date	- 1 page image 12MB - */PieceInfo*, */Private* dictionaries - No text/attachments - Adobe Photoshop traces	Manipulated PDF file with Adobe Photoshop software.
Evidence 3 - Scanned document - 2 signatures	- Incremental update, non-empty xref table - Large size (46 MB) - Adobe Photoshop - Creation date ≠ Modification date	- 1 page image 12MB - */PieceInfo*, */Private* dictionaries - No text/attachments - Adobe Photoshop traces	Manipulated PDF file with Adobe Photoshop software.
Evidence 4 - Digitally created - 1 signature	- No incremental update - PDFCreator - Creation date ≠ Modification date	- 1 signature image - 1 signature negative - Document text - 2 text boxes - No attachments	Manipulated PDF file with PDFCreator, adding the signature image and text boxes.

Evidence 5 - Digitally created - Atypical shape - 2 signatures	- Incremental update, non-empty xref table - Adobe Photoshop - Creation date ≠ Modification date	- 1 page image - */PieceInfo*, */Private* dictionaries - No text/attachments - Adobe Photoshop traces	Manipulated PDF file with Adobe Photoshop software.
Evidence 6 - Digitally created - Atypical shape - 2 signatures	- Incremental update, non-empty xref table - Adobe Photoshop - Creation date ≠ Modification date	- 1 page image - */PieceInfo*, */Private* dictionaries - No text/attachments - Adobe Photoshop traces	Manipulated PDF file with Adobe Photoshop software.
Evidence 7 - Digitally created - 1 signature	- No incremental update - PDFCreator - Creation date ≠ Modification date	- 1 signature image - 1 signature negative - Document text - 2 text boxes - No attachments	Manipulated PDF file with PDFCreator, adding the signature image and text boxes.
Evidence 8 - Digitally created - 2 Signatures with non-matching colour and background	- No incremental update - PDFCreator - Creation date ≠ Modification date	- 2 different signature images - Document text - No attachments	Converted to PDF and manipulated with PDFCreator, by adding two signatures in the original file as images.
Evidence 9 - Digitally created - No signatures	- Incremental update with empty xref table - Microsoft Word 2013 - Creation date = Modification date	- Document text - No attachments or images	Digitally created with Microsoft Word 2013 and converted afterwards to PDF with the option 'save as PDF'. It is not manipulated

In Table 2, firstly we can see the initial impressions and details that can be observed with a PDF reader tool, and give us a general idea of what we are going to encounter. Next, we can see the information present in metadata section(s), specifying useful fields such as dates and times, or creator and producer tool. Afterwards, the content is address, specifying only the relevant objects and elements that will give us the keys to learn which manipulations have been performed if any. Finally, some brief conclusions are presented with the discoveries and final decisions made about the integrity of the files under investigation. In next section this results will be discussed and evaluated, as well as other aspects of the method apart from the effectiveness.

Evaluation of results

As we can observe from Table 2, the method has been proven quite effective when dealing with typical cases of PDF file manipulation. In order to test if all the manipulations have been actually discovered, a list including details on how they were done (provided by the actual manipulator, Expert Inna Ivask) is included in Appendix IV. Regarding the results of the tests performed, we could evaluate them as follows:

- In *Tests 1, 5 and 6,* the shape of the PDF file page was atypical and they look like they have been digitally created. However, the main difference was the absence of mask image, therefore the manipulations could not be specified. Nevertheless, within the page leaf, /PieceInfo and /Private dictionaries still appeared, which, as stated before, is Adobe Photoshop specific product data and resources. All these elements indicated that these files were manipulated.
- *Tests 2 and 3*: The appearance of these evidences suggested that they were scanned documents (the shadows and edges revealed it). Size was big (28 MB and 46 MB) and this was mainly due to the large size of the page image (11MB), and in Evidence 3 due to an extra object including Adobe Photoshop specifications (which was probably due to differences when exporting the document to PDF). There were no mask images, so the exact manipulations could not be specified. As /PieceInfo and /Private dictionaries, along with traces in metadata from Adobe Photoshop were found, we can say that both PDF files were manipulated, although we cannot specify the exact changes performed.
- *Tests 4 and 7*: these evidences looked as regular digital files converted to PDF, but when examining them with the method we found signatures images (and their negatives) as well as text boxes that were certainly added afterwards. There was no doubt that these files were manipulated.
- *Test 8*: This test resembled a priori to *Tests 4 and 7,* except for Evidence 8 not having text boxes added, and having two signatures instead of one. Moreover, if we take a look at the objects, we find one text object referenced in /Contents, with all the text found in the content extraction with pdftotext utility, and two /XObjects corresponding to the two signature images, leading us to think that they were added afterwards. Therefore, this file was manipulated.
- *Test 9*: this evidence presented all the elements to correspond to a ‘saved as PDF’ non-manipulated file, and there were no suspicious traces whatsoever. In consequence, we conclude that this file was not manipulated.

Therefore, we can say that even when in some of the files (*Tests 2,3,5* and *6*) it was not possible to determine the exact manipulations made to the documents, we were still able to find traces of editing software. Moreover, the method has been proven capable of also detecting files which have not been manipulated, as shown in *Test 9*.

Also, as we have commented in the overview, we lack evidence context; in a normal court case, some information about the file is provided, like ownership, timestamps and nature of the document. In this validation process, we have made hypothesis about this information, and drawn conclusions that would be verifiable with the context we are missing.

Another important aspect to highlight, is the fact that in most of the tests, a simple look at the metadata would have been enough to suspect and sometimes even to determine that the file was manipulated. However, and as shown in previous sections of this document (*4.2 Metadata manipulation*) metadata can be deleted and/or modified, so it is not a hundred per cent trustable. But it has been shown that this does not suppose a real problem to our method, as metadata, even though it is a pretty accurate and useful source of information, is not the only place where we find traces of manipulation. As shown in section *4. Manipulation experiments,* changing even the slightest detail in a PDF document content is rather hard, you need to completely modify the source code and even though it is unlikely that you obtain a readable PDF file, not even mentioning the deep knowledge about PDF file structure it is needed to do so. This makes forgers to resort to editing software, and here we obtain the greatest head-start, as all software leaves a considerable number of traces.

In order to complete this validations process, and once the effectiveness of the method has been shown, some other aspects should be addressed, such as:

- *Performance*: although this method has been proven effective against certain delimited types of PDF files, with specific characteristics in terms of length or types of elements within the file, it has not been tested against more long, complex and convoluted PDF files. Therefore, the performance of this method applied to the thread model described in *3.4 Scope and limitations* turns out as satisfactory, but it remains uncertain outside this kind of PDF files and manipulations.
- *Correctness*: it is rather hard to assess this aspect of the validation process, as this method does not answer to any specific standard against which to evaluate it. Nevertheless, it has been constructed keeping in mind the examination process carried out by experts, and until formal use has been made of the method, it remains uncertain.
- *Usability*: this method consists on a set of layouts against which to compare the targeted files under investigation, and focus on the differences to prove their integrity. Even if the layouts are understandable and well presented, it is not the most convenient way to look for suspicious details, as it needs to be done manually and takes some effort. For this, and in future works, an automated software could be produced to easy this process.

7 Conclusions

Portable Document Format being the most used one when it comes to storing and communicating files involves that our best efforts must be done to ensure their integrity and to be able to detect manipulations and forgeries when performed against them. Even though the best solution to this issue would be the generalised used of digital signatures in PDF files, this practice has not been yet generally adopted, and until this objective is achieved, we need to figure out how to deal with digital evidence whose integrity has not been ensured.

This study on PDF anatomy and structure has confirmed the intricacy of the construction of these files, the difficulty of investigating them and their components, and most certainly the arduousness of making any manual change without ruining its composition. As using software to manipulate PDF files for fair purposes is totally lawful, the fact that traces are left behind in the source code of the file must not be a reason of concern. For our regards, this supposes an advantage when looking for manipulation traces, and if you combine this with the complexity of manual manipulations (which are normally the undetectable ones) we can conclude that a high percentage of the PDF file manipulations are detectable.

Regarding those manual manipulations, some of them have been performed in this work; even when some have resulted in minimum changes, like the limitations present when trying to manually manipulate text, metadata has been proved malleable and images replaceable. And what is more important, these forgeries, although proved to be harsh and tough for the little result achieved sometimes, have resulted undetectable, as no modifications at image level have been performed (invisible for image processing or text-line techniques) and all the traces have been covered in the source code (imperceptible for manual investigation). Next steps would be directed towards the study of how to tackle these types of manipulations and how to prevent them from happening, as well as the combination of existing image processing or / and text-line techniques and manual investigations procedures.

The result method of this thesis is only a first step towards the absolute effective detection of manipulation in PDF files, as it has been mainly directed towards the most common forgeries performed to relatively simple documents (low number of pages and no media apart from images included), by individuals who do not necessarily possess deep technical knowledge and skills. Larger PDF files including a higher variety of elements (such as forms or media) can be tested for manipulation and subsequent detection in following studies.

This method has been proven effective in finding traces of manipulating software in PDF files source code, residing in added objects that stand out from the ones normally present in a determined type of PDF file; all this has been successfully gathered in the constructed layouts against which targeted files can be compared. However only 3 layouts have been created, as a generalisation of the types of PDF files that examiners encounter (scanned with and without OCR and converted to PDF), so PDF files responding to other formats might be more difficult to prove manipulated. For this, a deeper study in different types of PDF files must be carried out, creating more layouts or even improving and perfecting the ones presented in this thesis.

Just like the construction of the method, the validation phase has been built with the objective of proving the success of detecting typically manipulated files with most used software. Even though a validation process must push the method to its limits, in the case of this thesis those limits were so broad that they were considered out of scope for the aim of this work; EKEI provided me with PDF files containing most usual manipulations they encounter, which was decided to be more practical than trying to perform outrageous forgeries almost nobody would perform. However, non-typical manipulations executed with different software, and different strategies have not been tested yet against the method, and they would help to improve it and find flaws it may have when addressing this other kind of forgeries. Moreover, we can observe some weaknesses in the method regarding its correctness and usability, which can be addressed in further studies and work in order to improve this method, making it more convenient in terms of use and automation and more appropriate and professional.

To conclude, the goal of this work is to ease, organise and normalise the process of detecting manipulations looking at the source code of the PDF file, before having to use other more tedious and inconvenient methods such as image processing or text-line examination. It is intended to be a previous step to tackle that can save a lot of time to examiners and provide them with more proof of manipulations regarding digital file evidence. When applied in this manner, the result method performs its aimed duty, representing a very important starting point in manual investigation for manipulation detection in Portable Document Format files.

8 Future work

As repeatedly pointed out during this thesis, this study and result method is just a starting point in this long way of PDF manipulation detection. Opening a window on this matter has caused more lines of investigation to arise, that will lead to a solid and complete way on how to detect any kind of manipulation in any kind of PDF file. Some of these lines of work could be:

- *Non-typical manipulations executed with different software*: in this study, only typical forgeries using most common editing software (such as Adobe Photoshop or PDFCreator) have been tackled. Other forgeries that are not so common but might become usual in the future can be tested against the method to find flaws and improve it, or even create a different one.
- *More variety of PDF types and elements*: as already mentioned, this study only addresses simple single-page PDF files with just text and images. There are more kinds of PDF files that could be tackled, as shown in the Background section *Types of PDF files,* as well as many more elements that can be present, such as forms, multimedia or attached files. A further study taking all this elements into account would broad the reach of the method.
- *Detection of manual manipulations*: in this document, manual manipulations have been performed as a method of studying the vulnerabilities of this format, but their detection has been left out of the picture. Addressing the detection of manipulations made directly to the source code, instead of the ones made using editing software that leaves traces will suppose a huge contribution to fill out this method.
- *Combination of detection techniques*: one step further in this field would suppose the combination of existing image processing and text-line examination techniques and manual investigations procedures, which will lead towards an even more complete and efficient method to use in order to cover all aspects in PDF file manipulation.
- *Untraceable manipulation software:* the manual manipulations performed in this work were for now untraceable because they were made in the source code. A valuable contribution would be the creation of an editing tool that would perform this manipulations in the source code, so there would be no need to perform them manually. Even if this seems detrimental for our investigation, when used to get a better knowledge of the format and to pose new challenges in this field it would suppose a great contribution in this matter.
- *Automatised investigation software:* even when the layouts presented in this thesis perform their duty, they are not the most convenient way to investigate files, as almost everything has to be done manually, using a variety of tools to combine all the capabilities needed for the investigation. A creation of a software capable of performing such actions altogether would be a great contribution to this field, and would help, easy speed up this process considerably.

9 References

[1] T. Bradley, "PDF Files Most Trusted...and Most Targeted," March 2011. Available: http://www.pcworld.com/article/221612/pdf_files_most_trusted_and_most_targeted.html. [Last accessed: January 2017].

[2] L. M. Hughes, "Why digitize? The costs and benefits of digitization," *Digitizing Collections: Strategic issues for the information manager*, Facet Publishing, 2003, pp. 3-30.

[3] Managed Outsourced Solutions Blog, "Advantages of Digitization," 2007. Available: http://www.managedoutsource.com/blog/2007/10/advantages-of-digitization.html). [Last accessed: February 2017].

[4] World Economic Forum, "The Global Information Technology Report 2012: Living in a Hyperconnected World," World Economic Forum, 2012.

[5] DocuSign Inc., "What are digital signatures?," 2017. Available: https://www.docusign.com/how-it-works/electronic-signature/digital-signature/digital-signature-faq. [Last accessed: March 2017].

[6] Scientific Working Group on Digital Evidence, "SWGDE Best Practices for Computer Forensics," Scientific Working Group on Digital Evidence, 2014.

[7] J. Ashcroft, D. J. Daniels and S. V. Hart, "Forensic Examination of Digital Evidence: A Guide for Law Enforcement," U.S. Department of Justice, Office of Justice Programs, National Institute of Justice , 2004.

[8] S. Mason, "International Electronic Evidence," British Institute of International and Comparative Law, 2008.

[9] Adobe Systems Incorporated, "PDF. Three letters that changed the world," 2017. Available: https://acrobat.adobe.com/ee/en/why-adobe/about-adobe-pdf.html. [Last accessed: January 2017].

[10] J. Warnock, "The Camelot Project," September 2013. Available: https://blogs.adobe.com/acrobat/files/2013/09/Camelot.pdf. [Last accessed: January 2017].

[11] L. Leurs, "The history of PDF," November 2016. Available: https://www.prepressure.com/pdf/basics/history. [Last accessed: January 2017].

[12] T. Willis, "8 Types of Standards - Each serves a unique purpose," December 2013. Available: http://blog.marconet.com/blog/bid/326753/8-Types-of-PDF-Standards-Each-Serves-a-Unique-Purpose. [Last accessed: February 2017].

[13] Adobe Systems Incorporated, "Document management — Portable document format — Part 1: PDF 1.7". Patent ISO 32000:2008, 1 July 2008.

[14] Adobe Systems Incorporated, "Extensible Metadata Platform (XMP) Overview Partners," 2017. Available: http://www.adobe.com/products/xmp.html. [Last accessed: 2017].

[15] Adobe Systems Incorporated, "XMP Specification Part 3: Storage in Files," 2010.

[16] Adobe Systems Incorporated, "XMP Specification Part 1: Data Model, Serialization and Core Properties," 2012.

[17] K. Selvaraj and N. F. Gutierrez, "The Rise of PDF Malware," Symantec Corporation, 2010.

[18] D. Zimmer, "PDF Stream Dumper," July 2010. Available: http://sandsprite.com/blogs/index.php?uid=7&pid=57. [Last accessed: March 2017].

[19] iText Group NV, "iText," 2017. Available: http://itextpdf.com. [Last accessed: March 2017].

[20] J. Berkenbilt, "QPDF Manual For QPDF Version 6.0.0," SourceForge, 2015.

[21] J. M. Esparza, "Peepdf - PDF Analysis tool," 2016. Available: http://eternal-todo.com/tools/peepdf-pdf-analysis-tool. [Last accessed: March 2017].

[22] D. Noonburg, "Xpdf: A PDF viewer for X," Available: http://www.foolabs.com/xpdf/home.html. [Last accessed: March 2017].

[23] N. Khannaa, G. T. C. Chiub, J. P. Allebacha and E. J. Delp, "Scanner Identification with Extension to Forgery Detection," SPIE, 2008.

[24] C.-H. Choi, M.-J. Lee, and H.-K. Lee, "Scanner Identification using Spectral Noise in the Frequency Domain," IEEE, 2010.

[25] S. A. Zeinab F. Elsharkawy, S. M. E. Abdelwahab and F. E. A. E.-S. Moawad I. Dessouky, "Accurate and Robust Identifying Forged Region Method in Scanned Images," International Journal of Computer Applications, 2013.

[26] R. M. Abed, "Scanned Documents Forgery Detection Based on Source Scanner Identification," American Journal of Information Science and Computer Engineering, 2015.

[27] S. Shang, N. Memon and X. Kong, "Detecting documents forged by printing and copying," EURASIP Journal on Advances in Signal Processing, 2014.

[28] K. Saini and S. Kaur, "Forensic examination of computer-manipulated documents using image processing techniques," Egyptian Journal of Forensic Sciences, 2015.

[29] S. Abramova and R. Böhme, "Detecting Copy–Move Forgeries in Scanned Text Documents," Society for Imaging Science and Technology, 2016.

[30] F. H. Naser Hasan, "New Method to Detect Text Fabrication in Scanned Documents," Islamic University in Gaza, 2015.

[31] J. Van Beusekom, F. Shafait and T. M. Breuel, "Text-line examination for document forgery detection," Springer-Verlag, 2012.

[32] IDR Solutions, "Make your own PDF file – Part 6: Graphics State," November 2010. Available: https://blog.idrsolutions.com/2010/11/make-your-own-pdf-file-—-part-6-graphics-state/. [Last accessed: March 2017].

Appendix

I. PDF version comparison

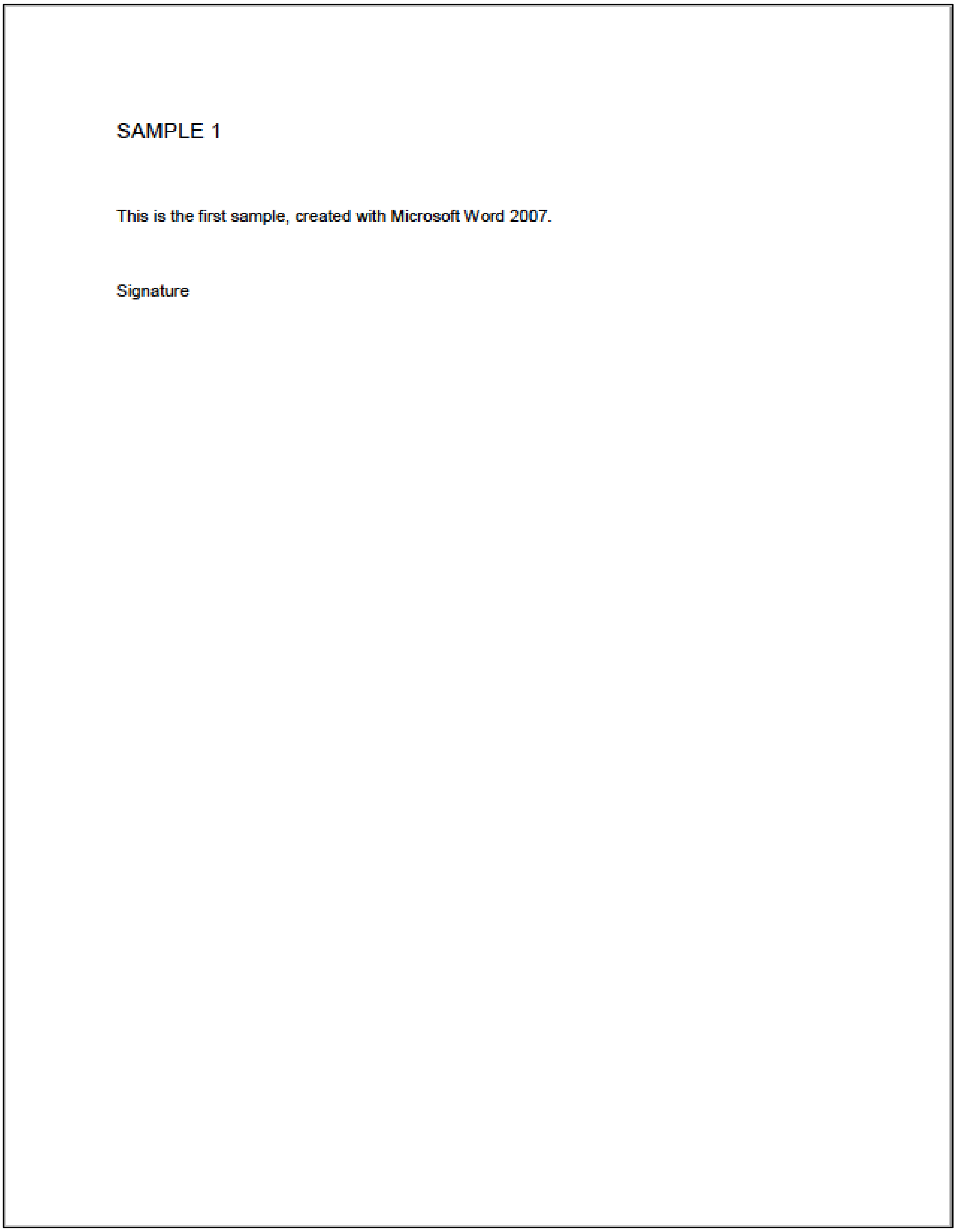
Table 3. PDF version comparison

Version	Availability and Matching Software	Added Features
PDF 1.0	November 1993, Adobe Acrobat 1.0	-
PDF 1.1	November 1994, Adobe Acrobat 2.0	Embedded links Article threads Security features (password protection) Device independent colour Notes
PDF 1.2	November 1996, Adobe Acrobat 3.0	Electronic form. Extended character sets. Multimedia features OPI 1.3 specifications (Online Product Information) Improved colour support. Embedded Halftone functions and overprint instructions
PDF 1.3	April 1999, Adobe Acrobat 4.0	2-byte CID (Character Identifier) fonts Support for OPI 2.0 Additional colour spaces Smooth shading Annotations Digital signatures JavaScript actions RC4 encryption (Rivest Cipher 4)
PDF 1.4	May 2001, Adobe Acrobat 5.0	Transparency Improved support for JavaScript Support for Tagged PDF JBIG2 compression 128-bit RC4 encryption

Version	Availability and Matching Software	Added Features
PDF 1.5	April 2003, Adobe Acrobat 6.0 & Acrobat Reader 6.0	Improved compression techniques (object streams & JPEG 2000 compression) Enhanced XRef table (XRef streams) Support for layers Improved support for tagged PDF XFA (XML Forms Architecture) Additional transitions (PDF presentations)
PDF 1.6	January 2005, Adobe Acrobat 7.0 & Adobe Reader 7.0	NChannel (defining spot colours) AES encryption (Advanced Encryption Standard) Enhancements to annotations and tagging Direct embedding of OpenType fonts Embedded files into PDF. Embedded 3D data - U3D (Universal 3D) XML forms
PDF 1.7	October 2006, Adobe Acrobat 8.0 & Adobe Reader 8.0	Improved support for commenting & security. Comments to 3D-objects More control over 3D animations. Embed default printer settings
	2008, Adobe Acrobat 9.0 & Adobe Reader 9.0	256-bit AES encryption
	2009, Adobe Acrobat 9.1 & Adobe Reader 9.1	XFA 3.0

II. Sample Documents

- First digitally created file by means of Microsoft Word 2007, converted to PDF with text recognition by Microsoft Print to PDF utility.



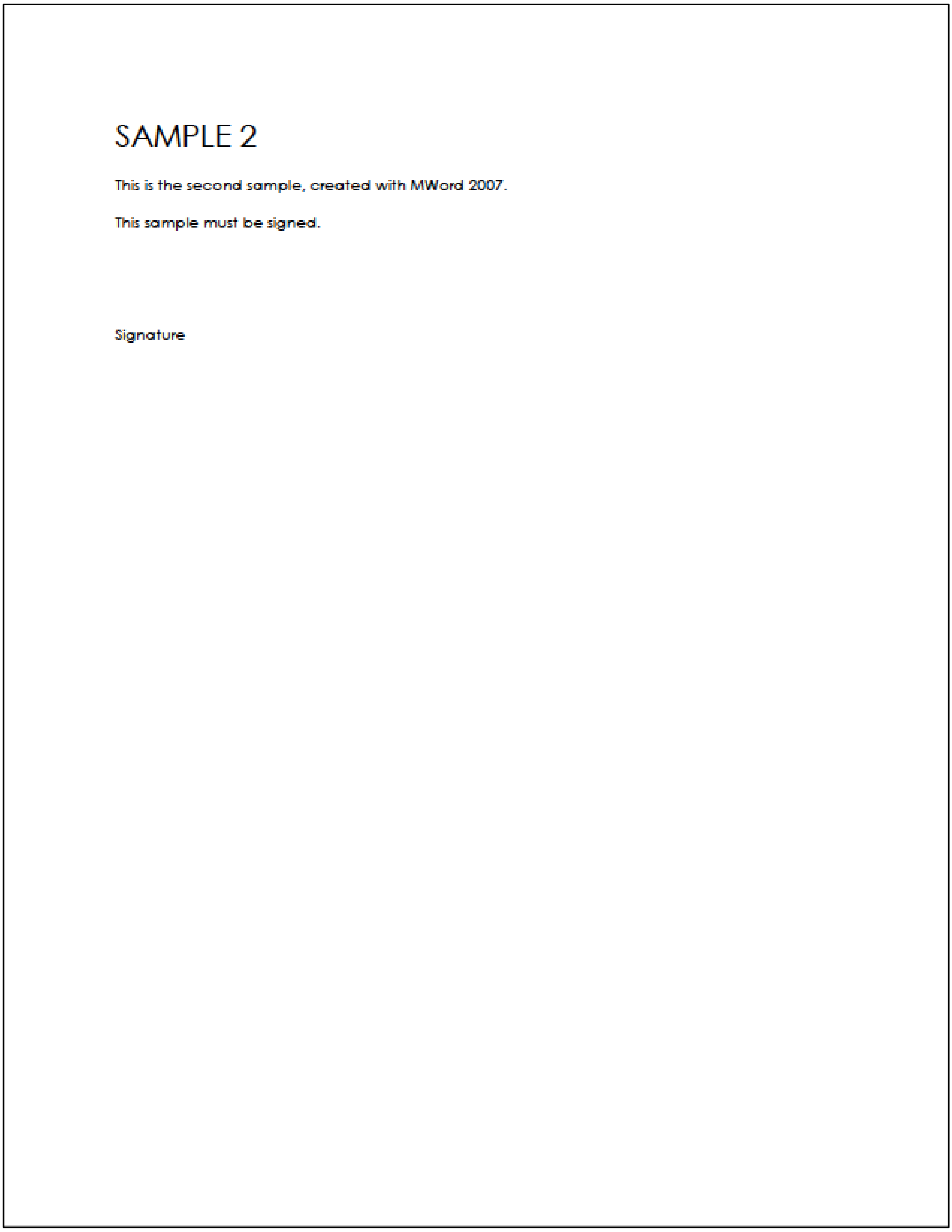
SAMPLE 1

This is the first sample, created with Microsoft Word 2007.

Signature

Figure 43. Sample1.pdf

- Second digitally created file by means of Microsoft Word 2007, converted to PDF with text recognition by Microsoft Print to PDF utility.



SAMPLE 2

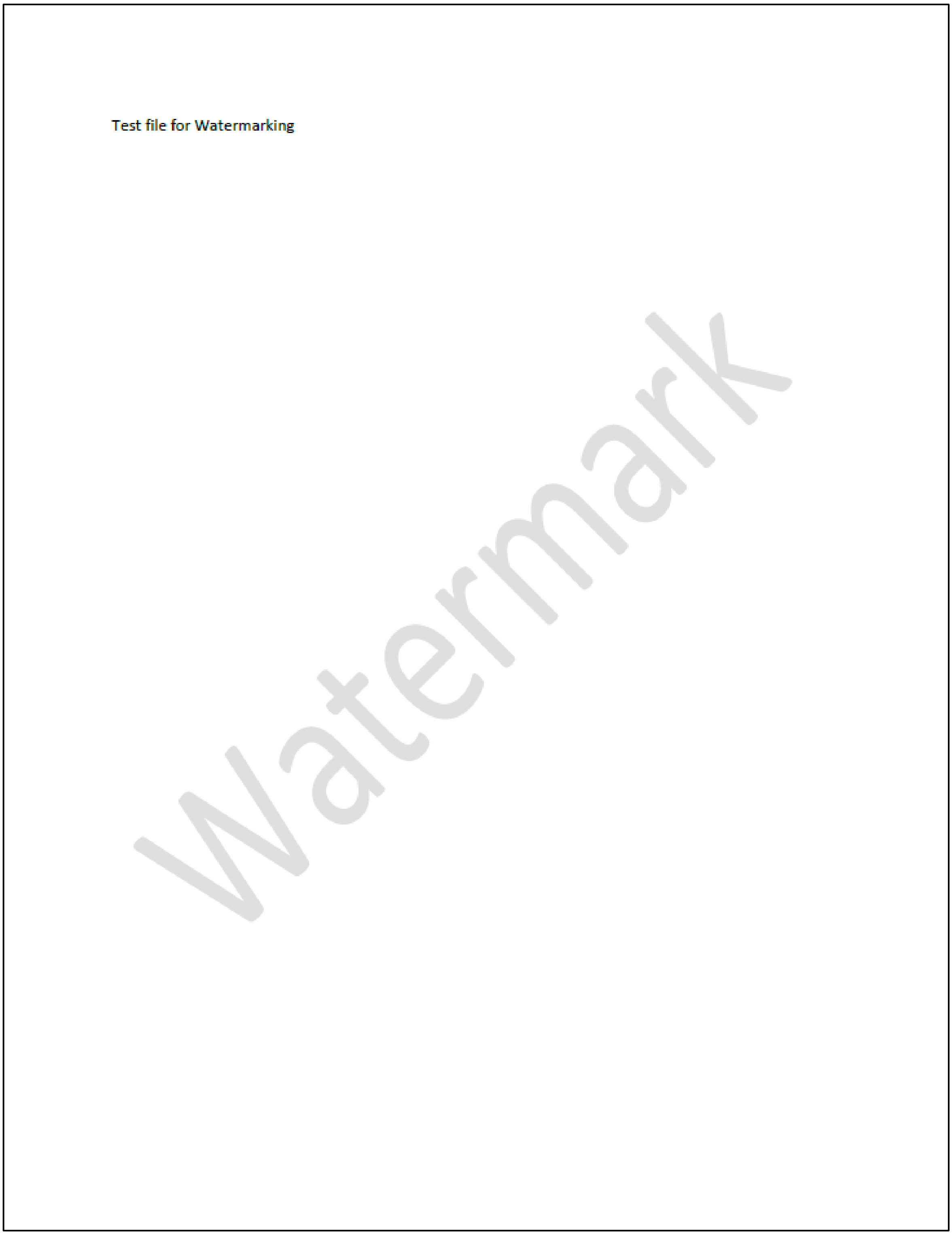
This is the second sample, created with MWord 2007.

This sample must be signed.

Signature

Figure 44. Sample2.pdf

- Third digitally created file by means of Microsoft Word 2007, including a small text and a watermark; converted afterwards to PDF with text recognition by Microsoft Print to PDF utility.

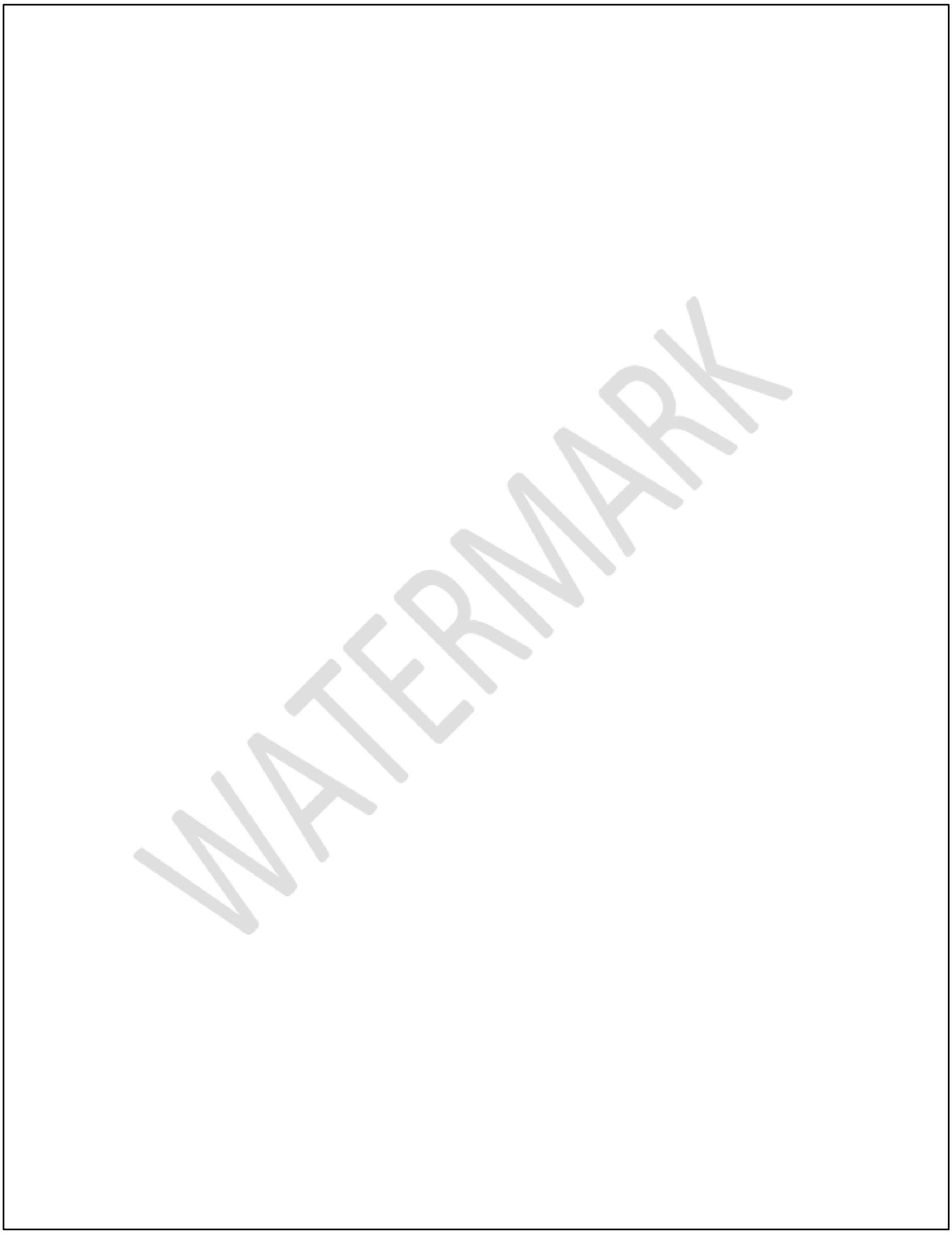


Test file for Watermarking

Watermark

Figure 45. Watermark_text.pdf

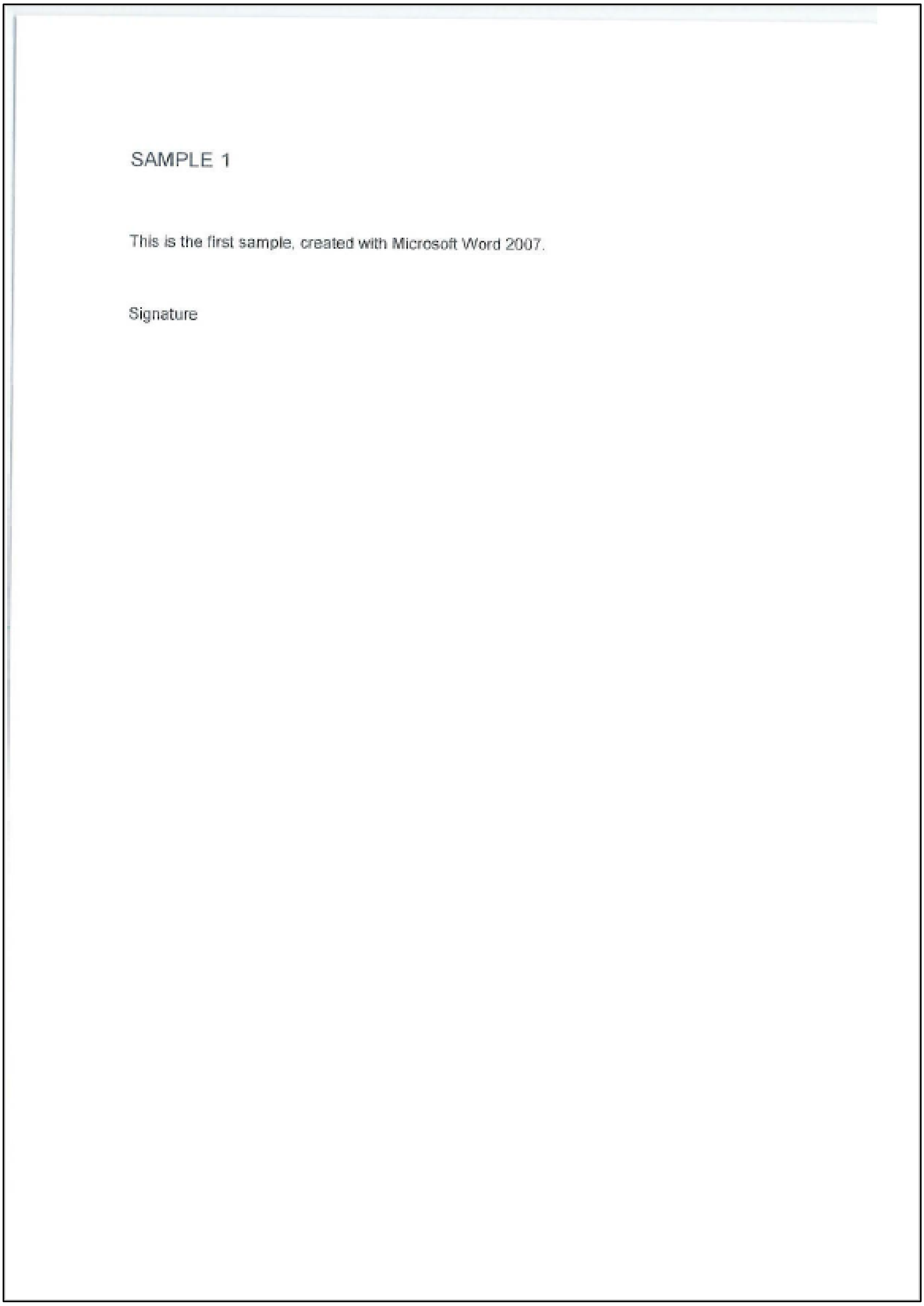
- Forth digitally created file by means of Microsoft Word 2007, including only a watermark; converted afterwards to PDF with text recognition by Microsoft Print to PDF utility.



WATERMARK

Figure 46. Watermark_only.pdf

- First scanned document without OCR; the document Sample1.pdf was printed and scanned afterwards.



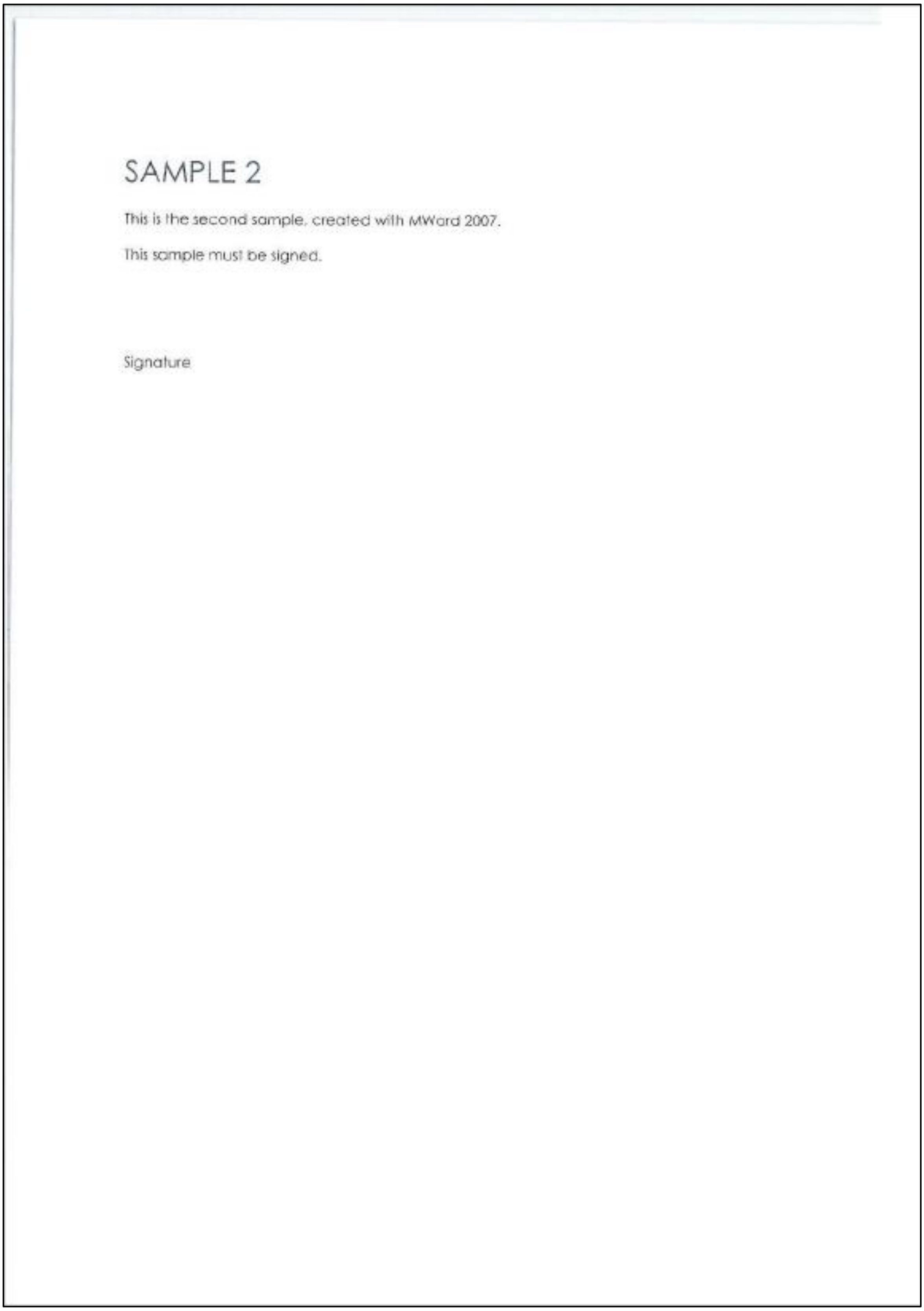
SAMPLE 1

This is the first sample, created with Microsoft Word 2007.

Signature

Figure 47. sc_Sample1.pdf

- Second scanned document without OCR; the document Sample2.pdf was printed and scanned afterwards



SAMPLE 2

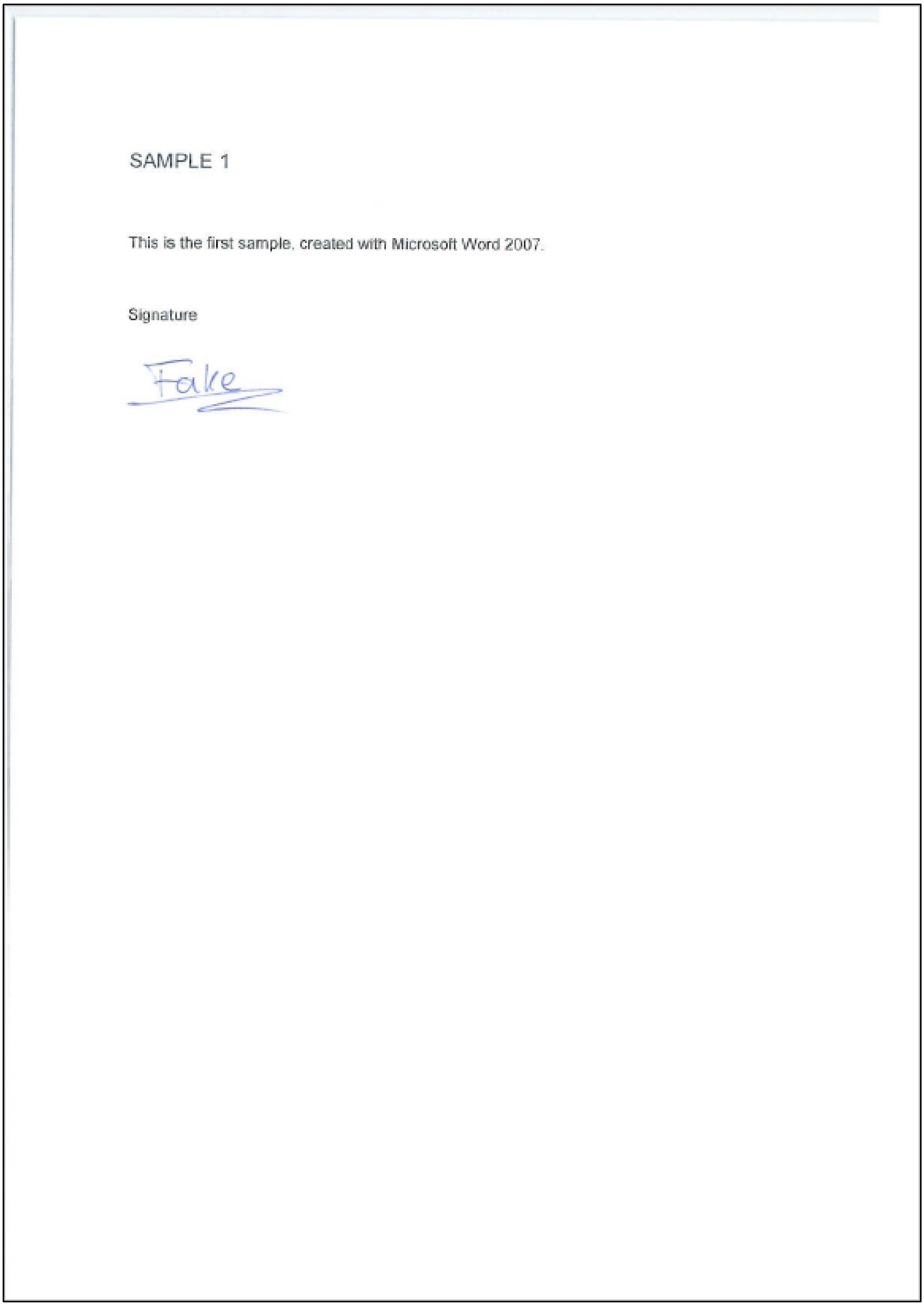
This is the second sample, created with MWord 2007.

This sample must be signed.

Signature

Figure 48. sc_Sample2.pdf

- Third scanned document without OCR; the document Sample1.pdf was printed, manually signed with a fake handwritten signature, and scanned afterwards.



SAMPLE 1

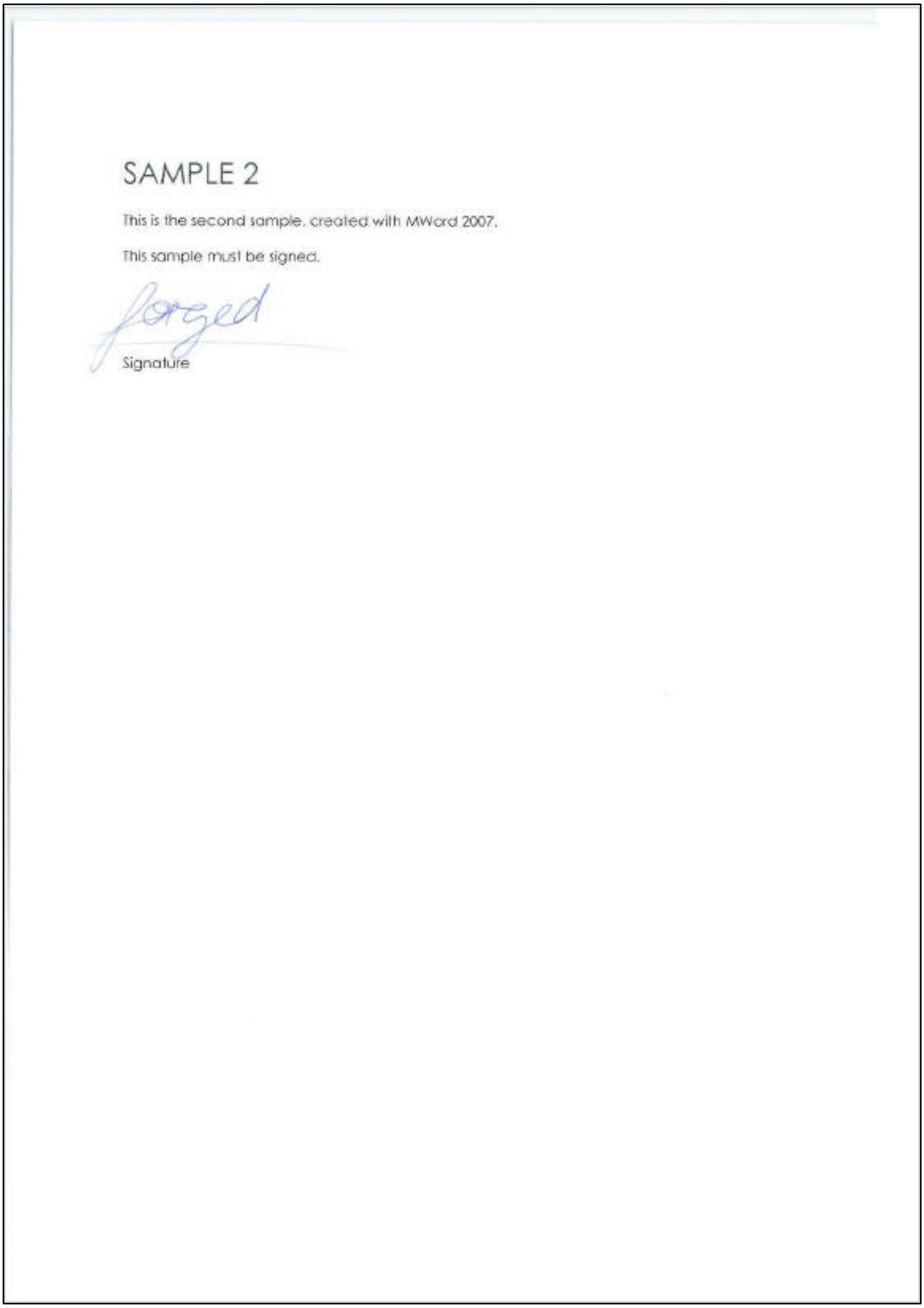
This is the first sample, created with Microsoft Word 2007.

Signature

Fake

Figure 49. sig_sc_Sample1.pdf

- Forth scanned document without OCR; the document Sample2.pdf was printed, manually signed with a fake handwritten signature, and scanned afterwards.



SAMPLE 2

This is the second sample, created with MWord 2007.

This sample must be signed.

forged

Signature

Figure 50. sig_sc_Sample2.pdf

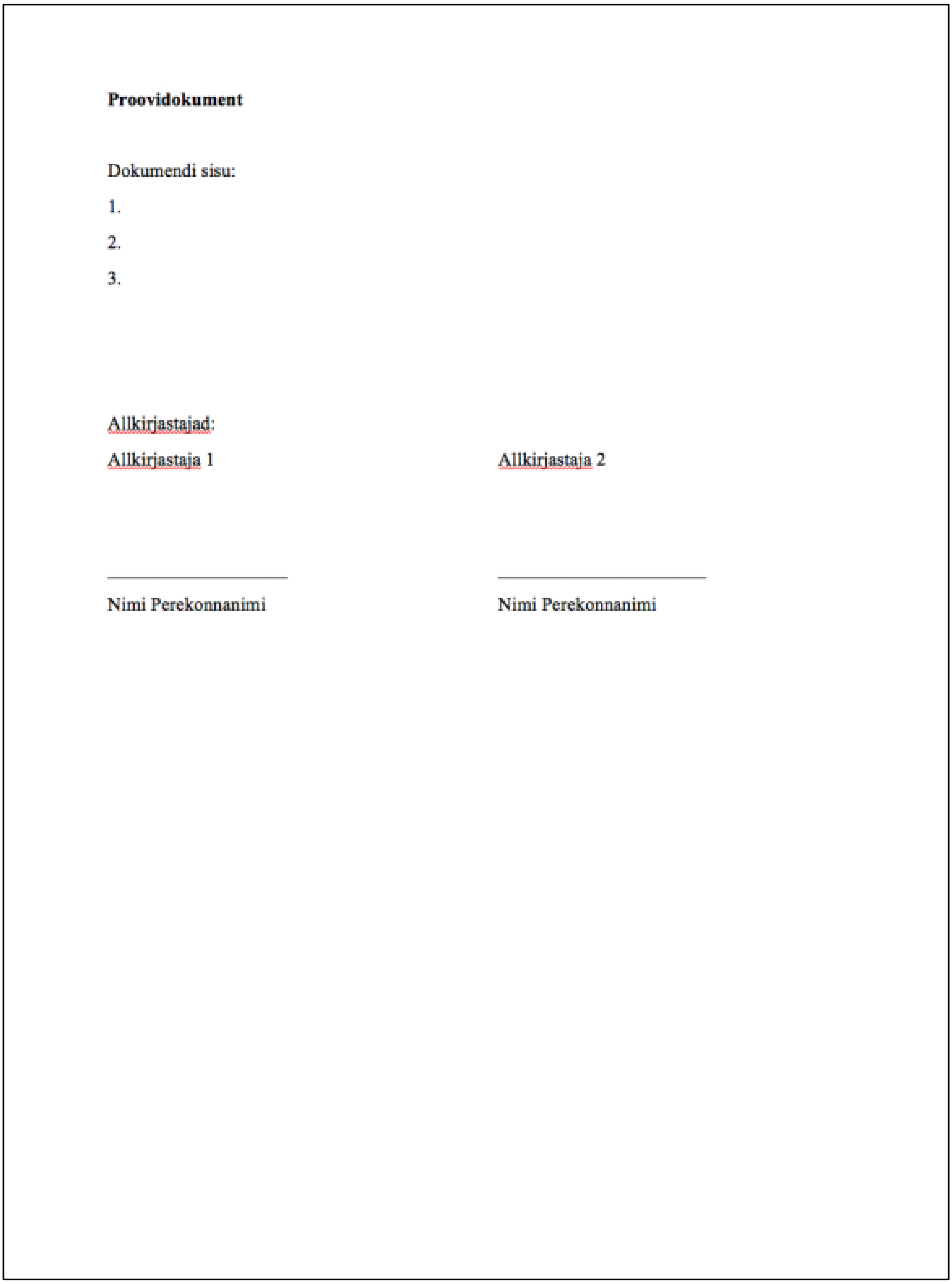
III. Validation file name mapping

Table 4. Files name mapping

Test & Evidence Number	Original name
1	17.03.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf
2	17.03.2017_11_51_24_55499_DocskanRGB8_plus_2_skanRGB8allkiri_flatten.pdf
3	17.03.2017_11_51_24_88495_DocskanRGB8_plus_2_skanRGB8allkiri_2kihti.pdf
4	AdobeReader_PDFdoc_plus_allkiri_plus_tekst.pdf
5	PDF_dokument_plus_2_pdfallkirja_flatten.pdf
6	Pdf_dokument_plus_2_RGB8skannallkirja.pdf
7	PDFcreatoris_PDFdoc_plus_allkiri_plus_tekst.pdf
8	Word_plus_2_korda_jpgprinterallkiri.pdf
9	Word_dokument.pdf

IV. Validation files manipulation details

For the manipulations, a file created by Microsoft Word software was used, shown in Figure 51; it was also printed, then signed and scanned again using *HP Colour Laser Jet CM6040MFP* printer, and saving it afterwards as a JPG image (Figure 52). Both documents were then converted to PDF.



Proovidokument

Dokumendi sisu:

1.

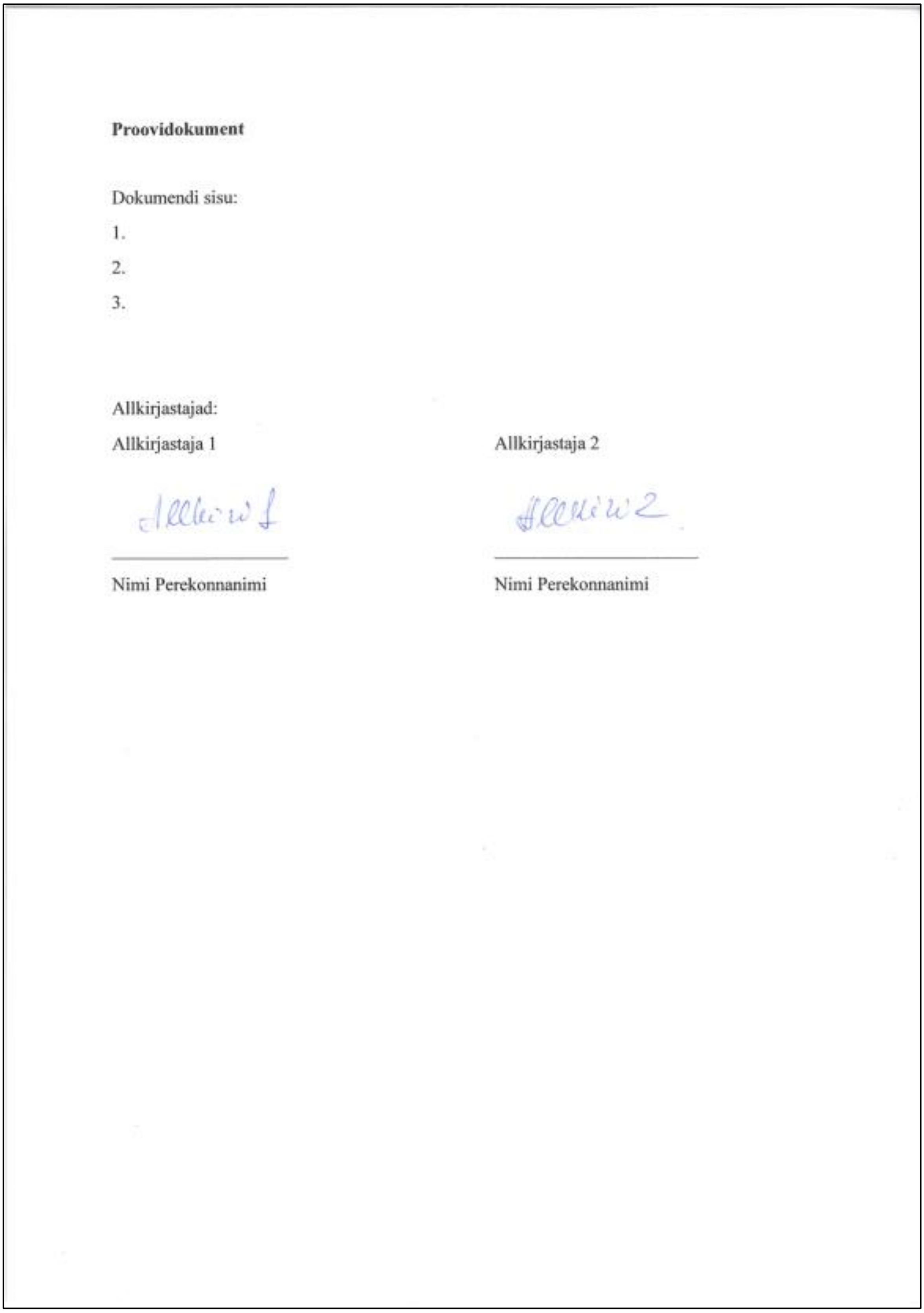
2.

3.

Allkirjastajad:

Allkirjastaja 1	Allkirjastaja 2
___________________	______________________
Nimi Perekonnanimi	Nimi Perekonnanimi

Figure 51. ‘Word_allkirjadeta.docx’ file



Proovidokument

Dokumendi sisu:

1.

2.

3.

Allkirjastajad:

Allkirjastaja 1

Allkiri 1

____

Nimi Perekonnanimi

Allkirjastaja 2

Allkiri 2

____

Nimi Perekonnanimi

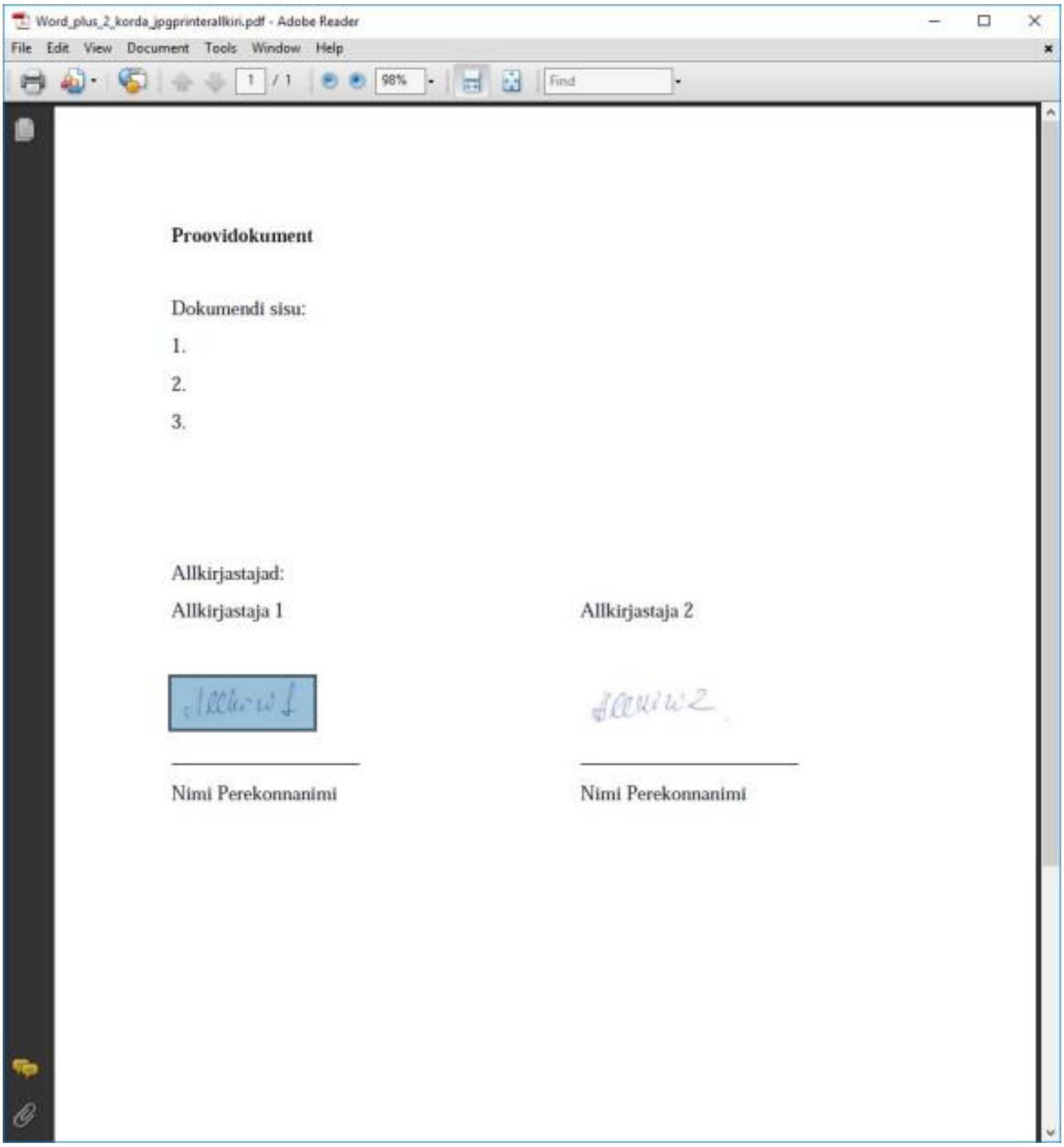
Figure 52. 'allkirjad JPG.jpg' file

Therefore, we have 4 files that will be the base for the manipulations:

1. Word_allkirjadeta.docx: created by Microsoft Word.
2. Word_dokument.pdf: *Word_allkirjadeta.docx* converted to PDF.
3. allkirjad JPG.jpg: scanned signed document and saved as JPG.
4. allkirjad.pdf: *allkirjad JPG.jpg* converted to PDF.

From them, the following manipulations were performed (shown in chronological order):

For *Evidence 8*, shown in Figure 53, the files 'Word_allkirjadeta.docx' and 'allkirjad JPG.jpg' were opened, and from the latter, the two signatures were cropped and inserted in the '.docx' file as images by means of Microsoft Word software. It was then saved to PDF.


Word_plus_2_korda_jpgprinterallkiri.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
98%
Find

Proovidokument

Dokumendi sisu:

1.

2.

3.

Allkirjastajad:

Allkirjastaja 1

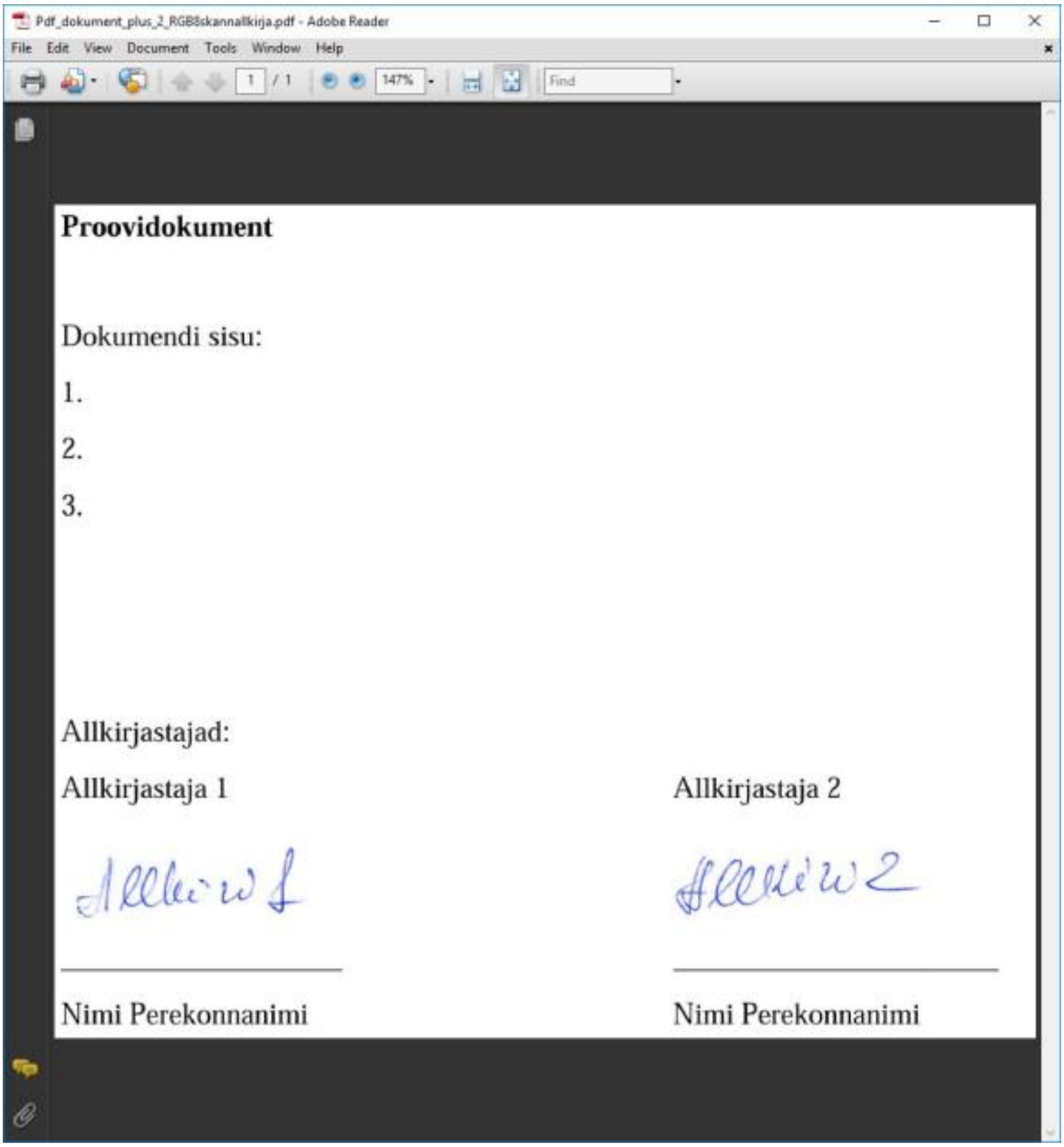
Nimi Perekonnanimi

Allkirjastaja 2

Nimi Perekonnanimi

Figure 53. Evidence 8 aspect, with one of the images selected

For *Evidence 6*, shown in Figure 54, the files ‘Word_dokument.pdf’, and ‘allkirjad JPG.jpg’ were opened in Adobe Photoshop CS2 software. From the latter, two images were cropped and saved as ‘allkiri1.jpg’ and ‘allkiri2.jpg’. Then, they were included as layers in the PDF file. It was then saved to PDF.



Proovidokument

Dokumendi sisu:

1.

2.

3.

Allkirjastajad:

Allkirjastaja 1

Allkiri 1

Nimi Perekonnanimi

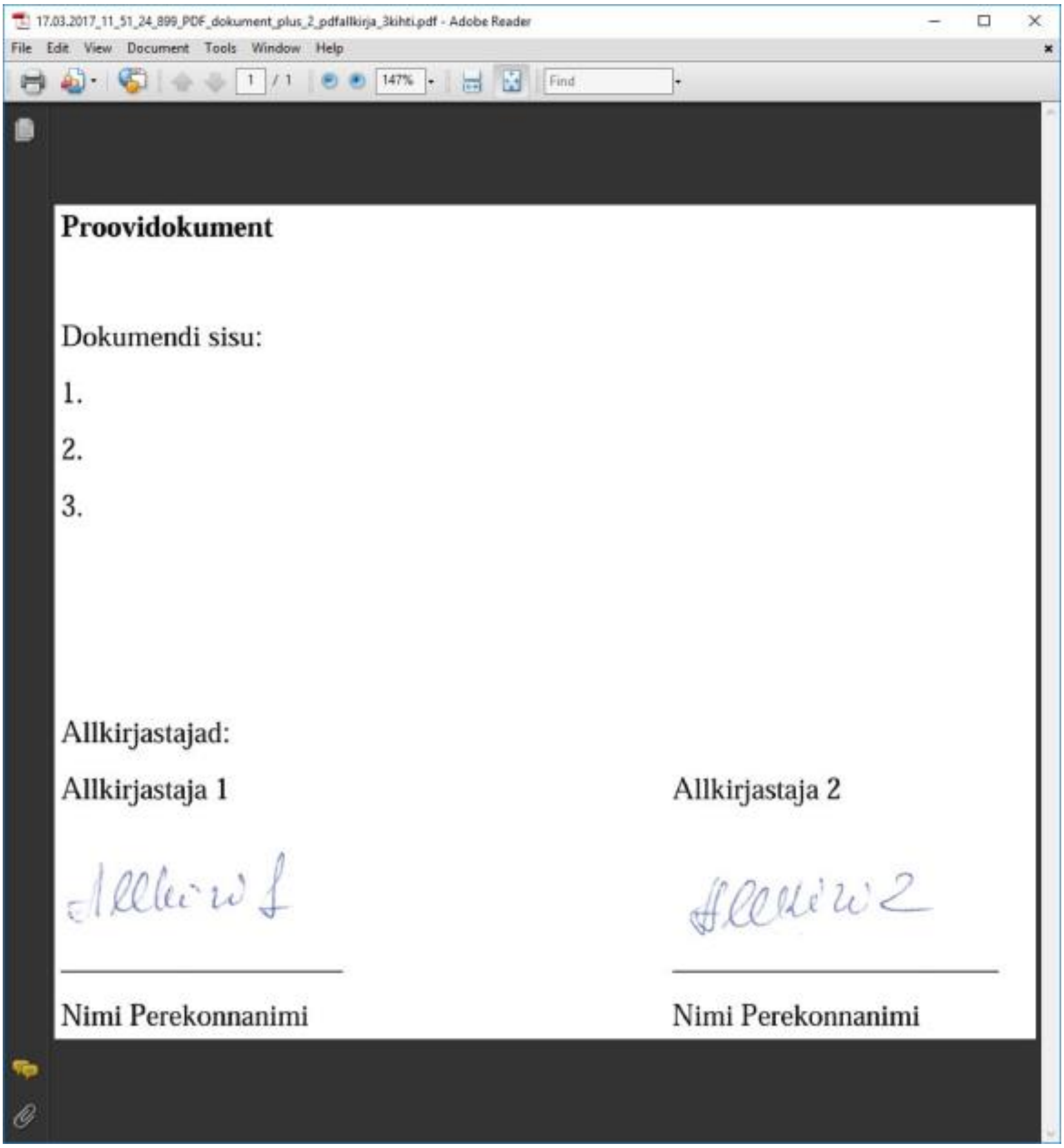
Allkirjastaja 2

Allkiri 2

Nimi Perekonnanimi

Figure 54. Evidence 6 shown by Adobe Acrobat Reader

For *Evidence 1*, shown in Figure 55, ‘Word_dokument.pdf’ and ‘allkirjad.pdf’ were opened in Adobe Photoshop CS2 software. The two signatures were cropped from ‘allkirjad.pdf’ and pasted on ‘Word_dokument.pdf’ as two different layers. Finally, it was saved as PDF file.


17.03.2017_11_51_24_899_PDF_dokument_plus_2_pdfallkirja_3kihti.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
147%
Find

Proovidokument

Dokumendi sisu:

1.

2.

3.

Allkirjastajad:

Allkirjastaja 1

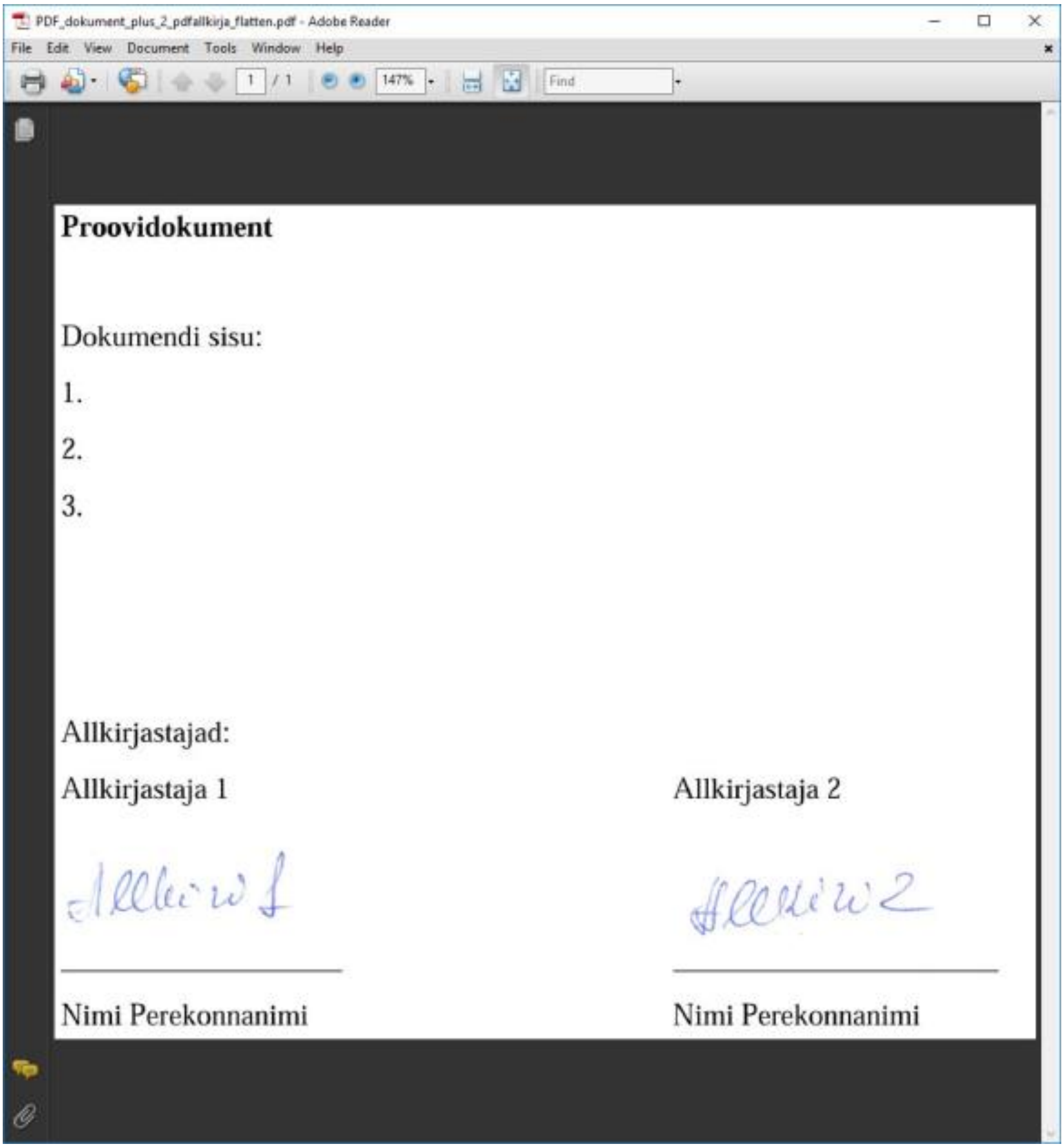
Allkirjastaja 2

Nimi Perekonnanimi

Nimi Perekonnanimi

Figure 55. Evidence 1 shown by Adobe Acrobat Reader

For *Evidence 5*, shown in Figure 56, the files ‘Word_dokument.pdf’ and ‘allkirjad.pdf’ were opened in Adobe Photoshop CS2 software. The two signatures were cropped from ‘allkirjad.pdf’ and pasted on ‘Word_dokument.pdf’ as layers, but saved it with ‘Flattened image’ option. Finally, it was saved as PDF file.


PDF_dokument_plus_2_pdfallkirja_flatten.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
147%
Find

Proovidokument

Dokumendi sisu:

1.

2.

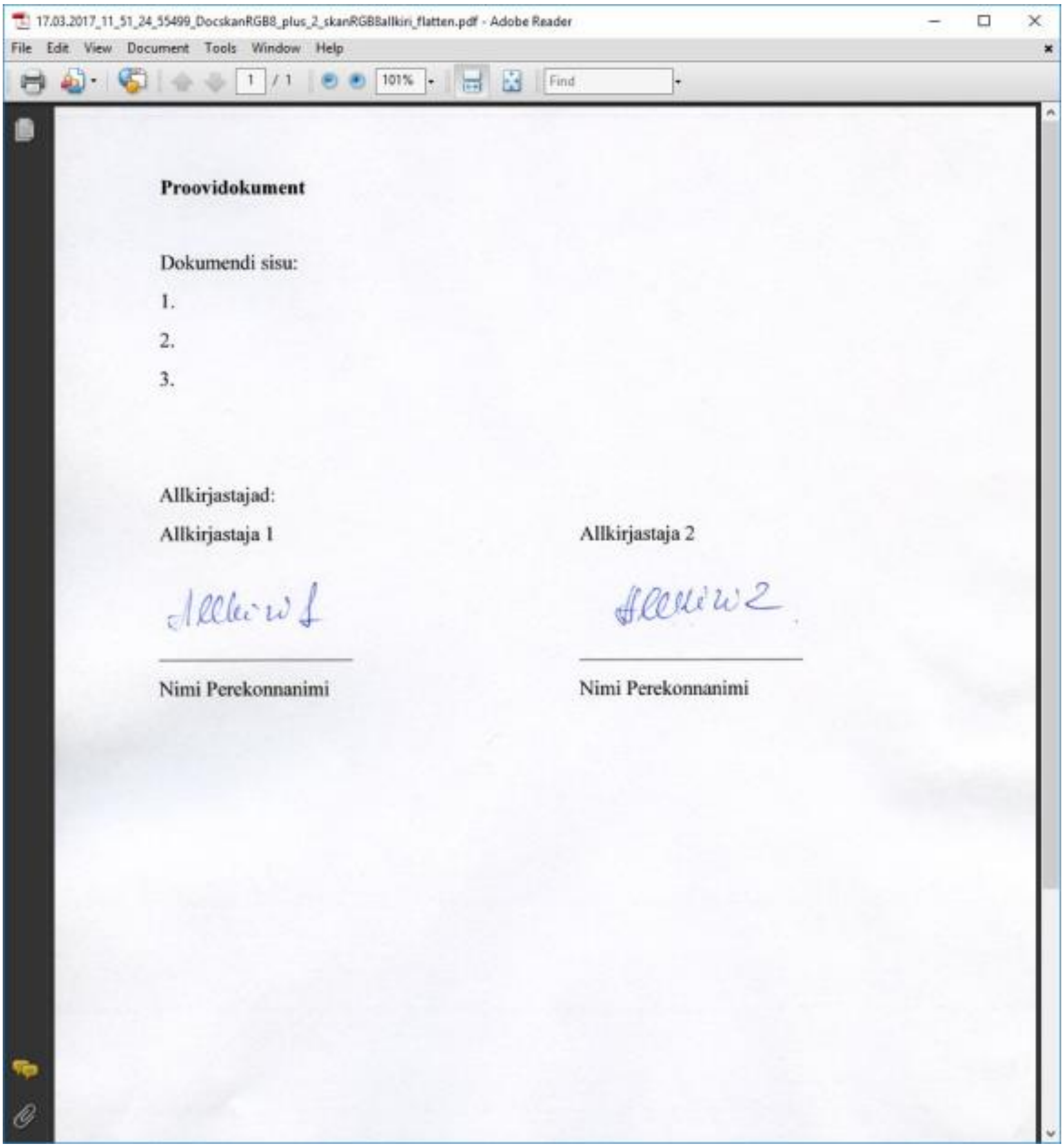
3.

Allkirjastajad:

Allkirjastaja 1	Allkirjastaja 2
Allkiri 1	Allkiri 2
Nimi Perekonnanimi	Nimi Perekonnanimi

Figure 56. Evidence 5 shown by Adobe Acrobat Reader

For *Evidence 3*, shown in Figure 57, ‘allkirjad.pdf’ and ‘allkirjad jpg.jpg’ were opened in Adobe Photoshop CS2 software, and the signatures from the latter were cropped and pasted in the former as two different layers. Finally, it was saved as a PDF file.


17.03.2017_11_51_24_55499_DocskanRGB8_plus_2_skanRGB8allkiri_flatten.pdf - Adobe Reader
File
Edit
View
Document
Tools
Window
Help
1 / 1
101%
Find

Proovidokument

Dokumendi sisu:

1.

2.

3.

Allkirjastajad:

Allkirjastaja 1

Allkiri 1

Nimi Perekonnanimi

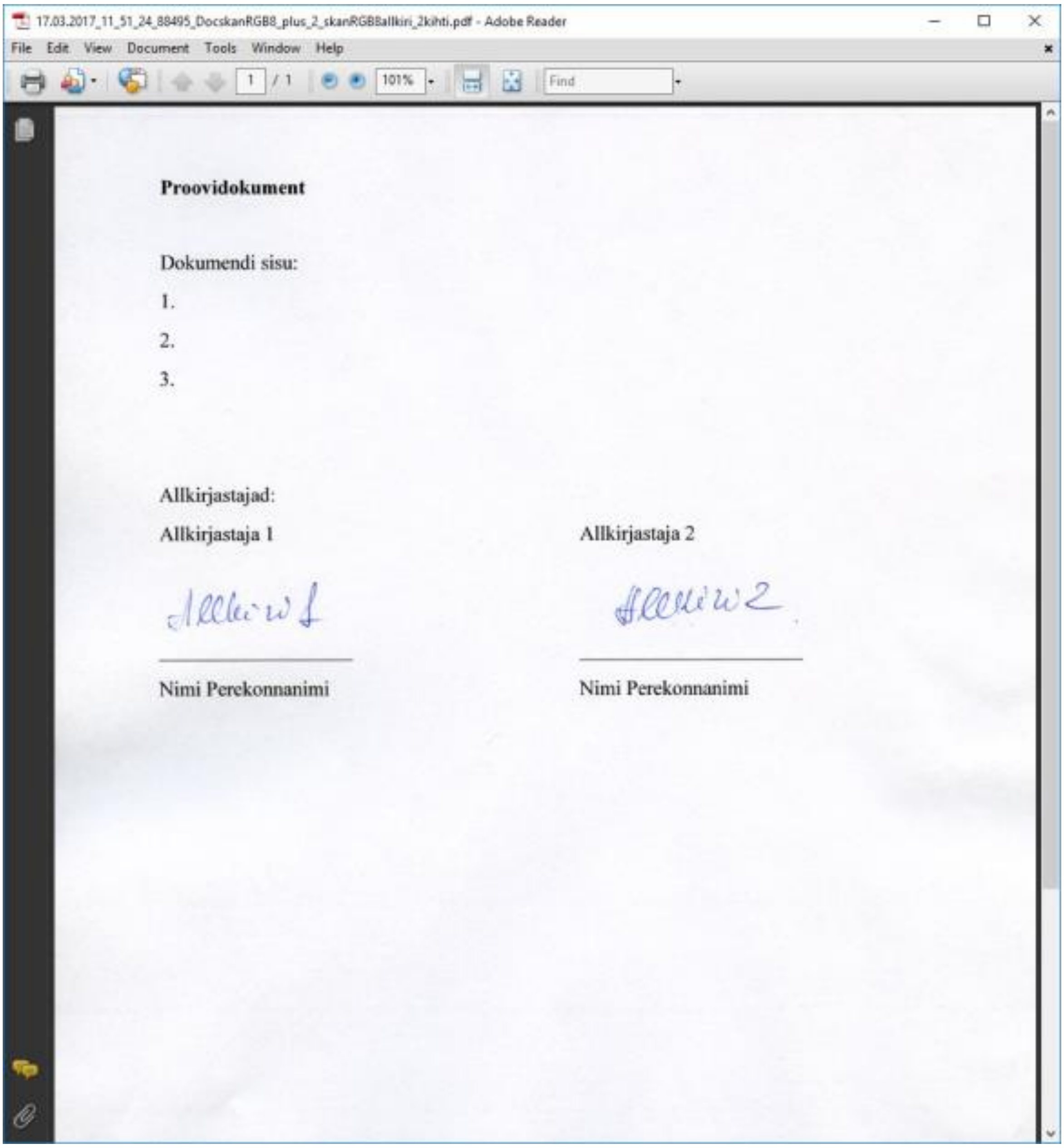
Allkirjastaja 2

Allkiri 2

Nimi Perekonnanimi

Figure 57. Evidence 3 shown by Adobe Acrobat Reader

For *Evidence 2*, shown in Figure 58, ‘allkirjad.pdf’, ‘allkiri1.jpg’ and ‘allkiri2.jpg’were opened in Adobe Photoshop CS2 software, and the two latter were inserted in the former as two different layers, but saved it with ‘Flattened image’ option. Finally, it was saved as a PDF file.



17.03.2017_11_51_24_88495_DocskanRGB8_plus_2_skanRGB8allkiri_2kihti.pdf - Adobe Reader
File
Edit
View
Document
Tools
Window
Help
1 / 1
101%
Find

Proovidokument

Dokumendi sisu:

1.

2.

3.

Allkirjastajad:

Allkirjastaja 1 Allkiri 1

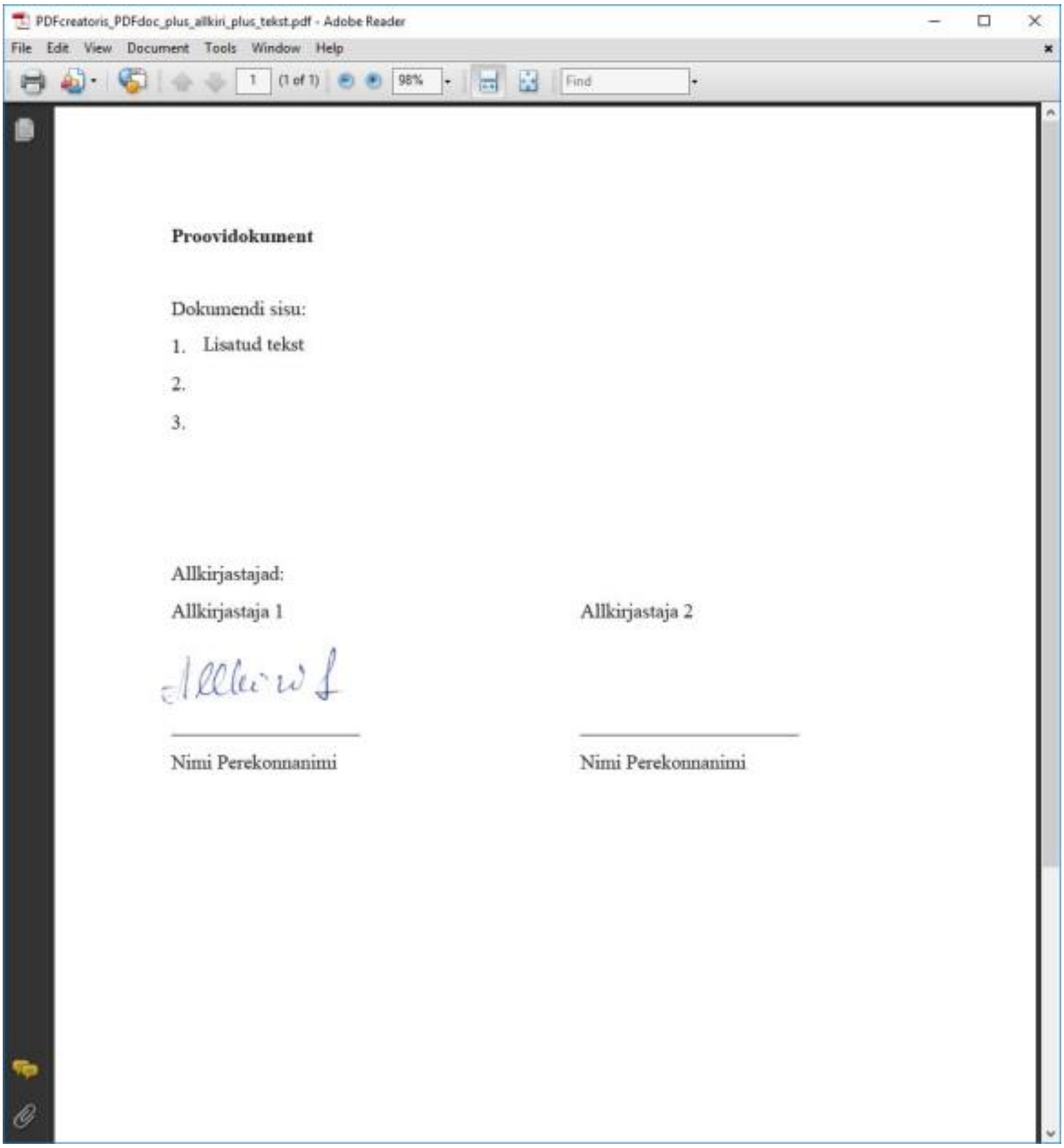
Nimi Perekonnanimi

Allkirjastaja 2 Allkiri 2

Nimi Perekonnanimi

Figure 58. Evidence 2 shown by Adobe Acrobat Reader

For *Evidence 7*, shown in Figure 59, ‘Word_dokument.pdf’ was opened in PDF Creator software, and the utility ‘Sign the document’ has been chosen, selecting the signature from an image file (allkiri1.jpg). Also, the function ‘Add text’ has been used to add the sentence ‘*Lisatud text*’ to the content of the file just after the first text bullet. Then, it has been saved as PDF.


PDFcreatoris_PDFdoc_plus_allkiri_plus_tekst.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 (1 of 1)
98%
Find

Proovidokument

Dokumendi sisu:

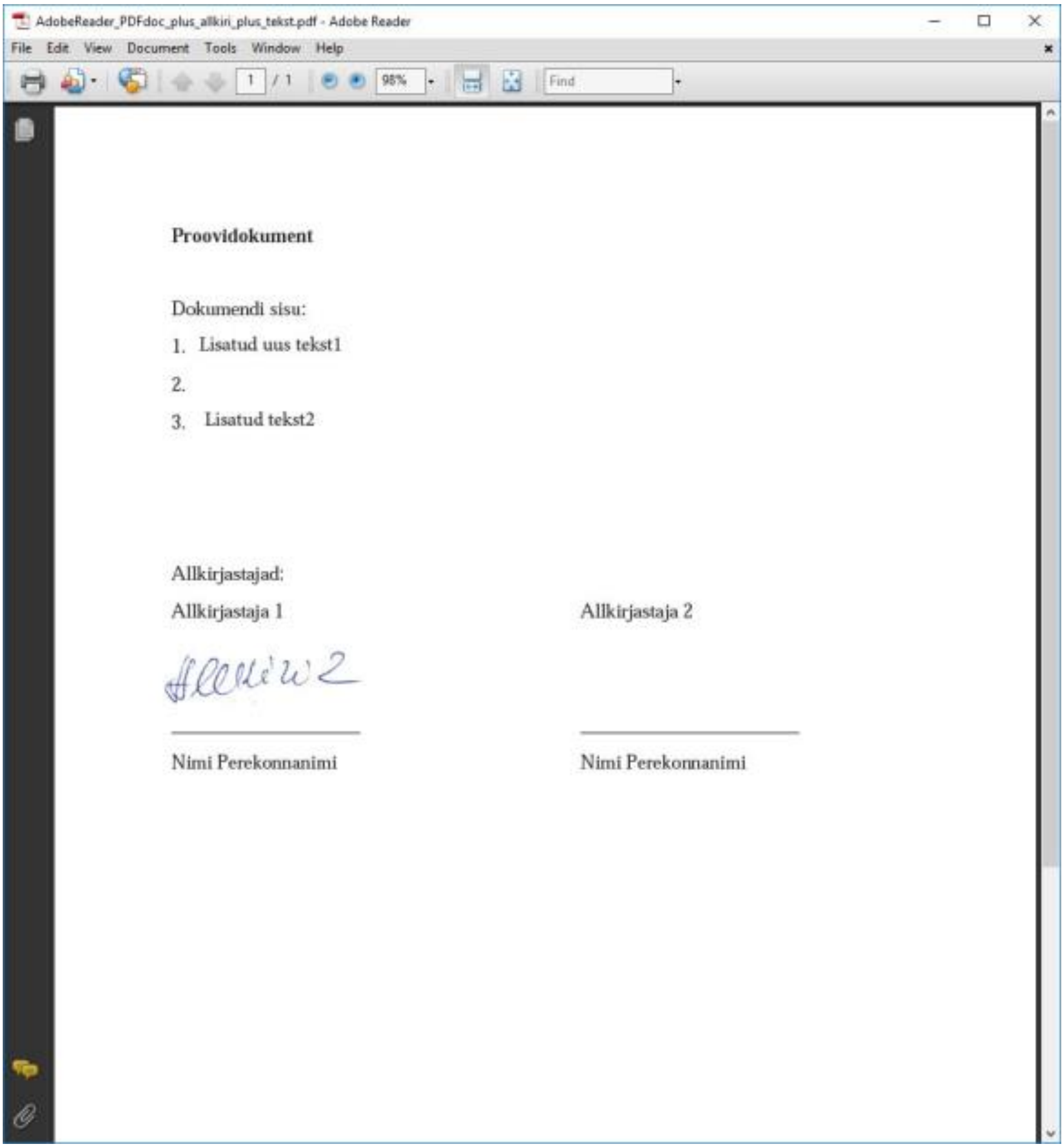
1. Lisatud tekst
2.
3.

Allkirjastajad:

Allkirjastaja 1	Allkirjastaja 2
Nimi Perekonnanimi	Nimi Perekonnanimi

Figure 59. Evidence 7 shown by Adobe Acrobat Reader

For *Evidence 4*, shown in Figure 60, ‘Word_dokument.pdf’ was opened in PDF Creator software, and the utility ‘Sign the document’ has been chosen, selecting the signature from an image file (allkiri2.jpg). Also, the function ‘Add text’ has been used to add the sentence ‘*Lisatud uus text 1*’ next to the first text bullet, and ‘*Lisatud text 2*’ next to the third text bullet. Then, it has been saved as PDF.


AdobeReader_PDFdoc_plus_allkiri_plus_tekst.pdf - Adobe Reader
File Edit View Document Tools Window Help
1 / 1
98%
Find

Proovidokument

Dokumendi sisu:

1. Lisatud uus tekst1
2.
3. Lisatud tekst2

Allkirjastajad:

Allkirjastaja 1

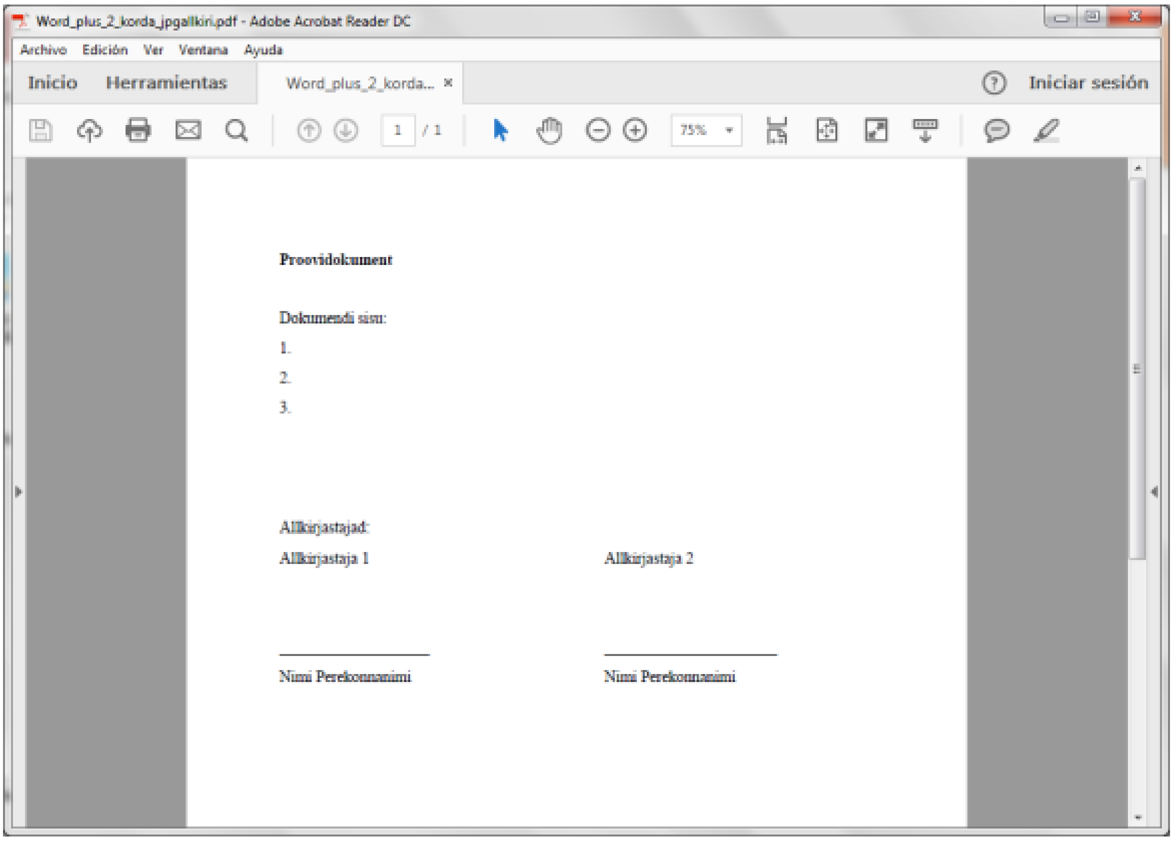
Allkirjastaja 2

Nimi Perekonnanimi

Nimi Perekonnanimi

Figure 60. Evidence 4 shown by Adobe Acrobat Reader

For *Evidence 9*, shown in Figure 61, ‘Word_allkirjadeta.docx’ was opened in Microsoft Word 2013 software, and the utility ‘save as PDF’ has been chosen, producing a brand new PDF file.



Word_plus_2_korda_jpgallkiri.pdf - Adobe Acrobat Reader DC
Archivo Edición Ver Ventana Ayuda
Inicio Herramientas Word_plus_2_korda...
Iniciar sesión
1 / 1
75%

Proovidokument

Dokumendi sisu:

1.

2.

3.

Allkirjastajad:

Allkirjastaja 1 Allkirjastaja 2

Nimi Perekonnanimi Nimi Perekonnanimi

Figure 61. Evidence 9 shown by Adobe Acrobat Reader

V. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Gema Fernández Bascuñana** ,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Method for Effective PDF Files Manipulation Detection,

supervised by Pavel Laptev, Inna Ivask and Raimundas Matulevičius

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **17.05.2017**