

# Forensic Analysis of OOXML Documents

Espen Didriksen



Master's Thesis  
Master of Science in Information Security  
30 ECTS  
Department of Computer Science and Media Technology  
Gjøvik University College, 2014

Avdeling for  
informatikk og medieteknikk  
Høgskolen i Gjøvik  
Postboks 191  
2802 Gjøvik

Department of Computer Science  
and Media Technology  
Gjøvik University College  
Box 191  
N-2802 Gjøvik  
Norway

## Abstract

Microsoft Office 2007 and subsequent versions use an XML-based file format called Office Open XML (OOXML) for storing documents, spreadsheets and presentations. OOXML documents are often collected in forensic investigations, and is considered one of the main sources of evidence by *the National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway* (Norwegian: *Økokrim*).

OOXML documents are zipped file containers which upon extraction reveals a file structure with files containing forensically interesting information. Metadata specified in the XML of these documents can often be used for e.g. attributing a document to a person or correlating time information to build a timeline of events. Revision identifiers are unique numbers appended to content in OOXML documents produced in Microsoft Word, and can be used in forensics to e.g. uncover previously unknown social networks, determine the source of a document and detect plagiarism of intellectual property.

We have used experimental methods to determine the forensic difference between the word processors Microsoft Word 2007, 2010, 2013, 365 and Online, in addition to LibreOffice Writer and Google Docs, with respect to original path preservation of inserted images, thumbnail creation and implementation of revision identifiers. Experimental methods have been used to determine how unique the revision identifiers are, which resulted in detecting that 2 of 100 documents shared revision identifiers without sharing any content, i.e. a 2% false positive rate. This means that revision identifiers can likely be successfully used in forensic investigations.

We present a forensic prototype, with the purpose of exploring the possibilities OOXML documents have in a forensic context. The prototype extracts metadata from documents, in addition to extracting and comparing revision identifiers from a set of documents, and displaying the documents with a relationship in a tree graph layout. This functionality has not previously been published in the existing literature or implemented in forensic tools. Interviews with two digital forensic experts working in law enforcement have determined that this implementation could have value in cases where a large amount of documents are collected.

## Sammendrag

Microsoft Office 2007 og etterfølgende versjoner bruker et XML-basert filformat som kalles Office Open XML (OOXML) for å lagre dokumenter, regneark og presentasjoner. OOXML-dokumenter beslaglegges ofte i etterforskningssammenheng, og regnes av Økokrim for å være en av de viktigste kildene til bevis.

OOXML-dokumenter er zippede fil-containerer som etter ekstraksjon avslører en filstruktur som inneholder informasjon som er interessant i etterforskningssammenheng. Metadata som er spesifisert i XML-filene i disse dokumentene kan ofte brukes til å for eksempel knytte dokumentet til en person eller til å korrelere tidsinformasjon for å bygge en tidslinje bestående av hendelser tilknyttet en sak. Revisjonsidentifikatorer er unike tall som legges til innhold i OOXML-dokumenter som er produsert i Microsoft Word, og kan brukes i etterforskningssammenheng til å for eksempel avsløre sosiale nettverk, fastslå hvor et dokument kommer fra eller detektere plagiat.

Vi har brukt eksperimentelle metoder til å forsøke å fastslå hvorvidt det er en forskjell mellom de forskjellige kontorprogramvarene Microsoft Word 2007, 2010, 2013, 365, Online, i tillegg til LibreOffice og Google Docs, med fokus på preservasjon av original filsti tilhørende bilder som er satt inn i dokumentet, generering av miniatyrbilder samt implementasjon av revisjonsidentifikatorer. Eksperimentelle metoder har blitt brukt til å fastslå hvor unike revisjonsidentifikatorene er, hvilket resulterte i at det ble fastslått at 2 av 100 dokumenter delte revisjonsidentifikatorer uten å dele noe innhold eller andre fellestrekk, med andre ord en 2% falsk-positiv-rate. Dette betyr at revisjonsidentifikatorer sannsynligvis er egnede i etterforskningssammenheng.

Vi presenterer en prototype med den hensikt å utforske hvilke muligheter som finnes med tanke på å bruke OOXML-dokumenter i etterforskningssammenheng. Prototypen ekstraherer metadata fra dokumenter, i tillegg til at den sammenligner revisjonsidentifikatorer fra et sett med dokumenter og viser dokumenter med relasjoner som en tre-graf. Denne typen funksjonalitet har ikke tidligere blitt publisert i eksisterende litteratur eller blitt implementert blant etterforskningsverktøy som finnes i dag. Intervju med to spesialetterforskere som jobber i politiet har vist at denne implementasjonen kan ha verdi i saker hvor et stort antall dokumenter beslaglegges.

## Acknowledgements

I would like to thank my supervisor, Hanno Langweg, for providing useful feedback throughout the course of this thesis.

I would like to thank my classmates André Jung Waltoft-Olsen, Kjetil Tangen Gardåsen, Ola Kjelsrud and Eirik Bae for hours upon hours of interesting discussions and distractions, and all the heavy squats, deadlifts, bench presses and gains together with Eirik and Ola.

Everybody on the forensics lab deserves a thanks; John Erik Rekdal, André Nordbø, Pieter Ruthven and Andrii Shalaginov for all the interesting discussions, shared frustration and laughter.

Thanks to Thomas Walmann in Økokrim for giving us the opportunity to visit, and for providing a lot of quality feedback. Thanks to Kripos for giving me the chance to visit on short notice, and a big thanks to Tom Sørensen Flølo in Kripos for providing his time and excellent feedback. Thanks to Rune Nordvik for using his time to perform analysis and giving me useful information.

Thanks to my father for supporting me throughout my years of education.

And last but not least, a big thanks to my girlfriend and best friend, Marthe Bartholsen Hansen, who has not only supported me during the entire course of this thesis, but also stayed extremely patient the entire time. You're the best, and I love you.

## Contents

<b>Abstract</b> . . . . .	<b>ii</b>
<b>Sammendrag</b> . . . . .	<b>iii</b>
<b>Acknowledgements</b> . . . . .	<b>iv</b>
<b>Contents</b> . . . . .	<b>v</b>
<b>List of Figures</b> . . . . .	<b>viii</b>
<b>List of Tables</b> . . . . .	<b>x</b>
<b>Glossary</b> . . . . .	<b>xi</b>
<b>1 Introduction</b> . . . . .	<b>1</b>
1.1 Topics covered . . . . .	1
1.2 Keywords . . . . .	1
1.3 Problem description . . . . .	1
1.4 Justification, motivation and benefits . . . . .	2
1.5 Research questions . . . . .	3
1.6 Contributions . . . . .	3
1.7 Thesis outline . . . . .	3
<b>2 Related work</b> . . . . .	<b>4</b>
2.1 Background . . . . .	4
2.2 Existing forensics tools for analysing OOXML files . . . . .	9
<b>3 Methodology</b> . . . . .	<b>14</b>
3.1 Scientific methods . . . . .	14
3.2 RQ1: What is the forensic value of OOXML documents, and how can they be used in forensic investigations? . . . . .	15
3.3 RQ2: Can the metadata of OOXML document be trusted? . . . . .	15
3.4 RQ3: Are there differences from version to version of the popular office suites, with respect to what forensically interesting data they record in the files? Does performing certain actions in different ways affect the recorded forensically inter- esting data? . . . . .	16
3.5 RQ4: In what ways can the revision identifiers be useful in a forensic investigation, and in what situations are they preserved? . . . . .	16
<b>4 OOXML file characteristics and use in digital forensics</b> . . . . .	<b>17</b>
4.1 History of the OOXML file format . . . . .	17
4.2 The OOXML package and file structure . . . . .	17
4.3 The forensic usefulness of a single OOXML document's metadata . . . . .	24
4.4 When change tracking is enabled . . . . .	31
4.5 Forensic usefulness of OOXML documents with reference documents . . . . .	32
4.6 Trustworthiness of evidence found in OOXML documents . . . . .	36

<b>5</b>	<b>OOXML Forensic Analysis Tool</b>	<b>41</b>
5.1	Introduction	41
5.2	OOFAT's functionality	41
<b>6</b>	<b>Experimental work</b>	<b>51</b>
6.1	Prerequisite for experiment #4: Collecting data set of test documents	51
6.2	Experiment #1: Interpretation of AppVersion number	52
6.3	Experiment #2: Revision identifier preservation in file and content copying	53
6.4	Experiment #3: Forensic difference between office suites	61
6.5	Experiment #4: Uniqueness of revision identifiers	68
<b>7</b>	<b>Conclusions</b>	<b>71</b>
7.1	RQ1: What is the forensic value of OOXML documents, and how can they be used in forensic investigations?	71
7.2	RQ2: Can the metadata of OOXML document be trusted?	72
7.3	RQ3: Are there differences from version to version of the popular office suites, with respect to what forensically interesting data they record in the files? Does performing certain actions in different ways affect the recorded forensically interesting data?	73
7.4	RQ4: In what ways can the revision identifiers be useful in a forensic investigation, and in what situations are they preserved?	74
<b>8</b>	<b>Future work</b>	<b>76</b>
8.1	Using visualization techniques to support forensic investigators	76
8.2	Optimizing the comparison process	76
8.3	OOXML spreadsheets and presentations in digital forensics	76
8.4	OpenDocument files in digital forensics	76
8.5	Microsoft Office's revision identifier generator algorithm	77
	<b>Bibliography</b>	<b>78</b>
<b>A</b>	<b>Path preservation results table</b>	<b>82</b>
<b>B</b>	<b>Thumbnail creation and readability experiment</b>	<b>85</b>
B.1	Word 2007	85
B.2	Word 2010	87
B.3	Word 2013	89
B.4	Word 365	91
<b>C</b>	<b>Facebook user identification based on inserted image</b>	<b>93</b>
<b>D</b>	<b>Transcription of workshop with National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway (ØKOKRIM), 14/3-2014</b>	<b>94</b>
<b>E</b>	<b>Interview with Tom Sørensen Flølo from National Criminal Investigation Service (Norway) (Kripos)</b>	<b>100</b>
<b>F</b>	<b>EnCase Forensic functionality</b>	<b>102</b>
F.1	EnCase metadata extraction	102
F.2	EnCase image information	103
F.3	EnScript output, extracting Exif metadata	104
F.4	EnCase displaying XML	105

---

F.5	EnCase showing manually altered values in <i>docProps/app.xml</i> . . . . .	106
F.6	ExifTool output of sample image . . . . .	107
<b>G</b>	<b>Forensic Toolkit (FTK) functionality</b> . . . . .	<b>110</b>
G.1	FTK metadata extraction . . . . .	110
G.2	FTK viewing individual XML file . . . . .	111
G.3	Sample document . . . . .	112
<b>H</b>	<b>Change tracking example</b> . . . . .	<b>125</b>
H.1	Screenshot of document edited with change tracking enabled . . . . .	125
H.2	XML of document edited with change tracking enabled . . . . .	125
H.3	Uniqueness of revision identifiers result table . . . . .	126
H.4	Application information extracted from data set . . . . .	130
<b>I</b>	<b>Contents of <i>docProps/custom.xml</i> in sample document</b> . . . . .	<b>131</b>
<b>J</b>	<b>Source code of OOFAT's most important functionality</b> . . . . .	<b>134</b>
J.1	Document validator . . . . .	134
J.2	Document metadata extractor . . . . .	136
J.3	Revision identifier extraction . . . . .	140
J.4	Revision identifier comparison . . . . .	142
J.5	Tree graph layout output . . . . .	144



## List of Figures

1	Screenshot of Langweg’s prototype. . . . .	10
2	Screenshot of the DSO tool . . . . .	11
3	Metadata files in an OOXML document, adapted from [1][p. 4985] . . . . .	21
4	Merging two sample documents. . . . .	22
5	Barcode visualization of language settings throughout the document, from the complete version of [2] . . . . .	27
6	Barcode visualization of paragraph creation revisions of document in Appendix G.3	29
7	Diary entry of the suspect in the scenario. . . . .	31
8	Flowchart showing a process of checking a collected document against database of known sensitive documents. . . . .	33
9	Hypothetical social network of people sending emails with attachments to each other. . . . .	35
10	Flowchart showing the validation process. . . . .	42
11	Flowchart showing the process of metadata extraction. . . . .	43
12	Revision identifier extraction process. . . . .	45
13	Revision identifier comparison process. . . . .	46
14	Table output of revision identifier comparison process. . . . .	48
15	Example graph showing the relationship between four documents. . . . .	49
16	Example graph showing the relationship between documents from data set. . . . .	49
17	Example details page showing the information associated with a clicked edge. . . . .	50
18	Example details page showing the complete metadata of a document of interest. . . . .	50
19	Chart showing the file size distribution of the collected data set. . . . .	52
20	Example document with six revisions, and its revisions recorded in settings.xml. . . . .	59
21	Tree graph layout of OOFAT showing the first iteration of revision identifier comparison, used for inspecting document pairs to determine their relationship. . . . .	69
22	Thumbnail produced by Office 2007 . . . . .	85
23	Screenshot of first page of Office 2007 document . . . . .	86
24	Thumbnail produced by Office 2010 . . . . .	87
25	Screenshot of first page of Office 2010 document . . . . .	88
26	Thumbnail produced by Office 2013 . . . . .	89
27	Screenshot of first page of Office 2013 document . . . . .	90
28	Thumbnail produced by Office 365 . . . . .	91
29	Screenshot of first page of Office 365 document . . . . .	92
30	Screenshot of Facebook user’s published image, identified based on inserted image’s original filename . . . . .	93
31	Result of EnCase extracting metadata from sample OOXML documents . . . . .	102

32	Result of EnCase extracting information from sample image . . . . .	103
33	Output of sample EnScript extracting Exif metadata from sample inserted image .	104
34	EnCase displaying XML of sample file in document package . . . . .	105
35	EnCase showing manually altered values in <i>docProps/app.xml</i> . . . . .	106
36	Screenshot of FTK extracting metadata from an OOXML document. . . . .	110
37	Screenshot of FTK viewing individual XML file. . . . .	111
38	Screenshot of sample document, with paragraph revisions marked. . . . .	112
39	Screenshot of document edited with change tracking enabled . . . . .	125

## List of Tables

1	The six keywords for questions investigators may seek to have answered, adapted from [3]. . . . .	5
2	Comparison of EnCase Forensic and FTK's metadata extraction . . . . .	13
4	Metadata recorded in docProps/core.xml, adapted from [1][p. 4985 - 4986], [4][p. 41] . . . . .	18
5	Metadata recorded in docProps/app.xml, adapted from [1][p. 4986 - 4987] . . . .	19
6	Types of revision identifiers in OOXML documents. . . . .	23
3	File structure of extracted sample document. . . . .	40
7	AppVersion interpretation experiment results . . . . .	53
8	Implementation of revision identifiers in office suites; creating new OOXML document . . . . .	63
9	Implementation of revision identifiers in office suites; editing OOXML document made in Office 2007 . . . . .	63
10	Original path preservation results of image insertion . . . . .	64
11	Thumbnail creation and their readability . . . . .	66
12	Original path preservation results of image insertion (extended version) . . . . .	82
13	ExifTool output of sample image . . . . .	107
14	Classification table; description of each number in Table 15 . . . . .	126
15	Result of uniqueness of revision identifiers experiment. Column name "Cl" refers to "Classification" (see Table 14); "FP" refers to "false positive" . . . . .	126
16	Application information extracted from data set. . . . .	130

## Glossary

**OOXML** Office Open XML, a Microsoft-developed file format for storing files such as documents, presentations and spreadsheets.

**OOXML document** A *WordprocessingML* package following the specifications of OOXML, used to represent a document. An OOXML document has the file extension *.docx*.

**Revision identifier** A 32-bit number represented in hexadecimal, used to determine in what session the associated content was edited. All content within a document sharing the same revision identifier value was edited during the same editing session, i.e. the period of time between two saves.

**Intersecting revision identifiers** Two documents sharing the same revision identifier value(s) are said to have *intersecting revision identifiers*.

# 1 Introduction

## 1.1 Topics covered

Computers are commonly used for running office suites to create, modify and view files including documents, spreadsheets and presentations. Such use of computers is especially common in professional environments, but is also massively used by individuals for private purposes.

Microsoft Office is a very popular office suite that runs on Microsoft Windows and Mac OS X. While previous versions of Microsoft Office used proprietary binary formats for storing the files edited with their software, current versions compose documents by using an XML-based format called Office Open XML (shortened “OpenXML” or “OOXML”) [5][p. 1]. The alternative office suite Google Docs has functionality to import and export OOXML documents, and LibreOffice by default use Open Document Open Document Format for Office Applications (ODF), but also support OOXML.

In this thesis, we examine the forensically interesting data stored in OOXML documents edited by the popular word processors Microsoft Word (Office 2007, 2010, 2013 and 365<sup>1</sup> and Online<sup>2</sup>), LibreOffice Writer and Google Docs, and attempt to identify scenarios where the information can be used in a forensic investigation. Our focus is primarily on Microsoft Word, but the other word processors are inspected in several of our experiments. We furthermore attempt to determine if OOXML documents have unexplored forensic possibilities. A prototype forensic tool is presented, with the purpose of demonstrating the identified possibilities.

## 1.2 Keywords

Digital forensics, Metadata, Document structure analysis, Revision identifiers, OOXML forensics, Microsoft Office forensics, DOCX.

## 1.3 Problem description

The most popular office suites of today store their files in zipped XML-based containers. Microsoft Office 2007 and subsequent versions store the document and all its related information in a file container format called OOXML, as opposed to previously proprietary binary formats. Other alternatives such as LibreOffice and OpenOffice.org save their files as ODF by default, but have read and write support for OOXML documents.

Documents that are produced with word processors are often part of a forensic investigation, e.g. extracted from seized media from computers in a company that is under investigation, and

---

<sup>1</sup>Microsoft’s subscription-based access to Office [6].

<sup>2</sup>Microsoft’s online version of Office [7].

is by some considered the main source of evidence [8][Appendix D]. The XML of these files contain data that may support investigators in e.g. determining the source of a document, building a timeline of criminal events, uncovering social networks and detecting plagiarism of intellectual property [5][p. 1][9][p. 4].

Currently available forensic tools tend to only present the information from documents, without providing any analysis or interpretation functionality. Some commercial forensic tools fail to extract every type of metadata available in the XML files in the document package. No research has been published on the whether the forensically interesting information in OOXML documents should be considered trustworthy, which could be of high importance if the trustworthiness of the evidence is disputed in a court of law.

In this thesis, we attempt to determine how the information contained in OOXML document can be used in forensic investigations. Since there is possibility that the various office suites store different forensically interesting information, experiments will be performed to determine if some office suites record more or less forensically interesting data. In order to demonstrate the identified analysis possibilities, a prototype forensic tool is built.

#### **1.4 Justification, motivation and benefits**

Forensic investigators work under a time pressure, and might not have the resources to perform manual analysis of seized files. This is particularly true if the investigators are faced with a large amount of documents in a case, which could make it unfeasible to inspect each document manually. Information extracted from documents is considered the main source of evidence for National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway (Norwegian: ØKOKRIM) [8][Appendix D], and this motivates research being performed on using OOXML documents in a forensic context.

Currently available forensic tools supporting OOXML fail to extract every type of metadata available in OOXML documents, although they are easily retrievable. OOXML documents contain unique identifiers that could be used for document tracking to e.g. uncover previously unknown social networks [9][p. 3-4]. None of the currently available forensic tools have implemented this, even though it could have value in cases where the goal is to track the source of a document. This could for example be used in uncovering extremist networks [10][Appendix E].

Existing published research on OOXML documents in the context of digital forensics only provide a brief overview of some of the interesting characteristics of OOXML documents and the office suites supporting the file format. Having an indepth understanding of the topic is important in order to properly utilize the possibilities of documents in a forensic context, and to know if there are any uncertainties that should be taken into consideration. Currently available forensic tools seem to not have prioritized OOXML files, and leveraging the currently unexplored possibilities or extending currently weak functionality could be directly beneficial in forensic investigations, and could motivate future forensic tool developers to implement the possibilities.

## 1.5 Research questions

The following list provides the main research questions we attempt to answer in this thesis.

1. What is the forensic value of OOXML documents, and how can they be used in forensic investigations?
2. Can the metadata of OOXML document be trusted?
3. Are there differences from version to version of the popular office suites, with respect to what forensically interesting data they record in the files? Does performing certain actions in different ways affect the recorded forensically interesting data?
4. In what ways can the revision identifiers be useful in a forensic investigation, and in what situations are they preserved?

## 1.6 Contributions

This thesis seeks to provide a detailed understanding of the characteristics of OOXML documents, and how they can be utilized in a forensic context. As part of the task of identifying and demonstrating the possibilities OOXML documents have in a forensic setting, a prototype has been built for future use for forensic investigators and forensic tool developer. The inner workings of this prototype is presented in Section 5.

We have identified some forensically interesting information in OOXML documents, and relate them to use case scenarios in Section 4. These hypothetical scenarios are provided to demonstrate the possibilities information extracted from OOXML documents may have in various types of forensic investigations, both when faced with just one OOXML document and when reference documents are available.

## 1.7 Thesis outline

The following list provides a short outline of the following chapters of this thesis.

- Chapter 2 presents related work; both published literature and available forensic tools.
- Chapter 3 explains the methods utilized in order to answer the research questions.
- Chapter 4 describes the characteristics and structure of OOXML documents, and relates them to usefulness in digital forensics.
- Chapter 5 presents the forensic prototype that was developed during this thesis work.
- Chapter 6 provides detailed descriptions of the experiments conducted in this thesis; experiment setup, experiment execution, experiment results and experiment discussion.
- Chapter 7 provides conclusions for each research question.
- Chapter 8 presents our recommendations for future work.

## 2 Related work

In this chapter, we identify work that has been done in the field, both related literature and published forensics tools.

### 2.1 Background

This section first presents related literature giving an overview of digital forensics in general, then narrows the literature down to what types of information is typically desired in a forensic investigation, and then moves towards literature on using OOXML documents in forensic settings.

#### 2.1.1 Digital forensics, digital evidence and metadata

The report from the first Digital Forensic Research Workshop (DFRWS) presented the following definition of *digital forensic science*:

“The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations” [11][p. 16].

Digital forensics as a field of science is relatively new; its development started growing during the late 1990s and early 2000s when crimes involving computers increased. While the term *computer forensics* was originally used to describe the field, this was changed as the use of other types of digital units became widespread in society, and digital evidence no longer were retrieved exclusively from computers [12][p. 1].

Casey describes *forensic* as “a characteristic of evidence that satisfies its suitability for admission as fact and its ability to persuade based upon proof (or high statistical confidence)” [13][p. 14]. Although the term typically refers to the use and admissibility of evidence in a court of law, it is also used in relation to e.g. corporate investigations where the goal could be to determine if an employee has broken any corporate policies [13][p. 15].

Casey defines *digital evidence* as “any data stored or transmitted using a computer that support or refute a theory of how an offense occurred or that address critical elements of the offense such as intent or alibi” [13][p. 7]. Even though a piece of evidence might be strong on its own, correlating several pieces of evidence might be used to build an even stronger case to support or refute a hypothesis the investigators have formed [13][p. 16].

*Metadata* is a commonly used term in digital forensics and other communities, and is defined



as “structured information that describes, explains, locates, or otherwise makes it easier to retrieve, use, or manage an information resource. Metadata is often called data about data or information about information” [14][p. 1]. File metadata is often set by the application creating the file, and could include information such as when the file was created and by whom. There are three main types of metadata in general [14][p. 1]:

- *Descriptive metadata*: Used for discovery and identification of the resource, e.g. *title, abstract, author* and *keywords*.
- *Structural metadata*: Used to explain how different parts of the resource are put together, e.g. the order of pages.
- *Administrative metadata*: Technical information used to manage the resource, e.g. creation timestamps, how the resource was created, file type of the resource.

### 2.1.2 The role of metadata in investigations

Buchholz and Spafford identified what information in a system is relevant for forensic investigators, in particular what role metadata have in forensic investigations. In addition to presenting some examples of what forensically information is available<sup>1</sup>, they also presented types of information they considered beneficial for forensic investigators in future systems. One of the main goals in a forensic investigation is to as fully as possible reconstruct events that have occurred, in order to support or refute hypotheses that arise based on the available information [3][p. 5].

Buchholz and Spafford identified six keywords for questions that forensic investigators may seek to get answered in an investigation: *Who, what, when, how, where* and *why*. Table 1 provides a short summary of the meaning of each of the keywords.

Table 1: The six keywords for questions investigators may seek to have answered, adapted from [3].

Question	Description
Who?	The person or the people responsible or associated with the actions.
What?	The type of actions that occurred.
When?	The period of time the actions were performed.
How?	How the actions were performed.
Where?	The location of the person or people responsible or associated with the actions, alternatively the origin of a file.
Why?	The motives of the person or the people who performed the actions.

<sup>1</sup>At the time of writing the paper, in 2004.

Determining the physical person or people responsible for performing the action is often important in forensic investigations, referred to as *who* by Buchholz et al. [3][p. 6]. Although inspecting the metadata of a particular file may result in determining that a particular user identifier performed a certain action, it is difficult to conclude that it actually was this user identifier and this particular person who was responsible for performing the action [3][p. 6]. One example could be if an employee in a company neglects to lock his computer when leaving the office, and an adversary with physical access performs certain actions while the employee is absent. Examining the metadata of the affected files will show that the employee performed the actions although it was an adversary, and therefore it is not wise to directly conclude who did it without additional evidence.

The origin of a certain file or several files could be very important in a forensic investigation, referred to as *where* by Buchholz et al. [3][p. 7]. Such information could be e.g. attributed GPS coordinates, IP addresses or a Globally Unique Identifier (GUID), or any other piece of information that could be used to determine the location of the person responsible (the *who*). Determining the origin of a file is often not a simple task, but may sometimes be possible by correlating available information [3][p. 7]. There are many examples for why it might be desirable to track the person responsible for a file or actions related to a file; examples include distribution of illegal pornography, terrorist threats and malicious software.

The point of time or interval of time an action was performed, referred to as *when*, is often interesting for forensic investigators. Timestamps from various sources are often used to build a timeline of events in case, in order to pinpoint what happened and at what time, and thereby determine the order of the occurring events. However, different file systems and operating systems tend to record different timestamps, and the exact meaning of the timestamps might in some cases be ambiguous [3][p. 8].

*How* an action was performed could be important in a forensic investigation. It is, for example, relevant for an investigator whether it appears that the user performed the action, or if it was performed by malicious software. Such information is, however, usually not recorded in the and might therefore not available for investigators [3][p. 9]. Determining exactly *what* was done in a system or in a file could naturally also be very forensically interesting [3][p. 10]. This information may or may not be available, in some cases depending on the level of detail required to verify or refute a hypothesis. Lastly, determining *why* the actions were performed could be particularly important when the evidence reaches a court of law.

### **2.1.3 Using unique revision identifiers of OOXML documents to detect copied content**

Fu et al. [5] wrote one of the most comprehensive papers about the OOXML format in the context of digital forensics. In this work, the researchers performed experiments consisting of creating and deconstructing documents in order to see what forensically interesting data is stored in the container file when a document is edited in Microsoft Word. While their main focus is using the recorded information for investigating illegal copying of documents, it is still applicable for other

types of investigations [9].

The particular piece of information they focused on is the concept of unique revision identifiers used by Microsoft Word for document comparison. These unique revision identifiers are 8-digit hexadecimal attributes appended to content for each revision of the document. Once a revision identifier is created by the word processor, it will not change during the evolution of the document as long as at least one printable character remains unaltered [5][p. 4].

The researchers interestingly observed that when content is copied from one document to another, the unique revision identifiers are preserved as long as one or more characters remain with the same formatting. This also applies if a document is copied and the content is changed. Based on this, they propose a method for determining whether documents originate from the same source by extracting and comparing the revision identifiers of several documents. If any identifiers match, the suspicious document can be assumed to be a copy of the original document or contain content copy-pasted from the original document [5][p. 5].

The result of this is that it is possible to prove that the suspicious document contains content that is likely copied from another document, even though the text might e.g. be rewritten and not appear to be copied. Therefore, the concept of unique revision identifiers may be used to help determine the source of a document, e.g. in an investigation involving plagiarism.

#### **2.1.4 Forensically interesting information stored in OOXML documents**

Garfinkel et al. [9] went into depth in both OOXML and ODF files for forensic purposes by performing similar experiments as Fu et al. [5].

The unique revision identifiers are also in this paper presented as a potentially very important source of information, similar to what Fu et al. [5] presented. In addition to detecting plagiarism, another area of usage presented is to identify social networks that previously may have been unknown. This would be done by collecting documents from several suspicious sources and comparing the revision identifiers. In case any of them match, it could be assumed that those in possession of the documents have been communicating [9][p. 3-4].

In addition to using the revision identifiers to determine how a OOXML file is constructed and edited, they also discuss the possibility of creating a database containing the extracted revision identifiers of documents collected in an organization that is under investigation. The idea is to generate an alert when revision identifiers in a collected suspicious document matches any of those in the database. The suspicious document could e.g. be input to the forensic tool manually, or collected automatically on a network [9][p. 5].

Timestamps are often important in a forensic investigation, as they could e.g. show when a document was last modified. OOXML and ODF documents may contain several sources of time information: Created and modified time of the container file, inside the XML files and as meta-

data of any embedded files such as images [9][p. 5].

Images and objects that are embedded in a document are stored as separate files in the file container. Some word processing software saves a thumbnail of the first page of the document, which could be very useful in a forensic investigation. In the case where the thumbnail and the actual document do not match, this could indicate that the document or the thumbnail have been maliciously altered. If the retrieved document is damaged, the content of the thumbnail could indicate what the document contained, as long as the thumbnail is still intact [9][p. 3].

Since some documents might be more complexly built than other, it is important that forensic tools used for analysis are built in such a way that they do not fail due to the input documents' complexity. One example of this is when a document contains another embedded document. In such case, the forensic tool should be able to recursively analyse each document to ensure that no data is unintentionally ignored [9][p. 2]. Garfinkel et al. note that the forensic tool must not take any shortcuts when analyzing XML document containers, in order to avoid false negatives. The tool must be able to handle e.g. strings that are represented with hexadecimal encoding, and strings containing comments. If the tool does not take such obfuscations into account, string searches will likely fail and may lead to important data being ignored.

The researchers note that Microsoft Office 2008 stores a JPG thumbnail of the first page of the document by default, while NeoOffice stores both a PNG and a PDF of the first page. Furthermore, they discovered that Microsoft Word 2007 does not store a thumbnail by default, as this is opt-in functionality [9][p. 3]. These details provide a good starting point for further research on forensic differences between office suites, as it indicates that differences exist.

#### **2.1.5 Forensic analysis of the July 22nd terrorist document**

Norway experienced a terrorist attack July 22nd, 2011. Prior to the attacks, the terrorist distributed a “manifest” containing descriptions of his planned attacks, motivation, diaries, etc. The document was distributed as a large OOXML document, and quickly appeared on various websites. Langweg performed a forensic analysis of the document, attempting to determine if there were several contributors to the document as the terrorist at the time claimed [2][p. 1].

In his research, Langweg performed content and structural analysis of the terrorist document in order to determine i) if there were any evidence of any other authors contributing to the document, ii) if the document's structure appeared to be consistent with the events presented in the diary section of the document. The structural analysis part consisted of looking at how the document was composed, by analysing the document's table of contents, revision identifiers and changes in format and language in paragraphs [2]. The content analysis part consisted of looking at how the text was “divided into logical parts”, the origin and usage of images, language usage throughout the document, and inconsistencies in wording or the described events [2][p. 1].

Langweg extracted the metadata contained in *docProps/app.xml*, *docProps/core.xml* and *settings.xml*,

and based on this information he was able to come to several conclusions. For example, he concluded that the document likely was composed from other sources, based on timestamps and the recorded editing time. Based on the values of XML elements specifying “theme font language” and decimal symbol format, he furthermore concluded that the document was created and edited on a system with Norway set as the location in the settings of the operating system [2][p. 7]<sup>2</sup>.

Langweg developed a forensic analysis prototype which he utilized in his analysis of the document. In this prototype, he implemented visualization techniques in the form of horizontal “barcodes” graphically representing i) creation revisions, ii) modification revisions, iii) glyph modification, which is further described in Section 2.2. Visualizing these details extracted from a document of such size provided an overview of the document’s composition, which is knowledge that would be difficult to gain based on manually investigating the XML document since the document’s body is 886 664 lines.

## 2.2 Existing forensics tools for analysing OOXML files

This section presents currently available forensics tools used for performing analysis of OOXML documents.

### **read\_open\_xml.pl**

Kristinn Gudjonsson has written a Perl script known as `read_open_xml.pl`, for the purpose of extracting the metadata of OOXML documents [15]. It takes an OOXML document as input, extracts it and reads the data stored in `docProps/app.xml` and `docProps/core.xml`, which contains document metadata such as the title, author, number of revisions, number of pages, last printed timestamp, created timestamp, modified timestamp, total editing time, name and version of word processor [15].

While the script could be very useful for quickly extracting metadata of documents without needing to parse the XML files manually, it only deals with the metadata contained in two of the XML files. It does not analyse nor present the markup and content of e.g. the XML file containing the document content itself, `word/document.xml`, which contains revision identifiers that could be useful for determining the history and source of the document.

### **DOCXRevisions**

DOCXRevisions is an unpublished tool which was made to perform the analysis work Langweg performed [2]. It extracts the revision identifiers from `document.xml` and separates them into each category they belong to; default run revision identifiers, paragraph revision identifiers and paragraph glyph formatting identifiers. A screenshot of the tool is provided in Figure 1, with a sample document loaded.

The tool has functionality to create a colored horizontal barcode that represents the editing composition of the document, i.e. how much content was added in that editing session compared to

---

<sup>2</sup>This information is from the “full”, unpublished version of Langweg’s paper.

the total content of the document. This is based on the number of revision identifiers attributed to each run, paragraph or paragraph glyph. Visualizing the editing history of the document with the colored barcodes is a convenient way of understanding the document, as this knowledge is very hard to derive from analysing the XML structure manually.

As DOCXRevisions was made for the purpose of analysing a specific document and has not been published, it obviously lacks some functionality that could be beneficial if the tool would be used by investigators. It is not apparent if the revision identifiers presented in each listbox belong to a *run*, *paragraph* or *paragraph glyph*. It could also be beneficial to present the content belonging to each revision identifier, e.g. when clicking on each value.

The tool does not extract and present other available metadata that are found in the XML files, e.g. *creator*, *company*, *creation time* etc., which forensic investigators very likely would consider beneficial. Due to the intended purpose of the tool, it lacks the possibility of performing analysis in bulk, which could be a very useful feature [8].

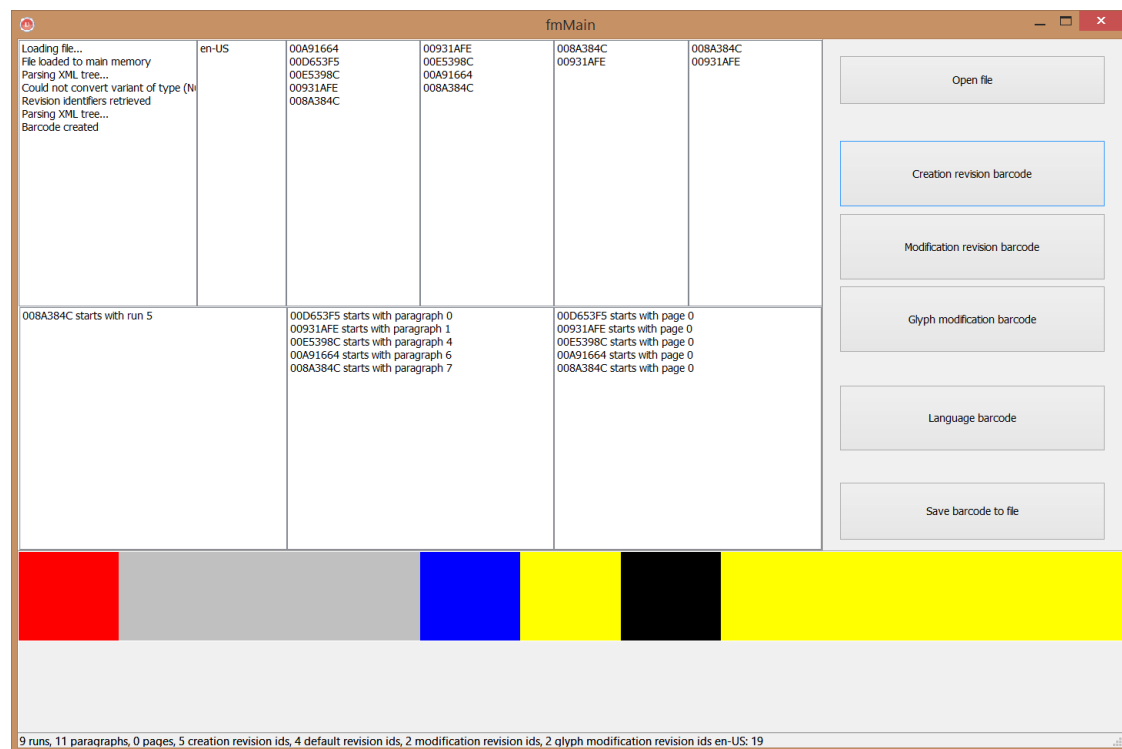


Figure 1: Screenshot of Langweg's prototype.

### DSO Tool: Detector for the Source of OOXML file

The DSO Tool (“Detector for the Source of OOXML file”) is published in relation to the research presented by Fu et al. [5]. It takes two OOXML files as input, and outputs the *creation timestamp*,

last modified timestamp, creator, last modified by and number of revisions for both documents. As the intended purpose of the tool is to detect copied content, e.g. plagiarism, it compares the revision identifiers extracted from both documents. In the case where any revision identifiers are identical, the values and their corresponding printable text is output as a row in a table. Figure 2 provides a screenshot of the tool, showing two sample documents with some identical revision identifiers.

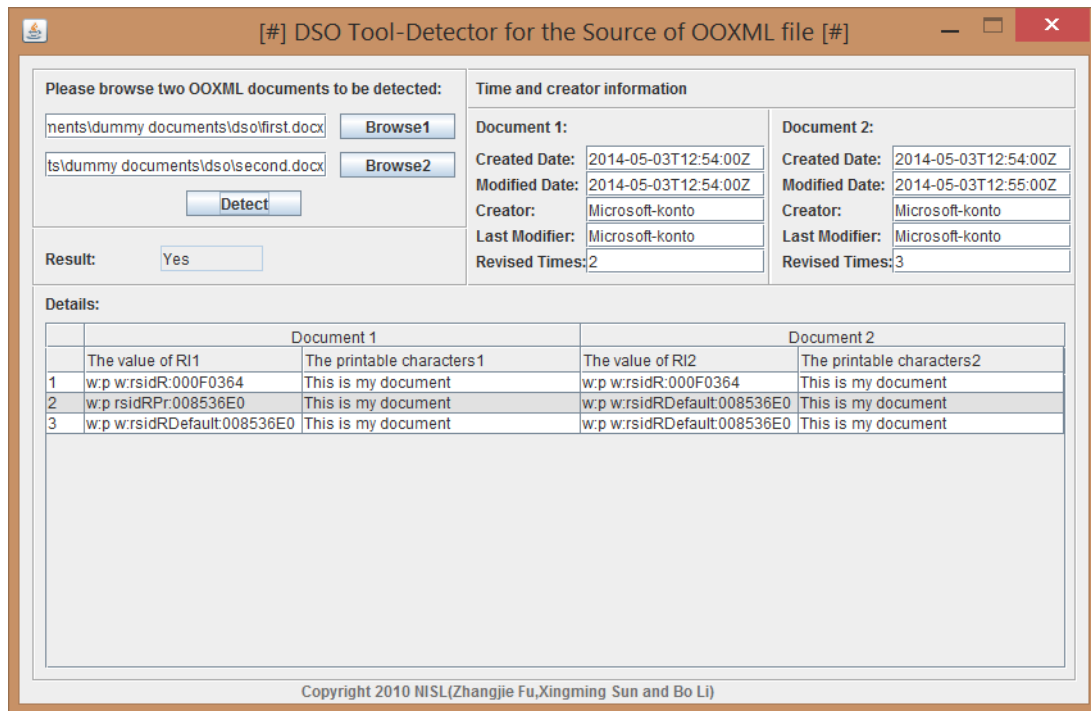


Figure 2: Screenshot of the DSO tool

### EnCase Forensic

EnCase Forensic is a commercial digital forensic tool used acquire and examine information in forensic investigations, from e.g. computers, smartphones and tablets [16]. EnCase is one of the most commonly used forensic tools used by law enforcement agencies in forensic investigations in Norway [8] [10]. We inspected some sample OOXML documents by using EnCase Forensic 6.18 and 7.09, in order to determine what functionality the tool has with respect to handling OOXML documents.

OOXML documents loaded in EnCase must be extracted manually through their extraction method, which is accessed by right-clicking on the loaded document. Embedded objects, such as another OOXML document or a spreadsheet, is likewise not automatically extracted, and must therefore be manually extracted if it is desirable to inspect the embedded object's associated XML.

EnCase by default only extract some parts of the metadata available in OOXML documents, and does not extract a number of other types of information available in *docProps/core.xml* and *docProps/app.xml*. Table 2 shows the types of information EnCase extracts, compared to FTK. Appendix F.1 provides a screenshot of EnCase extracting metadata from two sample OOXML documents.

EnCase by default shows very limited information about inserted images, but is extendable by scripts written in EnCase’s scripting language, “EnScript”. Appendix F.2 provides a screenshot of the information output from a sample inserted image, and Appendix F.3 provides a screenshot of the output of a sample EnScript for extracting some Exif metadata from images. For reference, Appendix F.6 provides the output of ExifTool<sup>3</sup>’s Exif metadata extraction performed on the same image as in Appendix F.2, F.3.

We note that EnCase by default neither displays XML with syntax-highlighting, nor with proper line breaks suited for human analysis, as shown in Appendix F.4. When OOXML documents are inspected with EnCase, the tool by default only displays the information without further interpretation. Although the sample document loaded in EnCase contained irregularities, in this case manually altered XML metadata values creating a large obvious mismatch between the metadata values and the document itself, no indications were provided. Appendix F.5 shows a screenshot of EnCase displaying some values that obviously are altered; the metadata claims the document is 150 pages, while it in reality is only one page.

Lastly, it should be mentioned that although EnCase seems to be lacking certain functionality that could be desirable with respect to inspecting OOXML documents, their scripting language provides a possibility of extending the functionality to perform tasks that are not possible by default. Furthermore, such scripts are often shared on the closed community forum which those who have purchased EnCase get access to [10][Appendix E].

### FTK

Forensic Toolkit (FTK) is a commercial forensic tool for creating forensic images, browsing seized file systems, viewing individual seized files, visualizing evidence and performing various evidence analysis [17]. We tested the functionality of FTK 3.4.1.34295, and observed that it seemed to by default has more functionality than EnCase Forensic for handling OOXML documents. Unlike EnCase, OOXML documents loaded in FTK are extracted automatically. Table 2 shows the types of metadata FTK extracts from *docProps/app.xml* and *docProps/core.xml* in OOXML documents, compared to EnCase Forensic.

The table shows that FTK extracts some more information than EnCase Forensic, but not all types of information. For example, the version of the word processor used to create the document, as reflected in the *AppVersion* element, could be interesting for an investigator, but is not extracted by neither FTK nor EnCase. Appendix G.1 shows a screenshot of the output of FTK’s

<sup>3</sup>Free program for Exif metadata extraction and manipulation; available at <http://www.sno.phy.queensu.ca/~phil/exiftool/>



metadata extraction. We note that the output when viewing the content of individual XML files is easier for humans to interpret in FTK compared to EnCase, since FTK uses proper line breaks and syntax highlighting. Appendix G.2 shows a screenshot of FTK displaying the XML of a sample file, in addition to showing some information about of the files in the OOXML package. Similar to EnCase, FTK only displays the information it extracts from OOXML files, without interpretation.

Table 2: Comparison of EnCase Forensic and FTK's metadata extraction

Metadata	File	EnCase Forensic	Forensic Toolkit (FTK)
category	core.xml		
contentStatus	core.xml		
Created	core.xml	x	x
Creator	core.xml	x	x
description	core.xml		
identifier	core.xml		
keywords	core.xml	x	x
language	core.xml		
lastModifiedBy	core.xml	x	x
lastPrinted	core.xml		
modified	core.xml	x	x
revision	core.xml	x	x
subject	core.xml	x	x
title	core.xml	x	x
version	core.xml	x	
Application	app.xml		x
AppVersion	app.xml		
Characters	app.xml	x	x
CharactersWithSpaces	app.xml		
Company	app.xml	x	x
DigSig	app.xml		
DocSecurity	app.xml		x
HeadingPairs	app.xml		
HLinks	app.xml		
HyperlinkBase	app.xml		
HyperlinksChanged	app.xml		
Lines	app.xml		x
LinksUpToDate	app.xml		x
Manager	app.xml		
MMClips	app.xml		
Pages	app.xml	x	x
Paragraphs	app.xml	x	x
Properties	app.xml		
ScaleCrop	app.xml		x
SharedDoc	app.xml		
Template	app.xml		x
TitlesOfParts	app.xml		
TotalTime	app.xml		x
Words	app.xml	x	x

## 3 Methodology

This chapter provides descriptions of the methods used to attempt answering the research questions of this thesis, in addition to reasons why each method is appropriate for each research question and how each method is applied in practical terms.

### 3.1 Scientific methods

#### 3.1.1 Qualitative research: Case studies

Qualitative research methods are often characterised by the utilization of observations of “real world” situations as the foundation for gathering data and understanding the phenomenon that is the subject of study [18][p. 139]. Case studies fall into the category of qualitative research, and are used as a tool to understand and derive knowledge from a particular observed phenomenon. One submethod of case studies is performing interviews with experts who are involved in and knowledgeable about the phenomenon of study. Since those working in the field of study very likely have valuable practical experience related to the phenomenon of study, the resulting of performing interviews should be additional knowledge about the phenomenon.

#### 3.1.2 Experimental research

Experimental research methods are generally performed by inspecting the resulting dependent variable after altering an independent variable [18][p. 232]. In general research, there are many possibilities in designing the experimental setup and performing the experiments. The validity of research results where experimental research has been conducted depends on the experimental setup design and experiment execution, since it is not possible to prove “cause-and-effect” relationships between the independent and dependent variable when the study is not controlled [18][p. 233].

#### 3.1.3 Literature study

Literature reviews are generally performed by searching through available databases for publications relevant to the phenomenon of study, in most cases in the form of academic papers and books. There are several benefits of performing a literature review, such as: i) being able to identify what research has already been performed and therefore avoid replication, ii) being able to identify what research is lacking or appearing to be inadequate, iii) being able to correlate or combine research that already has been performed with own research to gain additional knowledge without needing to do work that has already been performed [19][p. 1].

In order to perform a literature review, we follow a slightly modified version of the steps presented by Onwuegbuzie et al. [19][p. 2]:

1. Define a list of keywords to be used as search terms, based on the research questions.
2. Perform searches in scholarly literature databases provided through the search engines Google

Scholar [20], ScienceDirect [21], IEEE Xplore [22], ACM [23], SpringerLink [24];

3. Retrieve and skim read the resulting literature, to determine if they are relevant. In case it has no relevance, it is discarded.
4. Relevant literature is subject to fine-reading and a summary of it is written. In case any results or statements in the relevant literature appear to be of inadequate quality or otherwise of questionable validity, this is considered particularly important to note and, if possible, attempt to replicate.
5. Retrieve the literature cited by each of the identified relevant academic papers, and repeat step 3, 4 and 5 until the list of relevant literature appears to be exhausted.

### **3.2 RQ1: What is the forensic value of OOXML documents, and how can they be used in forensic investigations?**

This research question can to some extent be answered by performing a literature study, since OOXML documents share some similarities with other types of files, with respect to the types of forensically interesting information they record. Published research has identified what types of information in general is desirable for forensic investigators, and what the purpose of each type of information is in the context of digital forensics [3]. Furthermore, some of the research performed on OOXML documents has already identified certain possible use case scenarios [9, 5].

In order to determine what forensically interesting information is stored in OOXML documents, a combination of experimental research and study of the OOXML standard is utilized. This is in practice performed by dissecting sample OOXML documents which are self-generated or collected on the Internet, and relate the findings to the formal descriptions found in the standard.

We have chosen a qualitative approach to gain supporting knowledge used to answer this research question, in particular in the form of case studies including conducting interviews with experts working in the field. Forensic investigators working in the field are bound to have valuable “real-world” experience, and should therefore be able to provide useful knowledge about typical scenarios where the information in OOXML documents could be utilized. As supporting knowledge to answer this research question, feedback provided by forensic investigators in two law enforcement agencies in Norway is utilized, namely NCIS Norway (Norwegian: Kripos) and National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway (Norwegian: ØKOKRIM).

### **3.3 RQ2: Can the metadata of OOXML document be trusted?**

The purpose of this research question is to determine whether or not certain evidence should be trusted, which is highly important particularly in forensic investigations where the evidence is presented to a court of law.

In order to attempt answering this research question, experimental research methods are the primary sources of information. Uncertainties or irregularities should be possible to identify while

inspecting self-produced documents and documents in the data set of document collected from web sources. We attempt to determine whether falsifying evidence in OOXML documents is a trivial task, and if it is, whether it can be detected. This is relevant for both a court of law and forensic investigators [8][Appendix D].

By inspecting self-produced documents and the data set of documents collected on web sources, uncertainties or irregularities should be possible to identify. We furthermore attempt to provoke or recreate situations that could lead to irregularities, in order to determine what situations could produce the irregularities.

### **3.4 RQ3: Are there differences from version to version of the popular office suites, with respect to what forensically interesting data they record in the files? Does performing certain actions in different ways affect the recorded forensically interesting data?**

Published research has briefly touched upon the fact that different office suites may vary with respect to what forensically interesting information they store [9][p. 2]. This motivates further research on the forensic difference between different office suites supporting the OOXML standard, attempting to determine if some office suites record more or less forensically interesting information.

In order to answer this research question, experimental research techniques are utilized. The experiment setup and execution used to attempt answering this research question is in practise be performed by installing various office suites supporting OOXML, and using each office suite to perform a pre-defined set of actions that could affect any forensically interesting information recorded in the documents. Since there is a possibility that combining several actions might produce different results, each type of action is isolated to one document to avoid false “cause-and-effect” results. After performing the actions for each office suite, the resulting document are subject to inspection and comparison.

### **3.5 RQ4: In what ways can the revision identifiers be useful in a forensic investigation, and in what situations are they preserved?**

Published research has touched upon the use of revision identifiers in a forensic investigation [9, 2], and to some degree also in what situations the revision identifiers are preserved when copying and pasting [5]. A literature study can therefore to some degree help answer this research question, in addition to motivating further research.

Experimental research methods are utilized to determine what situations preserve the revision identifiers when copying and pasting, and is in practice performed by altering an independent variable (i.e. the situation, e.g. changing the style of the text), copying content from one document to another and inspecting the XML of the resulting documents. This setup is used in several experiments, with altering the independent variable as the only difference.

## 4 OOXML file characteristics and use in digital forensics

This chapter presents the characteristics of OOXML files, and discusses the possibilities the information stored in the files may have in the context of digital forensics. The information contained in OOXML files can be used for different purposes in various types of investigations, and some practical scenarios where the information can be utilized are presented in this chapter.

### 4.1 History of the OOXML file format

While the Office Open XML format was released in 2006, XML as a format was used as early as in Office 2000 (beta version released in September 1998) for certain functionality, such as metadata and vector markup. The beta version of Office XP (released in August 2000) supported XML as the format for storage of Excel (spreadsheets) files, using their format named *spreadsheetML*. The beta version of Office 2003 (released in October 2002) supported XML as a format for Word files, using their format named *wordprocessingML*. In May 2005, Microsoft announced that the new XML format would be default for Word, Excel and PowerPoint [25].

In November 2006, Microsoft Office 2007 was released, with full support for OOXML. In December the same year, ECMA approved OOXML as ECMA standard 376, and ECMA submitted OOXML to ISO/IEC<sup>1</sup> for approval as an ISO/IEC standard. This was approved in April 2008, and was published as ISO/IEC DIS 29500 [25][26]. The first version of ECMA-376 and ISO/IEC DIS 29500 are almost identical [27], and this thesis refers to ECMA-376 instead of ISO/IEC DIS 29500 since ECMA-376 is provided without cost [1].

The standardization was not uncontroversial: Some argued that it was “insufficient and unnecessary”, not fulfilling the criterias expected of an international standard [28], that the format was “designed by Microsoft for Microsoft products” [29], and that the format was bound to contain many errors due because of the size of the standard specification<sup>2</sup> and the time limit the reviewers were put on [30].

### 4.2 The OOXML package and file structure

Versions of Microsoft Office before Office 2007 used a proprietary, binary format for storing all the information associated with documents, including all its text, images and metadata. In order to extract information from these formats, the host program is required to identify the structures of interest and interpret their associated hexadecimal numbers [31]. Since the data structures are returned as hexadecimal numbers, the information of interest is not easily understandable for humans.

---

<sup>1</sup>International Standards Organization (ISO) and International Electrotechnical Commission (IEC).

<sup>2</sup>Over 6000 pages.

As opposed to the previous binary formats, all information belonging to an OOXML file is stored as regular ZIP file [32][p. 17], which is extractable as any other ZIP file. This ZIP file is referred to as a “package” in ECMA-376, and is defined as a container used for storing a collection of parts [4][p. 14], where a “part” is defined as a stream of bytes with MIME content type, usually referring to a file in a file system, a compound file stream or a HTTP URI [4][p. 6]. The intention of utilizing a package to store all the information associated with an OOXML file, is to provide the convenience associated with only needing to deal with a single file when distributing it [4][p. 14]. Table 3 provides the file structure of an extracted sample OOXML document, along with a short description of each file.

#### 4.2.1 Metadata stored in OOXML documents

OOXML documents contain two XML files in particular that contains metadata that will often be of interest to forensic investigators, known as *app.xml* and *core.xml*, located in the *docProps* directory. It should be noted that since this metadata is stored in XML files in the OOXML container, this notion of metadata is different from the notion of metadata that is commonly used in digital forensics, which in normal terms would refer to the metadata of the OOXML container itself. In this case, the difference is that the metadata stored in the XML files are normally recorded by the word processor, and the metadata of the container itself are normally recorded by the operating system.

ECMA-376 [1][p. 4985] describes what types of metadata is stored in OOXML documents, and Figure 3 shows how the metadata of OOXML documents are categorized. The various types of metadata recorded in *docProps/app.xml* are listed in Table 5, and the various types of metadata recorded in *docProps/core.xml* are shown in Table 4. Appendix G.3.3 shows the content of *docProps/app.xml* of a sample document; Appendix G.3.4 shows *docProps/core.xml* of the same document. As the figure shows, word processor applications may also choose to record custom metadata in *docProps/custom.xml*.

The custom metadata could be an important source of information and should not be ignored: Our inspection of documents in the data set (see Section 6.1) determined recorded content such as file paths of content included in the documents, additional names not recorded elsewhere in the metadata, server names hosting content included in the document and additional timestamps. Appendix I provides the contents of *docProps/custom.xml* of a sample document collected from web sources. We note that all values have been changed for privacy reasons.

Table 4: Metadata recorded in *docProps/core.xml*, adapted from [1][p. 4985 - 4986], [4][p. 41]

Element	Description
category	Category the document belongs to, e.g. “resume” [4][p. 41]

contentStatus	Current status of the document's content, e.g. "draft" [4][p. 41]
Created	Document creation date [4][p. 41]
Creator	Name of entity (e.g. person) creating the document
description	Description of the document's content [4][p. 41]
identifier	Identification reference to the resource, e.g. URI [4][p. 41][33]
keywords	Keywords used to support searching and indexing [4][p. 42]
language	Language of content of document [4][p. 42]
lastModifiedBy	The last user who modified the document [4][p. 42]
lastPrinted	Timestamp of last printing [4][p. 42]
modified	Timestamp of last modification [4][p. 42]
revision	Number of revisions performed on document [4][p. 42]
subject	Topic of document's content [4][p. 42]
title	Name of document [4][p. 42]
version	Version number of document [4][p. 42]

Table 5: Metadata recorded in docProps/app.xml, adapted from [1][p. 4986 - 4987]

Element	Description	Data type
Application	Name of the application that created the document [1][p. 3712]	string
AppVersion	Version number of the application that created the document [1][p. 3712]	string
Characters	Total number of characters in document [1][p. 3712]	integer
CharactersWithSpaces	Total number of characters in document, including spaces [1][p. 3713]	integer
Company	Name of the company associated with document [1][p. 3713]	string
DigSig	Digital signature of signed document [1][p. 3713]	DigSigBlob
DocSecurity	Security level of document, where 1 = password protected; 2 = recommended read-only; 4 = forced read-only; 8 = locked for annotation [1][p. 3713]	integer

HeadingPairs	Grouping of document parts and number of parts in each group [1][p. 3713]	VectorVariant
HLinks	List of hyperlinks in document at point of last save [1][p. 3714]	VectorVariant
HyperlinkBase	Base string for evaluating relative hyperlinks in document [1][p. 3714]	string
HyperlinksChanged	Specifies one or several hyperlinks were exclusively updated [1][p. 3714]	boolean
Lines	Total number of lines in document at point of last save [1][p. 3715]	integer
LinksUpToDate	Whether or not hyperlinks in document are updated [1][p. 3715]	boolean
Manager	Name of supervisor associated with document [1][p. 3715]	string
MMClips	Total number of multimedia clips present in document [1][p. 3715]	integer
Pages	Total number of pages in document [1][p. 3715]	integer
Paragraphs	Total number of paragraphs in document [1][p. 3715]	integer
Properties	Application-specific properties of file [1][p. 3716]	Properties
ScaleCrop	Whether or not a thumbnail of the document should be scaled to fit the display [1][p. 3716]	boolean
SharedDoc	Whether or not document is shared between multiple producers [1][p. 3716]	boolean
Template	Name of template used to create document [1][p. 3716]	string
TitlesOfParts	Titles of parts used to compose document [1][p. 3716]	VectorLpstr
TotalTime	Total time the document has been edited, denoted in minutes [1][p. 3717]	integer
Words	Total number of words in document at point of last save [1][p. 3717]	integer

#### 4.2.2 Logical structuring of content in OOXML documents

The textual content of a OOXML document is logically structured into different parts: *paragraph* elements, *run* elements and *text* elements, with the following relationship: A paragraph element contains one or more run elements, and a run element contains a text element. If the content is not textual, e.g. an inserted image or an embedded object, the text element is replaced by a *draw-*



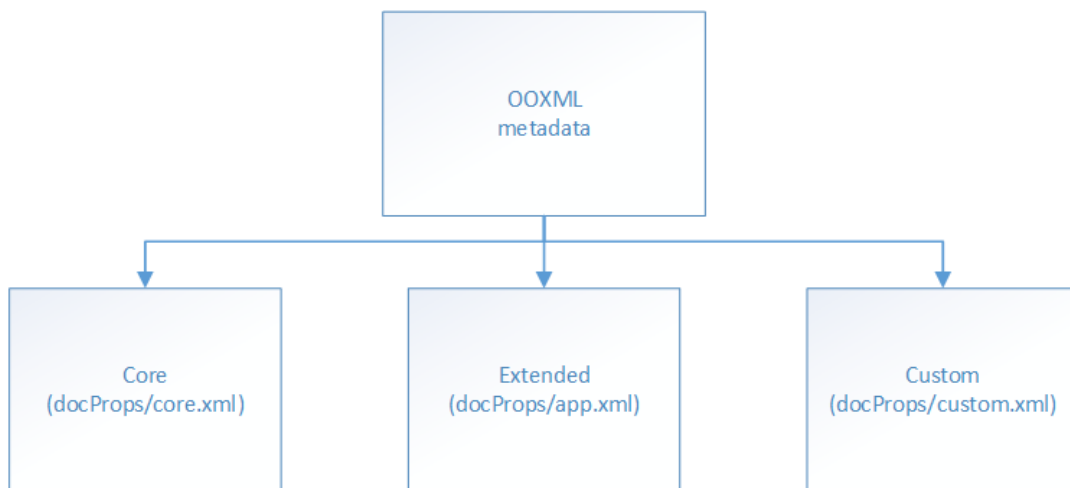


Figure 3: Metadata files in an OOXML document, adapted from [1][p. 4985]

ing or *object* element. A run is a logical container for representing content that share the same set of properties, such as boldface, underline and font color [1][p. 17]. Listing 4.1 provides sample XML showing a paragraph containing a run element and a run element containing a text element.

Listing 4.1: XML showing the logical structure of textual content in an OOXML document

```

<w:p>
  <w:r>
    <w:t>Showing some basic text</w:t>
  </w:r>
</w:p>
  
```

### 4.2.3 The concept of revision identifiers

While performing manual inspection of *word/document.xml* and *word/settings.xml* contained in OOXML document packages produced by Microsoft Word, a number of “revision identifiers” will appear as content properties in both files. Revision identifiers are 32-bit numbers represented in hexadecimal, with the intended purpose of providing a more effective and accurate way of merging two documents that origin from the same source [34]. Figure 4 displays an example of merging two documents that origin from the same source, where the revised document contain 4 insertions and 1 deletion. Appendix G.3.5 shows the *word/document.xml* of a sample document; Appendix G.3.6 shows *word/settings.xml* of the same document.

The concept of revision identifiers makes it possible to determine exactly what changes have been performed in a document, if both a revised version and an older version of the document is available. This functionality is similar to when change tracking is activated in a document, with the main differences being that change tracking provides some more information, e.g. who performed the changes and at what time, and that changes are attributed to content in the XML

of the document itself, which removes the need of having a reference document for comparison in order to determine what changes has been performed.

One of the reasons for utilizing revision identifiers instead of change tracking to determine what changes been made in a document, is to preserve the privacy of those editing the document [34]. From a privacy perspective, it might in many cases not be desirable to have changes attributed to a certain individual with an accompanied timestamp, e.g. because certain parts of the document is supposed to be contributed to anonymously and should not be connected to a certain person. Regardless, it might still be desirable to merge or compare two documents and easily determine their difference, and revision identifiers provides a compromise; privacy is protected to a higher degree, while some details are lost compared to what would be recorded if change tracking was enabled.

The office suites makes use of the revision identifiers by considering content from two documents sharing identical revision identifiers to be produced before the documents were forked into two separate documents, and the content with revision identifiers unique for each document to be produced after the documents were forked [34]. Since utilizing revision identifiers provides more details of what edits have performed between the original and a revised document, comparing and merging documents therefore provides more accuracy without the need for “full” change tracking.

The screenshot displays a document editor interface. On the left, a 'Revisions' sidebar shows a list of 5 revisions: 4 insertions, 1 deletion, 0 moves, 0 formatting, and 0 comments. The main area shows a 'Combined Document' window with the title 'Employee of the month suggestion' by Alice, R&D dep. The text reads: 'My suggestion is that MalloryBob should receive the employee of the month award. He is an outstanding colleague, and everybody in the entire company should note his extremely high morale and team working skills.' Below this, two windows are shown side-by-side: 'Original Document (first.docx - Alice)' and 'Revised Document (first - Copy.docx - Mallory)'. The original document shows the text: 'My suggestion is that Bob should receive the employee of the month award. He is an outstanding colleague, and everybody in the company should note his high morale.' The revised document shows the text: 'My suggestion is that Mallory should receive the employee of the month award. He is an outstanding colleague, and everybody in the entire company should note his extremely high morale and team working skills.'

Figure 4: Merging two sample documents.

There are 7 different uniquely named types of revision identifiers in OOXML documents, each serving specific purposes. Some of the revision identifiers share the same name, but their interpretation is context dependent. Table 6 provides an overview of the different types of revision identifiers. Common for all types of revision identifiers, is that ECMA-376 defines that their purpose is to provide the possibility of separating specific revisions, and that all content associated with the same unique revision belongs to the same editing session in a document, where “editing

session” in this context refers to the time span between each save [1][p. 1049].

Table 6: Types of revision identifiers in OOXML documents.

Name	Context	Description
rsidRPr	Paragraph	“Revision identifier for paragraph glyph formatting”, used to determine in which editing session the glyph character for the paragraph mark associated with the paragraph was modified [1][p. 237]
rsidP	Paragraph	“Revision identifier for paragraph properties”, used to determine in which editing session the associated paragraph’s properties were modified [1][p. 236]
rsidDel	Paragraph	“Revision identifier for paragraph deletion”, used to determine in which editing session the associated paragraph was deleted [1][p. 236]
rsidRDefault	Paragraph	“Default revision identifier for runs”, used by all runs in the associated paragraph that do not have declared an rsidR attribute [1][p. 237]
rsidR	Paragraph	“Revision identifier for run”, used to determine in which editing session the associated run was added to the document [1][p. 291]
rsidRPr	Run	“Revision identifier for run properties”, used to determine in which editing session the associated run’s properties were modified [1][p. 292]
rsidDel	Run	“Revision identifier for run deletion”, used to determine in which editing session the associated run was deleted [1][p. 291]
rsidR	Run	“Revision identifier for run”, used to determine in which editing session the associated run was added [1][p. 291]
rsidDel	Table row	“Revision identifier for table row deletion”, used to determine in which editing session the associated row was deleted [1][p. 464]
rsidR	Table row	“Revision identifier for table row”, used to determine in which editing session the associated row was added [1][p. 464]
rsidRPr	Table row	“Revision identifier for table row glyph formatting”, used to determine in which editing session the glyph character for the table row mark was modified [1][p. 465]
rsidTr	Table row	“Revision identifier for table row properties”, used to determine in which editing session the associated table row’s properties were modified [1][p. 465]

rsidDel	Section properties	“Section deletion revision identifier”, used to determine in which editing session the section mark for the associated section was deleted [1][p. 589, 591-592]
rsidR	Section properties	“Section addition revision identifier”, used to determine in which editing session the section mark was added to the associated section [1][p. 589, 591-592]
rsidRPr	Section properties	“Physical section mark character revision identifier”, used to determine in which editing session the character representing the section mark was formatted [1][p. 589, 591]
rsidSect	Section properties	“Section properties revision identifier”, used to determine in which editing session the character representing the section was formatted [1][p. 590, 591]

As Table 6 shows, the 7 uniquely named types of revision identifiers must be interpreted based on their context (with *rsidTr* and *rsidSect* as exceptions since they are only used in one specific context each and therefore unambiguous); depending on if they are associated with a *paragraph*, *run*, *table row*, or *section properties*. Although all types of revision identifiers may be of interest in the context of digital forensics, we argue that *paragraph rsidR* and *run rsidRPr* are particularly interesting. *Paragraph rsidR*, i.e. the value of *rsidR* in the context of paragraphs, are attributed to paragraphs when they are added, and may be utilized in visualization of paragraph creation as presented in Section 2.1.5. The *run rsidRPr* attribute is important since it is required for revision identifiers to be preserved in content copying, as will be presented in Section 6.3.3.

#### 4.2.4 Metadata of embedded objects

As presented by Garfinkel et al., objects that are embedded in OOXML files are stored in the resulting ZIP file as “their own parts” [9][p. 2]. This means that if e.g. another OOXML document, spreadsheet or presentation is embedded into an OOXML document, the file itself is stored in the file package of the receiving document.

This characteristic may have significant value in forensic investigations, since the metadata of the embedded objects are preserved in their original form. While a suspect may of course easily edit or remove the metadata of any embedded objects, it is a fair assumption that many perpetrators are non-technical people and will not attempt to do so. It is also likely that many perpetrators simply are not aware of the fact that embedded files are stored in their original form and therefore contain possibly forensically interesting metadata.

### 4.3 The forensic usefulness of a single OOXML document’s metadata

Reiterating the essence of Section 2.1.2; Buchholz et al. identified six keywords for questions forensic investigators typically seek to get answered in an investigation: *Who*, *what*, *when*, *how*, *where* and *why* [3][p. 5]. We define a piece of evidence’s forensically usefulness to be dependant on the extent of answers it can provide, with respect to the keywords *who*, *what*, *when*, *how*, *where* and *why*. Therefore, the more answers a piece of evidence can provide, the more forensi-

cally interesting it is considered. The “why” question deals with people and their motives, and is therefore considered out of the scope of this thesis. In this section, we deal with the case of a single seized document, where reference documents are not available as additional supporting evidence.

### **Who is associated with the document?**

Relating the *who* keyword to OOXML metadata, it is desirable to connect the actions performed, e.g. editing the document, to a specific physical person or several people. This can in some cases be answered with the name of the person who initially created the document, which as shown in Table 4 can be extracted from the *creator* element of *docProps/core.xml*, and is normally automatically set by the word processor based on the name associated with the operating system user account used when creating the document. In the case of Office 365, the value of *creator* is set to the user’s email address if the user is authenticated with his Microsoft account. Furthermore, the value of *lastModifiedBy* could reveal the name of another person responsible for editing the document, if that other person was the last one to edit the document. Likewise, if a company name is specified in the word processor, it can be extracted from the *company* element in *docProps/app.xml*.

Correlation of evidence is often useful in a forensic investigation, and metadata that does not directly include names of the people might still in some cases be used to support the attribution of a document to a person or a group of people if enough evidence is present. As presented in Section 6.4.3, inserted images with their original paths preserved in the document’s metadata could have significant value in attributing a document to a person, since preserved paths could include a person’s name or username, in addition to other information such as the user’s operating system. Furthermore, if a preserved path extracted from a seized document to some degree is unique and identical to a path existing on the computer belonging to the suspect, this could support the suspicion against the suspect. Office 2013 has functionality allowing the user to insert images from his Facebook account after successfully authenticating, and the image’s original Facebook filename is preserved in the metadata of the document. This could have significant forensic value, since parts of the filename can be used to identify the user’s Facebook profile, as shown in Appendix C.

The following scenario provides a practical example of evidence correlation used to attribute a document to a physical person, even though the document’s metadata does not directly reveal any names or similar information.

#### **Scenario:**

*A bomb threat detailing the plans of the perpetrator is emailed to a company, attached to the email as a PDF file. The PDF file contains textual descriptions of a planned attack, along with an image displaying a map of the area the perpetrator describes as his target.*

#### **Forensic process:**

*Police investigators are able to extract the originating IP address the email appears to be sent from, and track it to a student housing near a university campus. The police raid the housing, but all of*

*the residents claim their innocence. Computers and other electronic devices are seized and imaged, and forensic analysis determines that a .docx file is present on one of the resident's computers, and has been attempted deleted. The document is subject to analysis, in which package extraction is performed and the XML is read. Analysis determines that the path of the inserted map image is quite unique and refers to a location that exists in the suspect's computer's filesystem, and can therefore likely be considered a strong indication that the suspect's computer was used to send the email.*

### **What has been done in the document?**

Going further on the list of keywords, forensic investigators often seek to gain knowledge about *what* has happened, in this context referring to what actions have been performed within the document's content, in the document structure or on the document container file. One type of information that could be used to determine what has been done in document, is the concept of revision identifiers. As presented in Section 4.2.3, content in an OOXML document gets an associated unique revision identifier making it possible to determine in what revision content was added. We can consider revisions as actions performed in a document, and the revision identifiers provide an indication of what has been done in a document.

The content of OOXML documents produced in Microsoft Word will as previously described get attributed with revision identifiers with a type depending on the context of the content. When a paragraph is added to a document, the content of the paragraph is associated with a revision identifier named *rsidR*. The value of this revision identifier is unique within the document, and all content sharing the same revision identifier value is by definition added within the same editing session, i.e. the interval of time between two saves. The revision identifiers can therefore reveal some information about what has happened in the document.

In his paper, Langweg utilized visualization techniques to determine what had been done in the document he analysed [2]. The visualizations were based on extracted revision identifiers, and was used as supporting information to determine that the document was likely compiled from other documents. A similar type of visualization was used to detect language anomalies in the document, which was used to determine that the suspect at one point likely edited the document on a computer with a different language setup, likely in another country. Visualization techniques can therefore be useful to determine what has happened in a document, since they provide a representation of information that would be difficult to understand just by performing manual analysis. Figure 5 provides an example visualization, representing the language attributes detected in a sample document.

Thumbnails of documents can be forensically interesting, and can in some cases give an indication of what has been happening in a document. As presented in Section 6.4.2, thumbnails of documents are by default not saved by any of the tested word processors. However, once a document has been saved with the "Save thumbnail" box checked, all subsequent documents edited with the word processor will by default save a thumbnail until it is disabled. Garfinkel et

<sup>2</sup>The full version of the paper; unpublished.



Figure 5: Barcode visualization of language settings throughout the document, from the complete version of [2]

al. identified two possible uses of a thumbnail [9][p. 3]:

1. If comparing the thumbnail and the actual contents of the document results in a mismatch, the thumbnail or the content might have been altered after the thumbnail was generated;
2. If the document is damaged but the thumbnail is intact or restorable, it could provide some indication of what the document contained.

The recorded editing time, as reflected by *TotalTime* in *docProps/core.xml*, can in some cases provide indications of what has been done in a document. If the recorded editing time shows an unrealistic low number of minutes compared to the amount of content in the document, this could be an indication that the document was composed based on other sources, which could be interesting in forensic investigations involving theft of intellectual property. Since the document could be assumed to have been composed based on other sources, it could indicate that the investigators have not been able to seize all evidence if the sources have not been located.

The following scenario provides an example of a case where determining what has happened in a document could be useful in a forensic investigation.

**Scenario:**

*A large-scale drug dealer is apprehended and his location is raided, and the investigators seek to uncover his customers.*

**Forensic process:**

*The investigators seize all units found at the location, and create forensic images each unit. Upon inspection, the investigators discover a deleted document, *debt.docx*, which is partly overwritten. They manage to assemble and extract the package, but the document body is unrecoverable and cannot be read. However, they are able to partly recover a thumbnail of the first page found in the package. This thumbnail reveals names, purchases and debt of some of the suspect's customers, which is the type of information the investigators were looking for.*

**When did the actions performed in the document occur?**

Information about what point in time an action was performed is generally considered one of the most important pieces of information in a forensic investigation, since time information usually is necessary for correlating evidence [35][p. 1]. After an incident has occurred, the actions relevant to the incident are bound to have happened at a specific point in time, and one of the tasks of forensic investigators is to attempt reconstructing the events and place them on a chronological

timeline.

Assuming the events and their associated time information are correct and correctly placed on the timeline, investigators are able to use the timeline for correlating evidence from different sources. Investigators can for example correlate file timestamps with time information related to events claimed in interrogations, such the suspect claiming to be physically present at another place while the timestamps associated with a file determine that somebody was present at the computer at the specified time [10][Appendix E].

As Table 4 shows, OOXML metadata usually contain three timestamps:

- Timestamp representing the point in time the document was created;
- Timestamp representing the point in time the document was last modified;
- Timestamp representing the point in time document was last printed.

These timestamps could be useful in e.g. timeline creation and evidence correlation as discussed above, but it must be noted that timestamps should not blindly be trusted since it is always possible that the clock of the suspect's machine was incorrect when the document was created, edited or last printed. In addition, as will be discussed in Section 4.6, the *created* timestamp of OOXML documents created with Word Online should not be trusted in a forensic investigation.

In addition to the timestamps available in the seized OOXML document metadata itself, timestamps can be recursively extracted from embedded OOXML spreadsheets, presentations and documents if they have been embedded. When an OOXML document is embedded in another OOXML document, it is stored in its original, unaltered form if it is inserted by using the *Insert -> Object* functionality of Microsoft Word. If *Insert -> Text from file* is used, only text extraction is performed and the original OOXML document is not embedded. When OOXML spreadsheets and presentations are embedded into an OOXML document, their main XML bodies (*ppt/presentation.xml* for presentations; *xl/workbook.xml* for spreadsheets) are slightly modified: For presentations, an *extLst*<sup>3</sup> tag including a GUID is added; for spreadsheets, an *oleSize*<sup>4</sup> is added. This does not affect the possibility of extracting the embedded objects' metadata; *docProps/core.xml* and *docProps/app.xml* are kept in their original, unaltered form.

**Scenario:**

*A homicide has been committed. During an interrogation, the currently main suspect claims his innocence and claims that he was at home writing a letter at the time the homicide was committed, and that he could not possibly have been physically present at the location of the crime scene.*

**Forensic process:**

*The suspect's computer is seized, and a forensic image is created. The forensic image is indexed, and all documents are extracted from the system. The described document is located, and metadata*

<sup>3</sup>Extension list [36].

<sup>4</sup>Embedded object size [37].



extraction is performed. Metadata from `docProps/core.xml` shows a last modified timestamp which is consistent with the claims of the suspect; the document appears to have been last modified at a point in time that would make it practically impossible for the suspect to be present at the crime scene at the time of the homicide due to the amount of time travelling there takes. Timestamps could in this case be used to support the case of the suspect if other evidence also adds up, and the investigators could focus their attention on other possible suspects.

Revision identifiers and their associated content can in some cases reveal some time information if supported by additional information: If an editing session includes content that mentions or describes events, the earliest possible date the editing session took place must be the same as the described event. Likewise, content with revision identifier number higher than the revision identifiers associated with the content describing the events must have been written or edited at a point in time after the described event occurred, since revision identifiers in Microsoft Office are incremental within the document [1][p. 1049].

Figure 6 shows a barcode visualization of the sample document shown in Appendix G.3. Each colorized box represents the amount of *paragraph creation revisions* associated with the revision identifier specified under, in proportion to the total amount of *paragraph creation revisions* in the document. The colorized box to the left represents the paragraph creation revisions in the very beginning of the document, while the colorized box to the right represents the very last of the document. Content belonging to boxes appearing to the left of a chosen box is therefore written before the content of the chosen box.

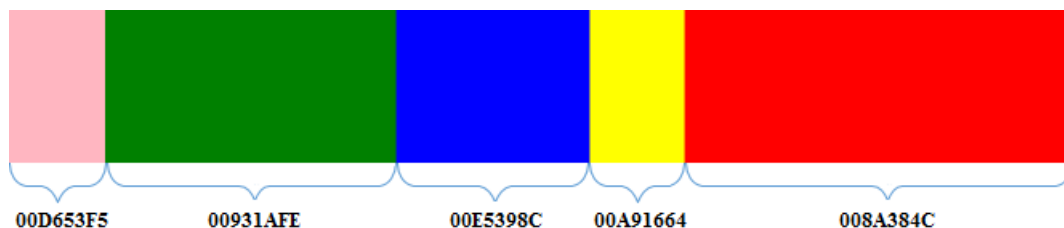


Figure 6: Barcode visualization of paragraph creation revisions of document in Appendix G.3

### How were the actions done in the document performed?

Evidence that could provide an indication of how actions were performed in a document could be important in a forensic investigation, since correlating evidence might reveal other information such as who actually performed the editing [3][p. 9]. Performing evidence correlation can, as previously discussed, provide further knowledge about the events that have taken place in a case, even though it might at first not be apparent. The following scenario provides an example to demonstrate how determining how actions were performed in a document could be used to determine who wrote the document.

#### Scenario:

*A publicly traded company is under investigation for leaking inside information, giving certain investors an unfair advantage compared to other investors (which is known as insider trading*

[38]). Since it is unknown who leaked the information, the goal is to determine who did it to thereafter be able to prosecute the person.

#### **Forensic process:**

*Based on details about information leaked, the investigators suspect that the information was leaked by somebody in a management position, and the number of suspects is thereby reduced. The investigators seize and create forensic images of the computers and other units of those in management positions, and perform analysis of these to look for evidence of the alleged actions. On a USB flash drive found in a trash bin, they find an OOXML document which has been attempted deleted, containing information very similar to what has been leaked.*

*Upon inspecting the document's metadata, it appears that the metadata has been stripped, removing information such as "Creator" and "lastModifiedBy". The "created" and "modified" timestamps, have, however, not been affected by the metadata cleansing process, in addition to "Application" and "AppVersion" still being intact. The application-specific information shows that Microsoft Office with the "AppVersion" 00.0001 was used to create the document, which means it was created with Word Online. This motivates inspection of the web browser cache of each seized system. The inspection reveals that one of the systems accessed Word Online at a time consistent with the timestamps recorded in the document found on the USB flash drive, which leads to strong suspicion against the person owning the computer.*

#### **Where were the actions performed in the document performed?**

Determining where the actions associated with the document were performed could be important to e.g. attribute the document to a specific person or a group of people, particularly if the document's metadata does not have specified any author-specific helpful information such as *creator* or *lastModifiedBy* that could be used to identify the person. It could also be used to help prove that the suspect has been at a certain location. In this context, "where" refers to both locations within computer systems, such as if a file was created locally or originates from an external source such as a USB flash drive, and the physical location of the people associated with the document.

Preserved original paths of inserted images could also in some cases be used to answer the "where" question, since it could reveal that the inserted image originated from e.g. a USB flash drive or an external hard disk drive. This information could have significant value in a forensic investigation, since it shows that a physical unit at some point likely has been present. This information could be very useful since it could indicate that the investigators might not have seized all possible evidence, for example a hidden USB flash drive, which could contain additional forensically interesting evidence [10][Appendix E].

Microsoft Office automatically detects the input language based on the content that is typed, in order to provide spell checking for that particular language. By default, the language is set to the language of the installed version of Office, and the language will automatically change after some content has been typed in another language. The following scenario is provided to demonstrate one way this could be used to determine where the person associated with the document has been.

**Scenario:**

An employee of a company in London is investigated for selling highly sensitive customer information to a criminal organization in Stockholm after a member of the criminal organization is apprehended. In an interrogation, the apprehended criminal confesses and claims that he physically met with the employee in Stockholm to exchange the contraband information and money. The employee denies everything, and claims he was at home that day, fixing his motorcycle and writing an entry in his diary.

**Forensic process:**

Police investigators seize and create a forensic image of the employee's laptop. Upon inspection, they find the mentioned diary written in Microsoft Office and stored as an OOXML document. They further find the diary entry the suspect referred to, which is shown in Figure 7. Upon inspection of the XML structure of word/document.xml, they discover that the headline, 12/03-2014, is attributed with `<w:lang w:val="sv-SE"/>`, as shown in Listing 4.2, while the rest of the document is attributed with `<w:lang w:val="en-GB"/>`. This indicates that the document at some point has been edited on a computer with Swedish set as the default language, which could e.g. have been the suspect visiting an Internet café or a library while in Stockholm.

12/03-2014

London is beautiful today. For once, sun is shining and I have not seen one cloud in the sky. Today, I took a walk around the neighbourhood just to get some fresh air for once. Work has been very stressful lately, and I cannot get my motorcycle to start.

Figure 7: Diary entry of the suspect in the scenario.

Listing 4.2: XML showing a Swedish language attribute to the headline of the diary.

```
<w:r w:rsidRPr="00034F84">
  <w:rPr>
    <w:lang w:val="sv-SE"/>
  </w:rPr>
  <w:t>12/03-2014</w:t>
</w:r>
```

#### 4.4 When change tracking is enabled

Microsoft Office has functionality to track all changes in a document, which is intended to be used when collaborating on a document, e.g. when a document is written by one person and reviewed by somebody else who suggests that certain parts of the document should be edited [39]. A document with change tracking enabled records the names of every author associated with the performing each change, a timestamp of when each edit occurred, in addition to revision identifiers associated with the editing.

A seized document with change tracking enabled therefore provides a wealth of information to a forensic investigator who attempts to determine what has been done in a document, by

whom and at what time; in other words useful in answering “who”, “when” and “what”. Appendix H.1 provides a screenshot of a document with change tracking enabled, and Appendix H.2 shows the underlying XML of performing one of the edits of that document. From a forensic perspective, change tracking is unfortunately disabled by default, but might still likely be found enabled in some forensic investigations.

## 4.5 Forensic usefulness of OOXML documents with reference documents

This section deals with cases where a seized document is of interest to the forensic investigators, and where one or more reference documents are available. In this context, reference documents refers to any other OOXML documents containing revision identifiers that can be used in a comparison process.

### 4.5.1 Detecting unauthorized distribution of sensitive documents

Garfinkel et al. [9][p. 5] briefly present the idea of creating a database consisting of revision identifiers that have been extracted from documents gathered from an organization under investigation. Once this database has been created, the revision identifiers extracted from documents appearing in the investigation at a later point can then be compared to the revision identifiers in the database.

The construction of an initial database consisting of documents and their corresponding extracted revision identifiers can be performed automatically by using a custom-made extraction process, such as the one presented in Section 5.2.3. The collection of documents at a later point of time could be performed automatically, and the revision identifier extraction could be performed in manner such as the one displayed in Figure 8.

In the case where one or more revision identifiers extracted from a collected document are identical to any of those recorded in the database, the collected document is likely the same document as the one matched in the database or contains content copied from the document in the database under the circumstances identified in Section 6.3.3. As it is likely that the collected document contains content that is considered sensitive, the tool could produce an alert which notifies the investigators of a suspicious document found being distributed. The suspicious document could e.g. be gathered by capturing it while it is being transferred over a network that is under active surveillance, or extracted from a captured system such as the hard drive of a seized computer [9][p. 5].

Using the first example of a method for collecting suspicious documents to be checked against the database, it is possible to implement such solution by using e.g. *tcpflow* for capturing the TCP packets sent and received on a network. *tcpflow* is a tool that captures TCP data packets on a network, and stores each TCP flow as an individual file [40]. As an example, a network monitored by *tcpflow* would capture a document being uploaded to an FTP server and save it unaltered as a separate file, as long as the document and the connection are not encrypted. The process checking for matches could monitor the cache directory for new documents and automatically

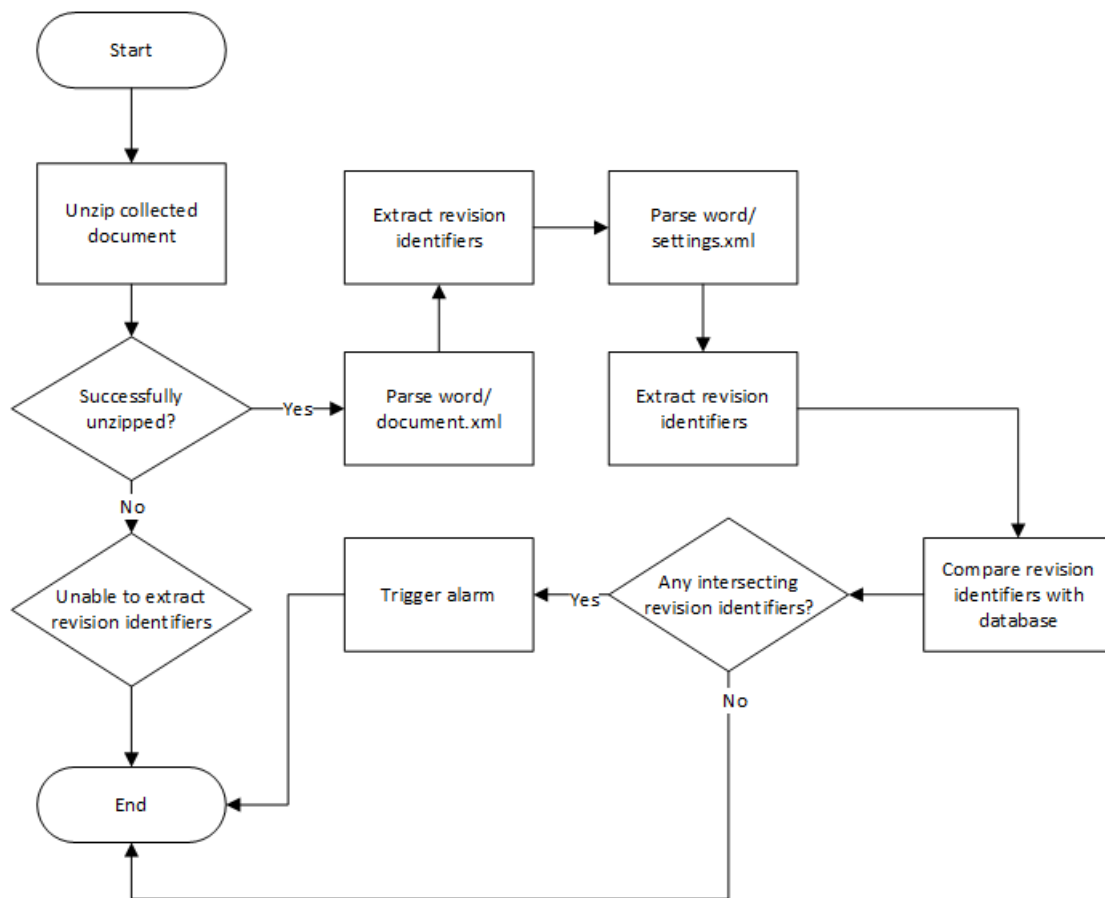


Figure 8: Flowchart showing a process of checking a collected document against database of known sensitive documents.

perform extraction and comparison.

This implementation has the benefit that the database of known sensitive documents does not need to store the actual content of the documents, since only the revision identifiers are compared. Since a database containing sensitive documents would be a prime target for attackers, the risk associated with a security breach is greatly reduced when only the revision identifiers and other non-sensitive metadata is stored.

#### 4.5.2 Attributing a document to a person

Attributing a document to a person could be relevant in many types of forensic investigations, as the document's content could e.g. contain evidence that could be used to make accusations in the investigation or in a court of law. Documents that are part of a forensic investigation might not always easily be attributed to a specific person if information such as creator name, company or preserved original file paths of images are not specified in the document or appear to be ambiguous. Such cases might be the result of intentional metadata removal after the document has

been created, or a ambiguous creator name specified in the operating system, e.g. *User*, which could be hard to attribute to a specific person.

Based on revision identifiers extracted from the seized document and a reference document, it might in certain cases be possible to attribute the document to a person even though the meta-data of the seized document cannot be used to identify the document's creator. In order for this to be possible, we have identified the following requirements:

- A reference document must be available;
- Some identifiable author information must specified in the reference document;
- The documents must have some intersecting revision identifiers attributed to content which there is reason to believe the creator of the reference document wrote.

OOXML documents created with Microsoft Office have a *rsidRoot* element specified in *word/settings.xml*, and ECMA-376 specifies that “*This element specifies the revision save ID which was associated with the first editing session for this document. [...] This information must be identical between any number of copies of the same document, as they all originate from the same original editing session*” [1][p. 1051]. Our experiments have determined that if several instances of Word are open when creating documents (i.e. saving a newly created document), all documents get the same *rsidRoot* value.

These two situations could be both beneficial and disadvantageous in a forensic investigation. The advantage is that since two documents with intersecting *rsidRoot* values come from the same source, it could be possible to attribute a document to a specific person if the reference document has identifiable author information specified. The disadvantage is that it is not possible to determine if documents have intersecting *rsidRoot* values due to their being created on the same machine simultaneously or being the result of a forked document. Without other supporting information, it is therefore not possible to prove that two documents with intersecting *rsidRoot* values were written simultaneously on the same machine. The following scenario is provided to show a practical example of how this information can be used.

**Scenario:**

*The employees of a company are under internal investigation for misconduct after business secrets are leaked to a competitor. A ripped page of a printed sensitive document is found in the trash in the copy room. Investigators are able to retrieve the original OOXML document from the cache on the print server, and upon inspection discover that the perpetrator has stripped the metadata with Microsoft Office's metadata removal functionality.*

**Forensic process:**

*The investigators create forensic images of all units in the office. They extract all OOXML files from the forensic images, and extract the revision identifiers from all of them as well as from the document discovered on the print server, which is possible since revision identifiers are not removed when the metadata is removed with Office. The revision identifiers extracted from the documents are compared to those extracted from the document from the print server. One document from*

one of the forensic images is found to have a `rsidRoot` value which is equal to the `rsidRoot` in the document from the print server. This indicates that the two documents were created simultaneously and on the same machine. Metadata extraction of the document found on the forensic image shows that it was created by the owner of the machine. The investigators therefore have reason to believe the perpetrator was the owner of the machine where they located the document.

### 4.5.3 Uncovering previously unknown social networks

Being able to uncover previously unknown social networks could be useful to determine who is communicating with whom in e.g. a terrorist network. Extremists increasingly use the Internet for communicating with each other and spreading their radical ideas, according to the Norwegian Police Security Service: “Today we see that Internet and increased use of social media have created new platforms for radicalisation. [...] Contacts made in the virtual world may eventually form the basis for extreme cells which meet in the real world” [41]. Terrorists and other criminals might send OOXML documents to each other, and correlating identifiers from the documents could be useful in uncovering social networks that previously may have been unknown.

Since revision identifiers are unique, they can be used to uncover previously unknown social networks if documents are seized from several known sources and some intersecting revision identifiers are detected. Figure 9 shows a hypothetical social network consisting of people sending emails to each other, and the emails in this scenario have OOXML documents attached, containing extremist content. The figure shows that Person 1 sends document A to Person 2 and 3, and document B to Person 4 and 5. Person 5 sends document B to Person 6 and 7.

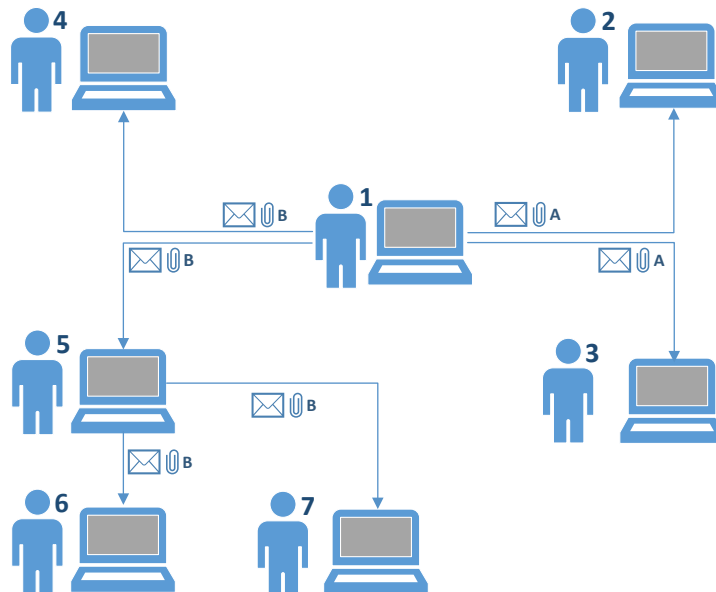


Figure 9: Hypothetical social network of people sending emails with attachments to each other.

In this scenario, the presented social network consists of extremists where we assume that some participants might perform or support acts of terrorism. We let the police apprehend Person 1

and seize his computer, create a forensic image of it and extract all OOXML documents from it. The police add all documents and their extracted revision identifiers to a database of documents seized from known “people of interest”, marked with the name of Person 1. Later, Person 6 is apprehended and the same extraction process is performed. The comparison process then detects intersecting revision identifiers from Document B found on the computers of both Person 1 and Person 6. Since the same information was found on both computers, it can be assumed that there is some connection between Person 1 and Person 6; they have likely communicated directly or belong to a network of people who have communicated at some point.

## 4.6 Trustworthiness of evidence found in OOXML documents

Being able to trust the evidence collected in forensic investigations is of the utmost importance, since uncertain, irregular or erroneous information might lead to forensic investigators drawing wrong conclusions. It is important that the evidence is sound in order for it to be accepted in a court of law; if its soundness is questionable, the evidence might be dismissed. Therefore, it is important to determine whether there is reason to believe certain evidence should not be trusted.

### 4.6.1 Alteration of XML metadata

In the context of uncertainties in the information contained in OOXML documents, one of the biggest potential challenges lies in the fact that OOXML documents are open, XML-based and the container is in practical terms just a regular ZIP package. It is therefore trivial to alter any information in an OOXML document, which makes it possible for an adversary to tamper with documents he knows or suspects will be collected in a forensic investigation. This possibility was verified by i) unzipping a sample document; ii) changing the metadata values in *docProps/app.xml*, *docProps/core.xml* and *word/document.xml*; iii) creating a ZIP archive of all files in package; iv) changing the file extension back to “.docx”. The document passes the validation process, and is interpreted properly by the word processor.

Depending on the intentions of the adversary, e.g. simply having the desire to cause problems for certain individuals or to draw the attention away from himself and his actions, there are several alterations he can perform that might lead to forensic investigators drawing wrong conclusions if his actions are not detected. The adversary might e.g. alter the document’s metadata and change the value of *Creator* or *lastModifiedBy* in *docProps/core.xml* to some other individual’s name, in order words performing evidence falsification.

If the adversary suspects that the forensic investigators will perform revision identifier extraction and comparison on the document of interest and a set of reference documents, he might change the values of one or more revision identifiers to match one or several reference documents, thereby making it seem like there is a relationship between the documents. Such evidence falsification might lead to innocent people being suspected for e.g. illegal copying, distribution of sensitive information, being a part of a criminal network and similar cases where revision identifiers are used.



## 4.6.2 Detecting alteration of XML metadata

Files contained in an extracted OOXML document package produced by Microsoft Office have set a timestamp to be *01.01.1980 00:00*, which marks the start of the FAT32 epoch [9][p. 5]. In the case of malicious manual alteration of the XML files in an OOXML package, such as changing a revision identifier value or the name of the author to produce false evidence, this can be detected by inspecting the modification timestamp of the files within the package. Listing 4.3 provides a sample extracted OOXML package, where *document.xml* has been manually altered. As shown, there is a discrepancy in *document.xml*'s associated modification timestamp, as it now reflects a real modification timestamp, *22.05.2014 22:03*, and not the default timestamp, *01.01.1980 00:00*.

Listing 4.3: Directory listing of *./word* in sample document where *document.xml* has been altered

---

22.05.2014	22:02	<DIR>		charts
22.05.2014	22:03		8 180	document.xml
22.05.2014	22:02	<DIR>		embeddings
01.01.1980	00:00		1 675	endnotes.xml
01.01.1980	00:00		1 261	fontTable.xml
01.01.1980	00:00		1 295	footer1.xml
01.01.1980	00:00		1 295	footer2.xml
01.01.1980	00:00		1 295	footer3.xml
01.01.1980	00:00		1 681	footnotes.xml
01.01.1980	00:00		1 295	header1.xml
01.01.1980	00:00		2 873	header2.xml
01.01.1980	00:00		1 295	header3.xml
22.05.2014	22:02	<DIR>		media
01.01.1980	00:00		3 023	settings.xml
01.01.1980	00:00		30 895	styles.xml
22.05.2014	22:02	<DIR>		theme
01.01.1980	00:00		497	webSettings.xml
22.05.2014	22:02	<DIR>		_rels

---

It must be noted that the modification timestamps of the files are preserved even after the file structure has been rezzipped and renamed to a *.docx* extension. Furthermore, our experimentation has determined that the modification timestamps are preserved even after the document has been modified in a word processor, saved and extracted again. This is highly valuable to a forensic investigator, as it provides a trivial method for detecting that manual alteration has occurred.

It must be noted that a determined person with the goal of falsifying evidence without leaving traces, could build his own office suite or a script that follows the requirements of ECMA-376 when creating documents. The output of this could be a document with falsified metadata, appearing to have been produced in Microsoft Office.

### 4.6.3 Discrepancies in recorded timestamps

Timestamps extracted from digital evidence are often essential parts of forensic investigations, making it possible for the investigators to e.g. create a timeline of events that have occurred in the case and correlate evidence. Timestamps are therefore used to support answering the “when” aspect related to the collected evidence in the particular case, as identified by Buchholz et al. [3][p. 8]. It is apparant that incorrect timestamps might lead to errornous conclusions, and it is therefore important to know whether or not the timestamps of the collected evidence are to be trusted.

#### **lastPrinted timestamp older then creation timestamp**

While performing manual inspection of the XML content of a certain document, we observed a possible discrepancy in the timestamps recorded in *docProps/core.xml*; a recorded *lastPrinted* timestamp appearing to be older than the recorded *Created* timestamp. Logically, this would of course be impossible as a document that has not been created yet cannot be printed. This motivated further analysis of the test data set (which is described in Section 6.1), where we attempted to determine if the *lastPrinted* timestamp discrepancy could be detected in other documents.

Analysis of the test data set showed that of the 76194 documents in the data set, 39535 (51.9%) a *lastPrinted* timestamp specified. Of these, 28688 (72.56%) documents had a *lastPrinted* timestamp older than the *created* timestamp; 868 (2.20%) had a *lastPrinted* timestamp equal to the *created* timestamp; 9979 (25.24%) had a *lastPrinted* timestamp later than the *created* timestamp.

One possible cause of *lastPrinted* timestamps being older than the *created* timestamp in a document, is when a document of the old Microsoft Word format (with the file extension *.DOC*) is edited in a version of Office supporting OOXML and resaved. This was confirmed by retrieving an old Microsoft Office document with a *lastPrinted* timestamp specified from the web, editing it with Microsoft Office 2013 and saving it as an OOXML document. We observed that the original *lastPrinted* timestamp is preserved in the new document, but the *created* timestamp is set to the time the document was saved as an OOXML document.

Investigators should pay attention *lastPrinted* timestamps being older than *created* timestamps, since this likely means that the document in reality is older than what the *created* timestamp reflects.

#### **Erroneous creation timestamp in Word Online**

Upon inspection of self-created documents made in Word Online, we observed that the *created* timestamp recorded in *docProps/core.xml* were obviously errorneous:

- *Document 1*: 2009-11-23T22:41:00.0000000Z
- *Document 2*: 2012-08-07T03:53:00.0000000Z

The true timestamps for creating the documents were the following:

- *Document 1*: 2014-05-02T10:44:17.2716589Z

- *Document 2*: 2014-05-29T15:58:35.7482626Z

We repeated the process of creating documents and inspecting the *created* timestamp in the metadata of each document over 3 days, and found that the erroneous timestamp *2012-08-07T03:53:00.0000000Z* reoccurred in each of the documents. Since these timestamps are obviously incorrect, this means that *created* timestamps from documents created in Word Online should not be trusted in a forensic investigation.

Table 3: File structure of extracted sample document.

File	Description
/	
_rels	
_rels	Contains package-level relationships; linking all package parts together [1][p. 29].
docProps	
app.xml	Extended file metadata properties of the document [1][p. 154].
core.xml	Core file metadata properties of the document [4][p. 41].
word	
_rels	
document.xml.rels	Contains part-level relationships; relationships between document and its associated parts [1][p. 30].
charts	
_rels	
chart1.xml.rels	Contains relationships between parts in chart.
chart1.xml	Chart used in the document [1][p. 127].
colors1.xml	Color information associated with chart [1][p. 130].
style1.xml	Style information associated with chart.
embeddings	
Microsoft_Excel	
_Worksheet1.xlsx	Embedded object (Excel spreadsheet) [1][p. 4994].
media	
image1.png	Image inserted into document [1][p. 157].
theme	
theme1.xml	Theme; color, font and format scheme [1][p. 135].
document.xml	The main document [1][p. 51].
endnotes.xml	Contains all Endnotes used in document [1][p. 37].
fontTable.xml	Contains every font used in document [1][p. 39].
footer1.xml	
footer2.xml	Footer displayed in document [1][p. 40].
footer3.xml	
footnotes.xml	Contains all footnotes used in document [1][p. 43].
header1.xml	
header2.xml	Header displayed in document [1][p. 48].
header3.xml	
settings.xml	All of the document's properties [1][p. 35].
styles.xml	Diagram style information [1][p. 134].
webSettings.xml	Web-specific settings used in document [1][p. 56].
[Content_Types].xml	Content types for relationship parts [1][p. 103].

## 5 OOXML Forensic Analysis Tool

### 5.1 Introduction

Currently available forensic tools supporting OOXML files have limited functionality in performing analysis of the documents [8, 10][Appendix D, E]. As described in Section 2.2, EnCase Forensic and FTK by default do not extract every type of metadata available in a OOXML document's *docProps/app.xml* and *docProps/core.xml*, although it is possible to extend EnCase's functionality by using custom scripts. Furthermore, neither EnCase nor Forensic Toolkit (FTK) by default provide any interpretation of the evidence available, leaving this job entirely up to the investigator. Neither of the two tools utilize revision identifiers from documents for any purpose.

We have developed a custom forensic tool prototype for analysing OOXML files, with the main purpose of exploring and demonstrating the possibilities OOXML files have in the context of digital forensics. In addition to exploring the analysis possibilities, the prototype was used to help answer the research questions. The prototype, simply named *OOXML Forensic Analysis Tool* or hereafter shortened *OOFAT*, has several functions that will be presented in this chapter.

The functionality of OOFAT was demonstrated to investigators from *the National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway* (Norwegian: *Økokrim*) and *NCIS Norway* (Norwegian: *Kripos*) in order to get feedback. The investigator from Økokrim responded that they perform OOXML document analysis manually because currently available forensic tools do not extract every type of information available [8][Appendix D]. The investigator from Kripos responded that they also perform analysis manually, which for them is feasible because they do not handle very large amounts of documents [10][Appendix E]. Both parties expected that at some point in the future there would likely be a case involving so many documents that performing analysis manually would be unfeasible, and that the functionality of OOFAT would be beneficial [8, 10][Appendix D, E]

The investigator from Kripos suggested that the revision identifier comparison process could be useful for e.g. *the Norwegian Police Security Service* (Norwegian: *Politiets Sikkerhetstjeneste*) in uncovering social networks to detect who has been communicating with whom, similar to what is presented in Section 4.5.3.

### 5.2 OOFAT's functionality

#### 5.2.1 Document validator

As will be presented in Section 6.1, a data set of independent OOXML documents was collected on the Internet through web searches. Since we have no control over these files, there is a possibility that they are corrupted or otherwise invalid, e.g. a file of another type with a false file

extension. A document validation process was therefore developed, with the purpose determining whether a file is a valid OOXML document. This process was also used to check if files were valid after performing the manual alteration described in Section 4.6. This was implemented by using the *OpenXmlValidator* method in the official OpenXML SDK from Microsoft, and each document input to the validation process is attempted validated as respectively Office 2007, 2010 and 2013 files.

Figure 10 shows the implemented process of determining whether or not a document is valid. *WordprocessingDocument.Open* uses the specified document path to create a new *WordprocessingDocument* instance [42]. *OpenXMLValidator* takes Office version 2007, 2010 or 2013 as a parameter, and uses the *Validate* method to attempt validation for the specified version of Office [43]. In the case where no errors are returned for the specified version of Office, the document is considered valid. In the contrary cases, where errors are returned for all version of Office, the document is considered invalid.

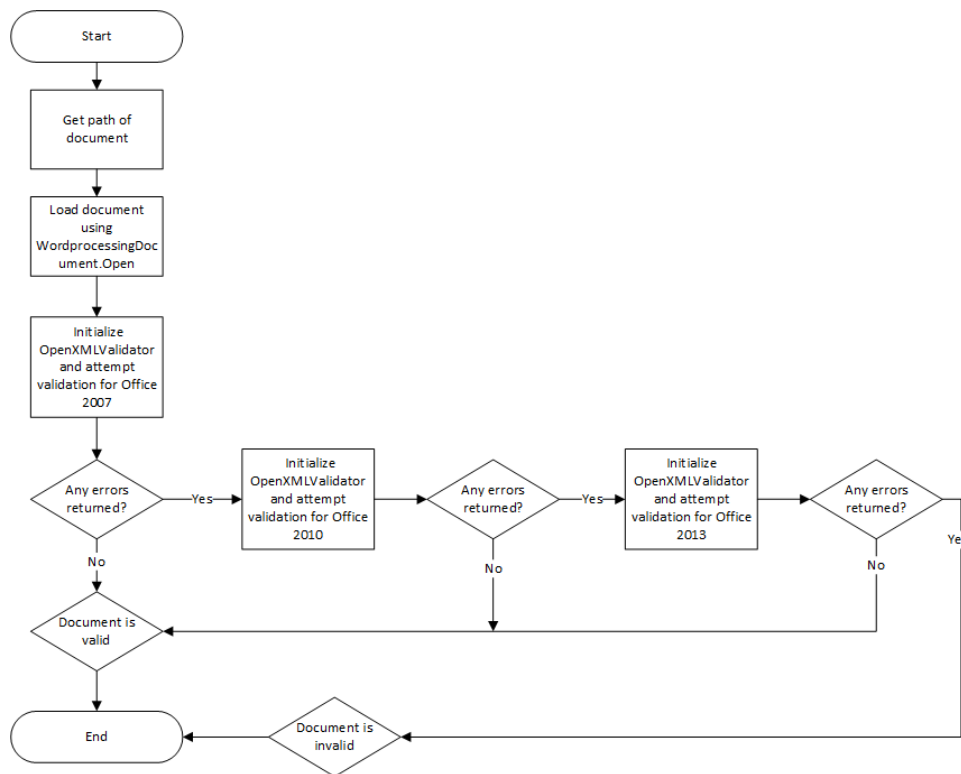


Figure 10: Flowchart showing the validation process.

### 5.2.2 Document metadata extraction

Forensically interesting metadata can be extracted from *docProps/app.xml* and *docProps/core.xml*, as described in Section 4.2.1. Functionality for extracting this information was implemented in OOFAT, and Figure 11 shows a flowchart of the implemented process of extracting metadata

from a document. We observe that neither EnCase nor FTK by default currently have functionality supporting metadata extraction in bulk.

Automating tasks is often desirable as it could save time, resources and ensure thoroughness [44][p. 5]. Performing bulk extraction of document metadata from a set of documents that are part of a forensic investigation could save a lot of time, in particular if the amount of documents is high [8, 10][Appendix D, E]. The document metadata extraction process is therefore included in a loop going through a list of input documents, where each document is unzipped and is input to the metadata extraction process. The output of this bulk extraction process is in the format of comma-separated values (CSV), which later can be processed in any way the investigators desire.

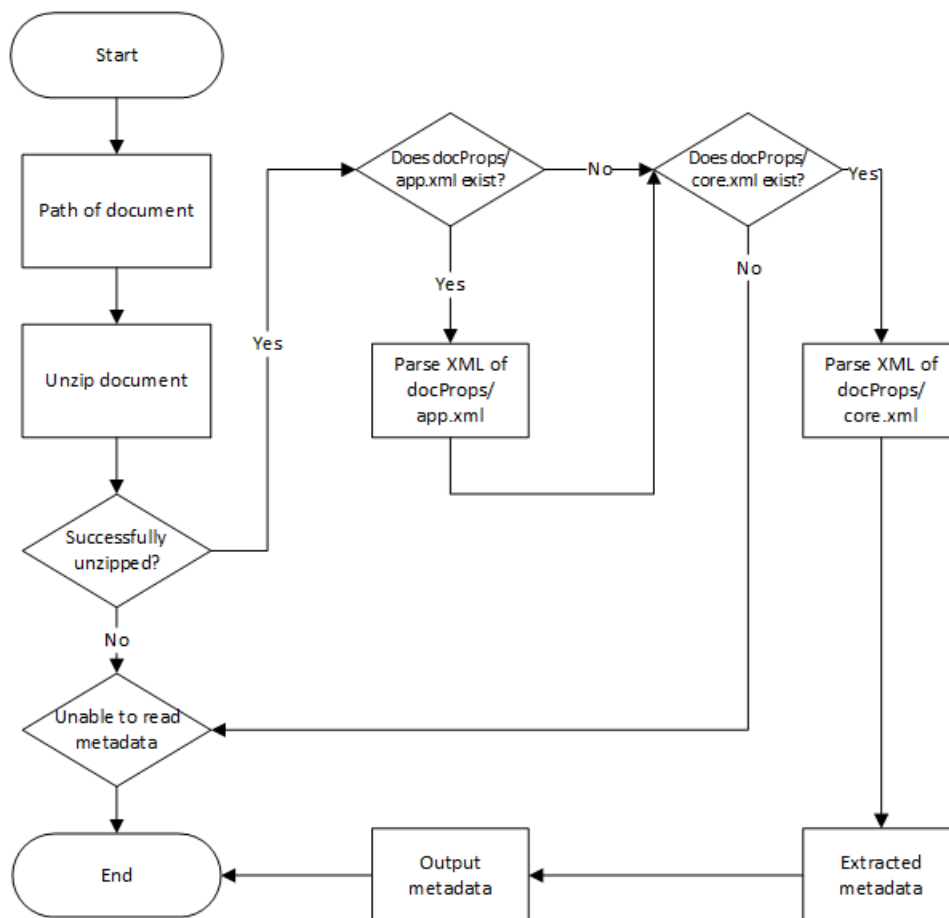


Figure 11: Flowchart showing the process of metadata extraction.

### 5.2.3 Revision identifier extraction and bulk comparison

Revision identifiers are present in *document.xml* and *settings.xml* in OOXML documents, as described in Section 4. The revision identifiers are as mentioned interesting from a forensic perspec-

tive as they may be used for e.g. document tracking, uncovering previously unknown networks and detecting plagiarism. A process for extracting revision identifiers from documents was implemented in OOFAT, in addition to a process for comparing the revision identifiers extracted from other documents.

The extraction process takes one or more documents as input, and for each document in the input collection, the steps shown in Figure 12 are executed. As the figure shows, an intermediate XML file is used for storing the extracted the document name, a document id, SHA1 hash and revision identifiers for each document. In this context, *document name* refers to the file name of the document and *document id* refers to a unique numeric value incrementing for each new entry in the intermediate XML file. The reason for storing the extracted information in an intermediate XML file is to reduce the time needed if it is desirable to perform revision identifier comparison more than once. The extraction and comparison could alternatively have been executed directly in the memory, but this would require additional computational time if the process is run more than once.

The process extracts the 6 most common types of revision identifiers from *word/document.xml*, namely:

- **Paragraph rsidR**: Paragraph creation; used to determine in which editing session the associated paragraph was added to the document [1][p. 236].
- **Paragraph rsidP**: Paragraph properties modification; used to determine in which editing session the associated paragraph's properties were modified [1][p. 236].
- **Paragraph rsidRPr**: Paragraph glyph formatting; used to determine in which editing session the glyph character for the paragraph mark associated with the paragraph was modified [1][p. 237].
- **Paragraph rsidRDefault**: Default value for all runs; used by all runs in the associated paragraph that do not have declared an *rsidR* attribute [1][p. 237].
- **Run rsidR**: run creation; used to determine in which editing session the associated run was added to the document [1][p. 291].
- **Run rsidRPr**: run properties modification; used to determine in which editing session the associated run's properties were modified [1][p. 292].

The process extracts *rsidRoot* and *rsid* values from *word/settings.xml*. After the extraction has been performed, the revision identifiers from one document could be compared with revision identifiers from one or more other documents to determine if they appear to originate from the same source.

The comparison process takes lists of revision identifiers from one or more documents, and compares them with lists of revision identifiers extracted from another document. This comparison is performed by using the *List.Intersect* method in C#, which compares each of the entries in two lists and outputs the values that are equal [45]. In case two documents have any intersecting



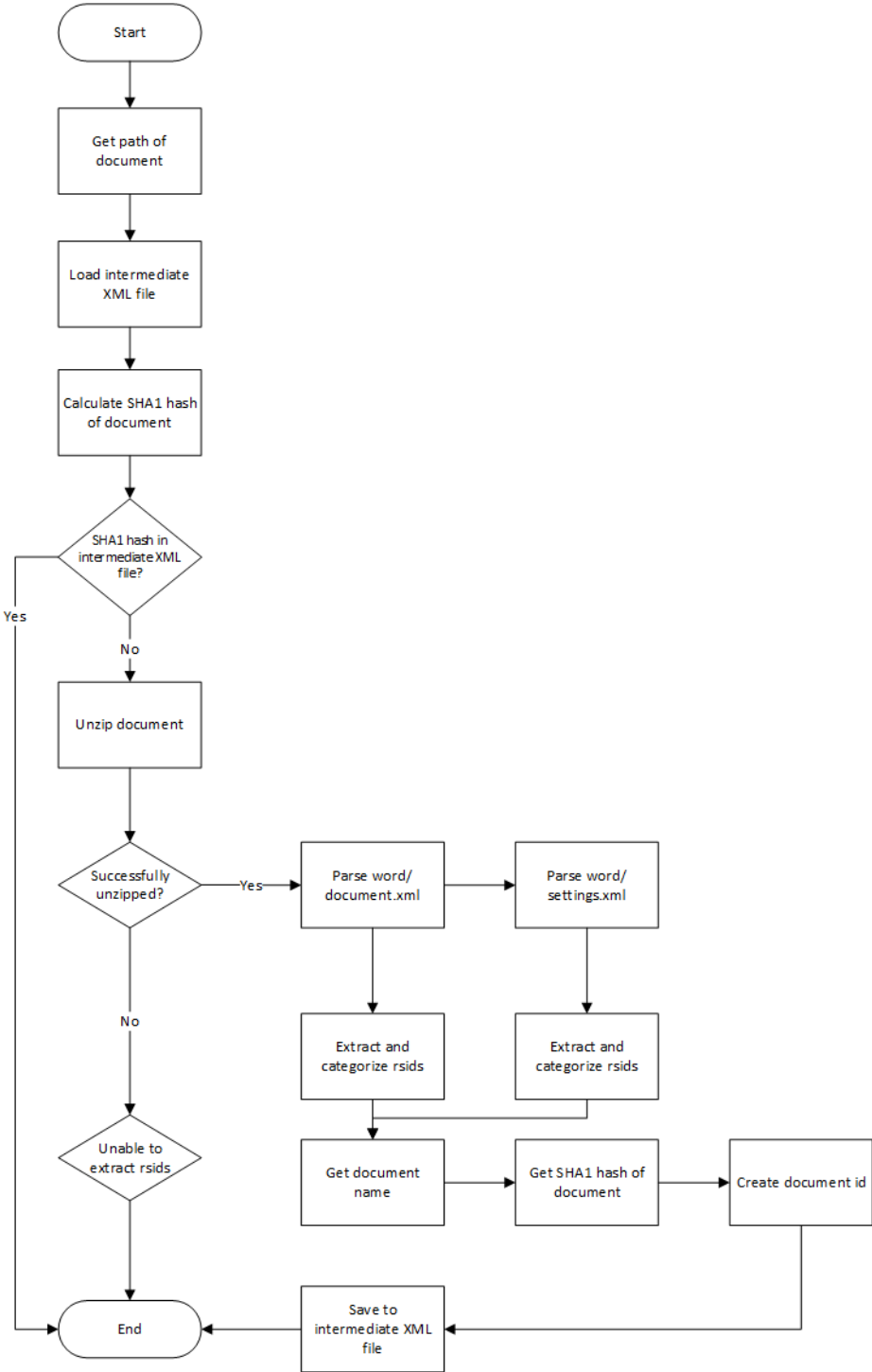


Figure 12: Revision identifier extraction process.

revision identifiers, this is as discussed in Section 4.5 an indication that the documents likely origin from the same source.

While the revision identifiers themselves cannot be used to determine which of the documents are the original, the creation timestamps from each document can be used to determine which of the documents appears to be the oldest, although timestamps are not always to be considered trustworthy. In case two documents share any revision identifiers, the comparison process of OOFAT therefore also compares the creation timestamps of each document when the output is presented as in a tree graph layout. The document with a creation timestamp that appears to be the oldest is considered the most likely original document. Figure 13 shows the implemented comparison process.

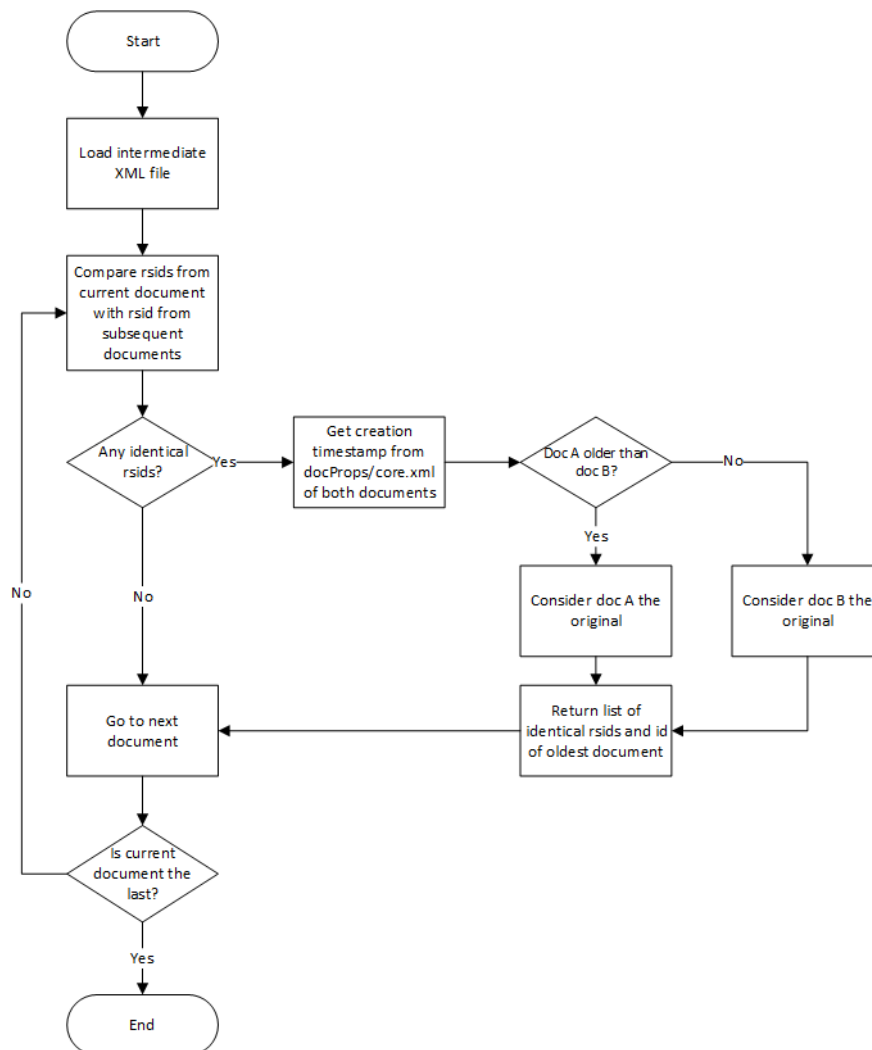


Figure 13: Revision identifier comparison process.

### 5.2.4 Revision identifier comparison output

OOFAT has implemented three possibilities of showing the result of the revision identifier comparison process described in Section 5.2.3:

- CSV output to text file;
- Table output;
- Tree graph layout output showing the relationship between documents graphically.

#### CSV output

The CSV output provides a simple, text-based output forensic investigators often desire [10, 8][Appendix E, D], since it can be processed in any way the investigators may desire afterwards. One line is formatted in the following way: *document 1 id, document 1 file name, document 1 SHA1 hash, document 1 created timestamp, document 2 id, document 2 file name, document 2 SHA1 hash, document 2 created timestamp, intersecting revision identifiers*. Listing 5.1 provides a sample line of the CSV output of the revision identifier comparison process.

Listing 5.1: Sample CSV output of revision identifier comparison process, should be read as only one line.

```
5,businessmodel.docx,30188681dced07d0d0490efa874a28232daa0190,
2014-08-14T12:34:08Z,4,letter.docx,
7f8d7d789eab89c3d9bd5fe408e31d9050944b40,2014-02-29T19:57:08Z,
00A274DD,00C50FC5,001D1694,00C50FC5,00584FEF,003151C9
```

#### Table output

The table output provides the result of the revision identifier comparison process in a spreadsheet-like environment, similar to the CSV output. Figure 14 provides an example table produced by comparing the revision identifiers extracted from a set of sample documents. In addition to presenting the *document ids, names, hashes* and *creation timestamps*, intersecting revision identifiers are categorized based on their type. If the intersecting revision identifier is of the same type in both documents, it is put into its respective category, e.g. *run rsidRPr*. If there is a mismatch between the revision identifier types, e.g. one of the type *rsidR* and the other of the type *rsidP*, the value is placed in the *Uncategorized* column.

This output format gives the investigator the possibility of relatively quickly determining exactly what revision identifiers from which documents are intersecting, which could motivate further inspection of the documents. Since each intersecting revision identifier is provided on its own line, except for the first intersection, document pairs with a large amount of intersecting revision identifiers easily stand out compared to those with few. We argue, however, that the table output format is best suited for a small set of documents, as scrolling through a large table could be tiresome.

#### Tree graph layout output

The tree graph layout displays the relationship between documents with intersecting revision identifiers visually, by implementing Microsoft Automatic Graph Layout (MSAGL). MSAGL is a

Doc 1 ID	Doc 1 name	Doc 1 created	Doc 1 hash	Doc 2 ID	Doc 2 name	Doc 2 created	Doc 2 hash	Shared (run) rsidRPr	Shared (run) runRsidR	Shared rsidRPr	Shared rsidR	Shared rsidP	Shared rsidRDefault	Uncategorized rsids
5	0-uke_...	2014-08...	993...	4	0-uke_...	2014-02...	bc92...	00A274DD	00C50FC5	001D1694	00C50FC5	00584FEF	003151C9	
5	0-uke_...	2014-08...	993...	4	0-uke_...	2014-02...	bc92...	00FA7874						
5	0-uke_...	2014-08...	993...	4	0-uke_...	2014-02...	bc92...	009A4B44						
5	0-uke_...	2014-08...	993...	4	0-uke_...	2014-02...	bc92...	001D1694						
10	2012-0...	2011-03...	90c1...	9	2012-0...	2010-01...	c1c5...	00C4059C	00165FF1	00B461A4	00165FF1	00B461A4	00165FF1	
10	2012-0...	2011-03...	90c1...	9	2012-0...	2010-01...	c1c5...	00DF7EC1						
10	2012-0...	2011-03...	90c1...	9	2012-0...	2010-01...	c1c5...	00E8313F						
10	2012-0...	2011-03...	90c1...	9	2012-0...	2010-01...	c1c5...	0062044A						
10	2012-0...	2011-03...	90c1...	9	2012-0...	2010-01...	c1c5...	003E192D						
10	2012-0...	2011-03...	90c1...	9	2012-0...	2010-01...	c1c5...	00B461A4						
15	2013%...	2008-06...	71a...	9	2012-0...	2010-01...	c1c5...	00DF7EC1						
15	2013%...	2008-06...	71a...	10	2012-0...	2011-03...	90c1...	00DF7EC1						
16	2013%...	2008-12...	f431...	9	2012-0...	2010-01...	c1c5...	004947CA						
16	2013%...	2008-12...	f431...	15	2013%...	2008-06...	71a9...	003F36EA	00333FEA					
16	2013%...	2008-12...	f431...	15	2013%...	2008-06...	71a9...	0094465C						
17	2013%...	2011-07...	eb1...	10	2012-0...	2011-03...	90c1...	00346B62						
18	2013-0...	2011-07...	9e0...	10	2012-0...	2011-03...	90c1...	00346B62						
18	2013-0...	2011-07...	9e0...	17	2013%...	2011-07...	eb10...	00346B62						
19	2013-0...	2011-01...	068...	14	2013%...	2011-01...	59d0...	00667752					008D55F9	
19	2013-0...	2011-01...	068...	15	2013%...	2008-06...	71a9...	005B3FBD						
24	2013.1...	2011-01...	e3c0...	20	2013-0...	2011-01...	e088...	002C3721						
25	2013_0...	2011-01...	952c...	24	2013.1...	2011-01...	e3c0...	007C40DC						

Figure 14: Table output of revision identifier comparison process.

library for building and displaying Sugiyama-style tree graph layouts, which are characterised by horizontal rows of nodes connected by directed edges [46, 47]. This library was chosen due to its academic-friendly license, which makes it free to use for research purposes [48].

Figure 15 shows an example of MSAGL displaying the relationship between four documents, with respect to intersecting revision identifiers. It should be noted that only those nodes (symbolized by ellipses) with a direct edge (symbolized by an arrow shape) between each other share some revision identifiers. In the example presented in the figure, *EnglishEssay2014.docx*, *new.docx* and *essay.docx* have some intersecting revision identifiers, but *deliverable2.docx* shares some identical revision identifiers only with *essay.docx*. Nodes are set as the source node, i.e. those with the starting point of the edge, if their creation timestamp is the oldest. This process is shown in the floatchart in Figure 13.

The example in Figure 15 shows an edge that is highlighted with the mouse pointer. This feature provides quick access to displaying the revision identifiers that are intersecting in the two specific documents the edge connects. If the investigator is interested in more details about the two documents, clicking an edge of interest provides the *id*, *name*, *creation timestamp*, *SHA1 hashsum* of both documents, in addition to the intersecting revision identifiers placed in their respective categories. In case the investigator desires to view additional document metadata, this is possible by clicking a button located within the edge-details window. Figure 17 provides an example showing the interface displayed when a specific edge of interest is selected; Figure 18 shows the interface displayed when attempting to view the complete metadata of a document of

interest. Figure 16 shows an example of the interface when the *document id* is shown instead of the document name, which is beneficial when a large amount of documents are presented.

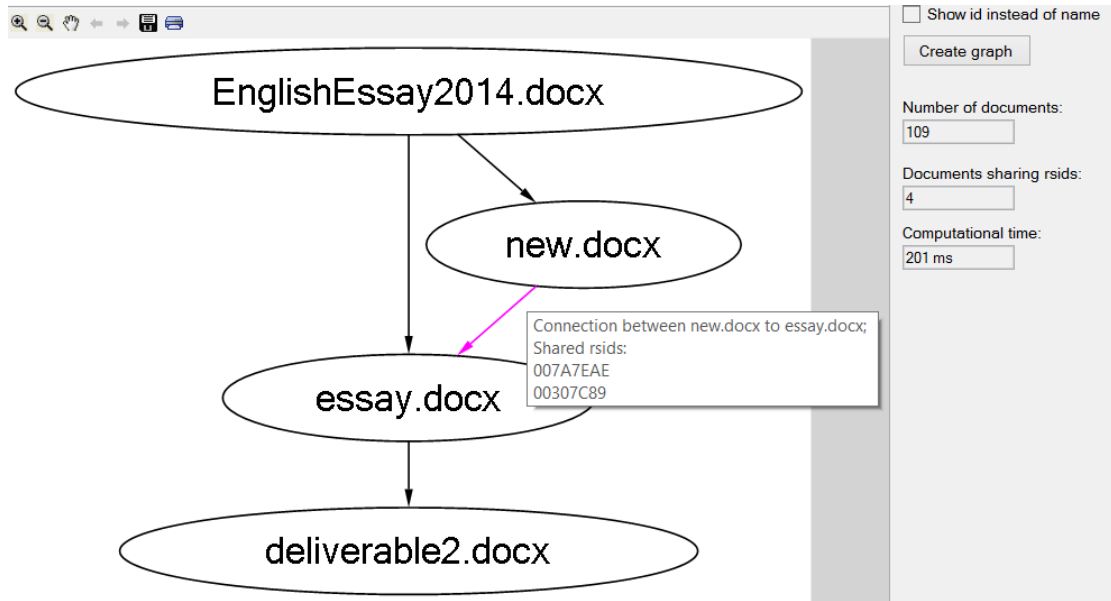


Figure 15: Example graph showing the relationship between four documents.

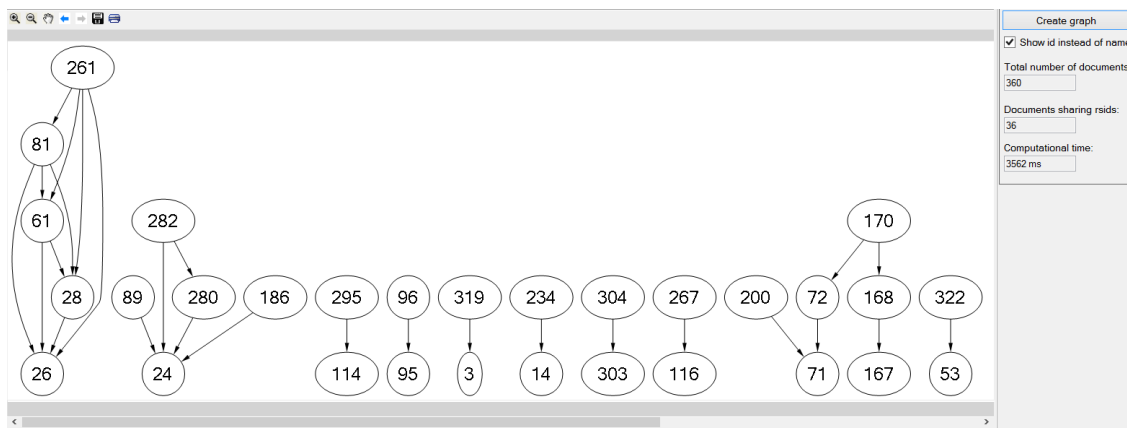


Figure 16: Example graph showing the relationship between documents from data set.

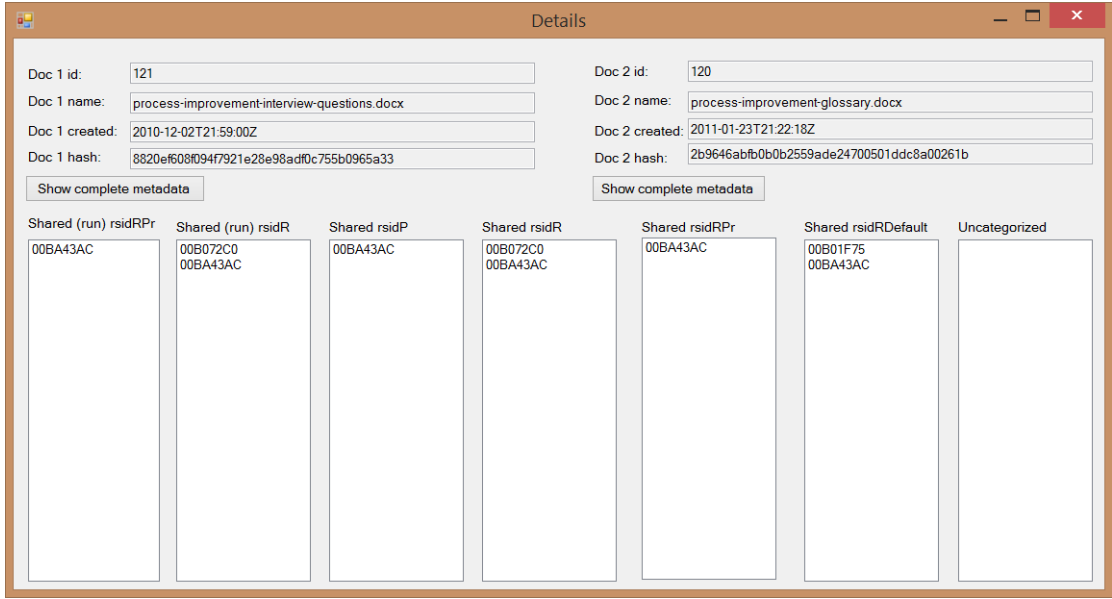


Figure 17: Example details page showing the information associated with a clicked edge.

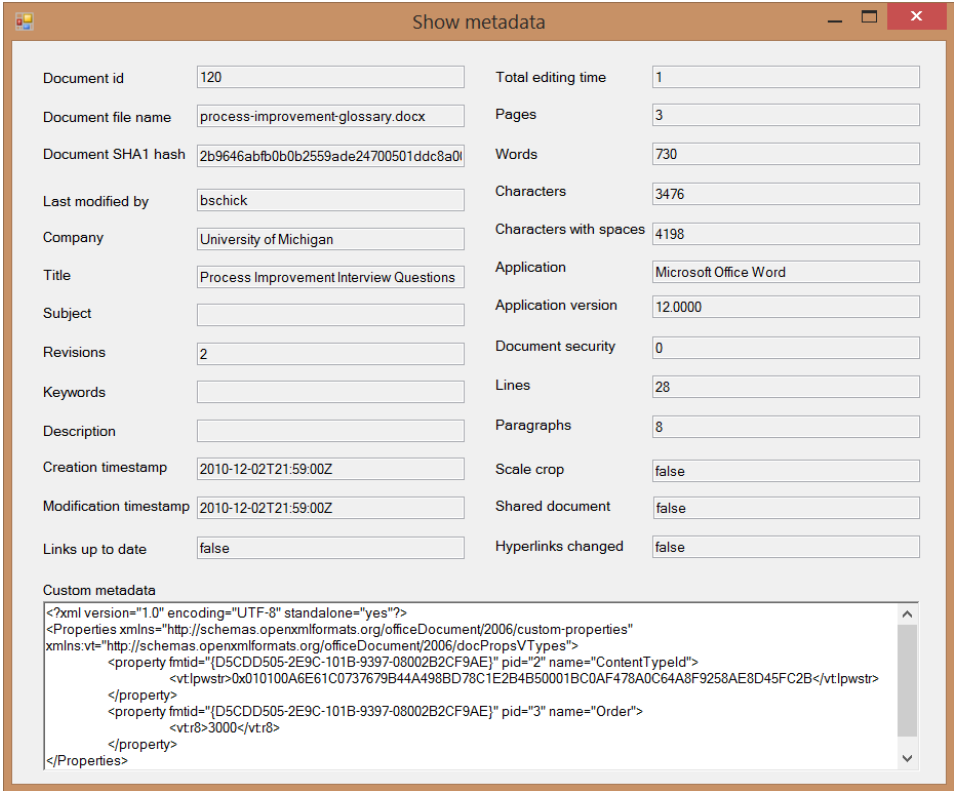


Figure 18: Example details page showing the complete metadata of a document of interest.

## 6 Experimental work

Several experiments were conducted in order to attempt answering the research questions and thereby trying to create additional knowledge about OOXML documents in the context of digital forensics.

Every office suite was installed inside Oracle VM VirtualBox virtual machines running Microsoft Windows 7 Professional, and each office suite was given its own clean Windows 7 install in order to avoid any potential conflicts that might arise when having several installations of the various office suites on the same system. Since we already owned copies of Microsoft Office 2007 and 2010, we used the “Standard” edition of the software bundle. For Office 2013, we chose the trial version which provides 60 days of “full-featured software” without cost [49]. A time-limited subscription of Office 365 Home [50] was purchased and downloaded the April 9th, 2014. The LibreOffice version used in the experiments was 4.1.1.2, which is provided without cost [51].

### 6.1 Prerequisite for experiment #4: Collecting data set of test documents

In order to attempt answering the research questions, a data set of independent test documents was needed. For the purpose of gathering a data set of independent test documents, two programs were developed. The first was a C# program implementing the Bing<sup>1</sup> search API. Bing supports custom search queries, making it possible to search for OOXML files by using a search query such as the one provided in Listing 6.1.

---

Listing 6.1: Bing search query for locating OOXML documents

---

```
<search term> filetype:docx ext:docx
```

---

Bing limits the amount of search results to 1,000 per search query executed, and provides a total of 5,000 queries per month for free. Therefore, a list of words producing a high amount (> 100) of hits was iterated and each word was used as *<search term>* in the search query listed above. Each URL was then output to a text file and cleaned for duplications and unrecognizable characters. It should be noted that since keywords producing a high amount of hits was used, a large amount of documents were bound to either directly origin from the same source, or contain content from the same source.

The second program we developed was a Python script with the purpose of iterating the file containing the URLs and downloading each of the documents. Once all files were downloaded, the program iterated the directory of collected files and calculated a unique SHA1 hash of each file. For each duplicate file, i.e. two or more matching SHA1 hashes, one instance of the file was

---

<sup>1</sup>Microsoft’s web search engine [52]

removed from the data set.

In total, 94621 OOXML documents were collected. After attempting to perform validation according to the standard, as described in Section 5.2.1, 18427 ( $\approx 19.47\%$ ) files were removed due to invalidity. The resulting data set therefore consists of 76194 files, with sizes ranging from 5 kB to 10213 kB (total size = 15266.1 MB, mean size = 205.167 kB). The size of the data set is displayed in Figure 19.

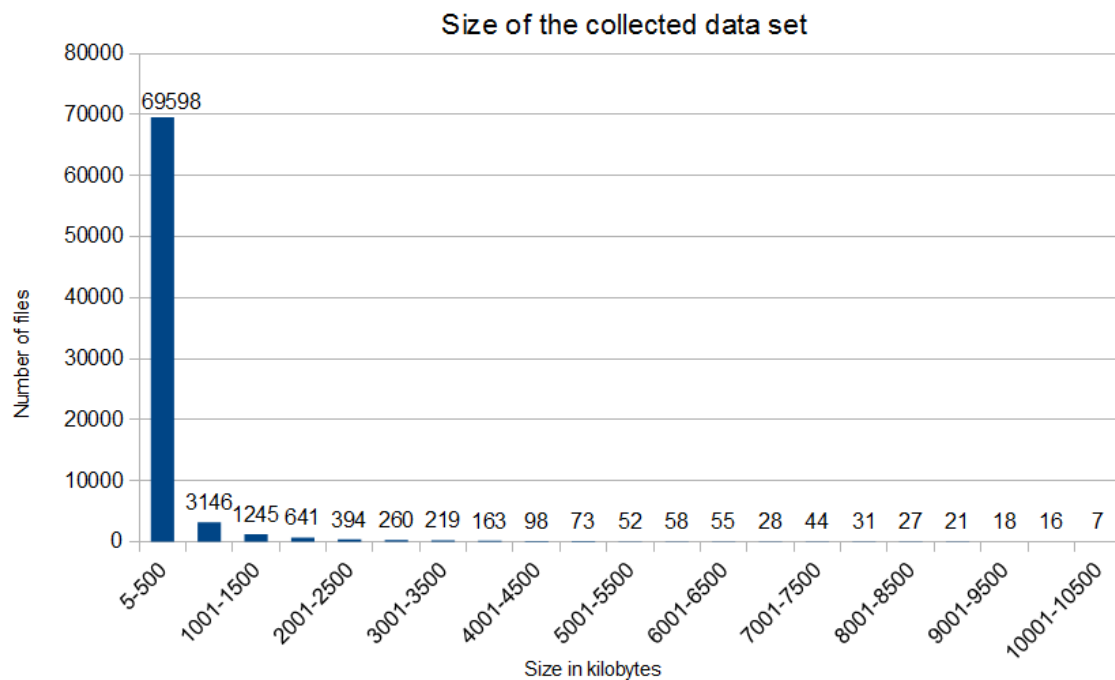


Figure 19: Chart showing the file size distribution of the collected data set.

## 6.2 Experiment #1: Interpretation of AppVersion number

As described in Section 4.2.1, *docProps/app.xml* in OOXML documents specify the name (XML tag named *Application*) and version (XML tag named *AppVersion*) of the application used for creating the document. We have not been able to identify research or official documentation from Microsoft that describes how the different values of *AppVersion* should be interpreted. Therefore, we performed a basic experiment to determine how to interpret the different version numbers we encountered while analysing the metadata of the test data set, as presented in Table 16.

### 6.2.1 Experiment #1 execution

Since we have access to Microsoft Office 2007, 2010, 2013, 365 and Word Online, these were the applications we wanted to correlate with the *AppVersion* numbers extracted from the metadata



of the test data set. Therefore, five basic experiments were executed, each with the following steps:

- Create document with the specific application;
- Unzip document package;
- Extract *AppVersion* number from *docProps/app.xml*;
- Correlate the version of Office used to create document with *AppVersion* number.

### 6.2.2 Experiment #1 results

Table 7 shows the result of executing the experiment for each version of Office.

Table 7: AppVersion interpretation experiment results

Version of Office	AppVersion number
2007	12.0000
2010	14.0000
2013	15.0000
365	15.0000
Online	00.0001

### 6.2.3 Experiment #1 analysis and discussion

Perhaps not entirely unexpected, the *AppVersion* number follows each new version of Office incrementally from Office 2007 to 2013, starting at 12.0000, ending at 15.0000, while skipping 13.0000, as shown in Table 7. This could be used by future researchers with the need to determine how to interpret the *AppVersion* number. It should be noted, however, that since Word Online is provided as a web resource, its *AppVersion* number could be changed at any time Microsoft desires.

## 6.3 Experiment #2: Revision identifier preservation in file and content copying

Fu et al. [5][p. 5] performed one experiment consisting of copying content from one Microsoft Word document to another, and another experiment consisting of creating a document based on a copied document. Their observation was that if some content is copied from a document to another, the receiving document records the revision identifiers generated in the original document.

In the case of a copied document used as a basis for a new document, i.e. editing a copy of a document, they observed that all the original revision identifiers are preserved in the new document if one or more characters remain with their original formatting. Furthermore, they found that the original revision identifiers are preserved in *settings.xml* if some content of the document is deleted or even if the document is blanked entirely.

When a document and some reference documents are available and of interest in a forensic investigation, these characteristics may have significant value for determining e.g. what has been done in a document and the document's origin, as presented in Section 4.5. It is therefore essential to verify that the presented results are correct for all situations, and if not, establish what situations could cause irregularities. Initial attempts to replicate the experiment resulted in failure to gain the same results. Consequently, we performed experiments with the purpose of verifying the validity of the results presented by Fu et al. [5][p. 5], and determining whether the research is applicable for newer versions of Microsoft Office, namely Office 2010 and 2013.

Both Fu et al. [5]'s and our experiments consisted of extensively testing copy-pasting material between documents. Our experiments used their research as a starting point, and we first attempted to duplicate their experimental setup to determine whether or not it yielded the same results in Office 2007, 2010 and 2013, respectively.

After performing the experiment duplication and analysis of the results, we first formed and attempted to verify the hypothesis “*Changing the text formatting of document A preserves the revision identifiers in document B when the affected content is copied from document A to document B*”. This hypothesis was formed after observing characteristics that will be presented in Section 6.3.2. These experiments were all performed using the setup as described in Section 6.

### 6.3.1 Experiment #2 execution

Since the starting point of this experiment was the research performed by Fu et al. [5], we first attempted to duplicate their experiments in order to verify their results. In their experiments, they first executed the following steps to generate sample documents used in further steps of their experiments [5][p. 5]:

1. Create a new document and type some new characters;
2. (Close the document/Word);
3. Open the document again and change e.g. text emphasis (boldface, underline, italics), color, font, font size;
4. (Close the document/Word);
5. Open the document again and type some new characters.

It should be noted that step 2 and 4 are not explicitly listed in their experiment execution, but can be extrapolated from step 3 and 5, as they specify that the document (i.e. an instance of Word with the document loaded) should be opened “again”.

This part of the experiment was executed 5 times (i.e. following the steps listed above to create 5 different documents) in Microsoft Word 2007, in order to determine if the subsequent steps in the experiment yielded consistent results, and thereby determining whether or not the results of Fu et al. [5] holds true for their setup. Furthermore, the same procedure was performed 5 times in Word 2010, Word 2013 and Word 365 to determine if the newer versions has the same

characteristics.

The first experiment we attempted to verify was their “forensics for file copy” experiment, where the purpose was to detect whether or not revision identifiers are preserved when a document is copied. The following steps were duplicated from Fu et al. [5][p. 5]:

1. Extract revision identifiers from *document.xml* and *settings.xml* from the original document and the copied document
2. Compare all revision identifiers extracted from both documents
3. If any *rsidR* attributes extracted from the two documents are identical, then it is possible to conclude that they are from the same source

The second experiment we attempted to verify was their “forensics for content copy” experiment, where the purpose was to see whether any revision identifiers were preserved in the new document when content was copied from one document to another. They did this by performing the sample document generation steps as listed in the beginning of this section, then they performed the same the extraction and comparison as listed in the steps above.

After duplicating these experiments, we used the revision identifier extraction and comparison functionality implemented in OOFAT, as described in Section 5.2.3. By using OOFAT, we reduced the amount of time needed for analysis, compared to performing manual inspection of the XML structure for each time the experiments were executed.

As will be presented in Section 6.3.2, the experiment duplication yielded results that motivated further experiments, in order to determine whether or not all situations of copy-pasting content preserves the original revision identifiers. We attempted to verify or refute our first hypothesis, “*Changing the text formatting of document A preserves the revision identifiers in document B when the affected content is copied from document A to document B*”, and this and all other experiments were executed with the following methodology:

1. Create document;
2. Write some arbitrary content in document;
3. Perform copy-pasting from document A to document B;
4. Extract and compare revision identifiers from both documents, in this case with OOFAT;
5. If revision identifiers in document A and document B match after repeating experiment, then the situation created in step 2 preserved the revision identifiers.

Fu et al. [5][p. 4] write “*Each new resulting RI value will be recorded once by the "setting.xml" file. The RI contained in the "setting.xml" file records the traces of the revisions of the document*”, where “RI” is an abbreviation of “revision identifier”. In order to clarify exactly what type of revision identifier is recorded in *settings.xml*, we performed an experiment, which consisted of creating and writing a document and revising it 6 times, and inspecting the document’s *word/settings.xml*

afterwards. The purpose of this experiment was to determine if those revision identifiers being preserved when content is copied is also preserved in *word/settings.xml*, which could be important if the content is removed.

### 6.3.2 Experiment #2 results

Our first experiment, a duplication of Fua et al. [5]’s “forensic for file copy” experiment, confirmed that their result holds true in the version of Office they used, Office 2007, and in addition for the newer versions Office 2010, Office 2013 and Office 365. While the presented research results holds true for their particular experimental setup, it does not hold true for every situation of copy-pasting content from one document to another.

Our experiments determined several additional important characteristics when copying content from one document to another, using Microsoft Word. As these characteristics might not be apparent, they will be presented in the following sections, where “document A” refers to the source (i.e. original) document, and “document B” refers to the receiving document.

#### Revision identifier preservation requirements

We found that when copying from document A to document B, the latter must be open in an instance of Microsoft Word if document A is closed *before* the content is pasted into document B, in order for the original revision identifiers to be preserved in document B. In other words, this means that if content is copied from document A to document B and the Microsoft Word process with document A is closed before the content has been pasted, the original revision identifiers will not be preserved if an instance of document B has not yet been started, and therefore new revision identifiers will be generated in document B and the original revision identifiers from document A will not be preserved. The original style (e.g. font, font size, color, bold) of document A will still be preserved in document B unless this functionality is disabled.

Through further experiments, it was observed that a *run* must have a *rsidRPr* attribute specified the XML of *document.xml* of document A, in order for the original revision identifiers to be preserved in document B when content is copied. In the context of *runs* (not to be confused with *rsidRPr* in the context of paragraphs), ECMA-376 specifies that the *rsidRPr* (*Revision identifier for run properties*) attribute “specifies a unique identifier used to track the editing session when the run properties were last modified in the main document.” [1][p. 292].

#### Situations producing the *rsidRPr* attribute

Our experiments determined several situations that makes Microsoft Word append the *rsidRPr* attributes to *runs*, which as mentioned are in most cases required in order for the revision identifiers to be preserved when content is copied from document A to document B. As the definition from the standard states, *rsidRPr* values track *run* properties modification. In practical terms, this means that when some text is typed into document A, no *rsidRPr* attribute is by default appended to the run since its properties have not yet been changed. Therefore, the original revision identifiers from document A will not be preserved when copying content to document B, unless the

properties of the run are changed and an *rsidRPr* attribute is appended to the run.

Experiments determined that if the style of the text is changed while writing, e.g. typing “*This is my*” then changing to bold text and continuing, “***first***”, changing back to non-bold and continuing “*document*”, no *rsidRPr* attribute is appended to the run. An example of the resulting XML of this is provided in Listing 6.2. If the style of the text is changed *after* the text has been written, e.g. typing “*This is my first document*” then changing “*first*” to bold after it has been typed, the modified content will get a *rsidRPr* attribute appended. An example of the resulting XML of this is provided in Listing 6.3.

Listing 6.2: Excerpt of XML from typing text and changing to bold *while* typing.

```
<w:r>
  <w:rPr>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">This is my </w:t>
</w:r>
<w:r>
  <w:rPr>
    <w:b/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t>first</w:t>
</w:r>
<w:r>
  <w:rPr>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve"> document</w:t>
</w:r>
```

Listing 6.3: Excerpt of XML from typing text and changing to bold *after* typing.

```
<w:r>
  <w:rPr>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">This is my </w:t>
</w:r>
<w:r w:rsidRPr="00B201A8">
  <w:rPr>
    <w:b/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t>first</w:t>
</w:r>
<w:r>
```

---

```

<w:rPr>
  <w:lang w:val="en-US"/>
</w:rPr>
<w:t xml:space="preserve"> document </w:t>
</w:r>

```

---

If any content (one or more characters from “first”) from the run represented with the *rsidRPr* value *00B201A8* is copied from the document shown in second example, Listing 6.3, the revision identifier *00B201A8* will be preserved in the new document. Therefore, the hypothesis “*Changing the text formatting of document A preserves the revision identifiers in document B when the affected content is copied from document A to document B*”, holds true.

In addition to getting *rsidRPr* attributes and values appended when style of the text is changed after it has been typed, experiments determined that runs get *rsidRPr* attributes if the content of document A is pasted from other sources, even if the source content has identical style preferences as the target document. In this context, “other sources” is defined as any other sources than document A itself, for instance other Word documents, web browsers, Notepad etc. This characteristic has high practical value from a forensic perspective, as the revision identifiers of the copied content from document A will always be preserved in document B, without the need for *runs* to be modified as described above.

Another characteristic discovered through our experiments, is that if the *final section* [1][p. 588] of the document, which is the very last paragraph break, is copied from document A and pasted into document B, the original revision identifier associated with the *final section* is preserved in document B. This revision identifier is appended as a *rsidRPr* attribute to the paragraph in document B. An example of the XML representing a paragraph break is provided in Listing 6.4.

When marking an entire paragraph in Word, the paragraph break will be selected by default, and the revision identifier will therefore be preserved if the last paragraph of the document is copied and pasted into another document. An example of the XML resulting from copying the last paragraph (including the paragraph break) is provided in Listing 6.5. The examples show that the *rsidRPr* attribute value *00EB6C6A* from the paragraph break is preserved and appended as an attribute to the paragraph revision.

---

Listing 6.4: Excerpt of example document A, showing the XML representing the last paragraph break.

```

<w:sectPr w:rsidR="00EB6C6A" w:rsidRPr="00EB6C6A">
...
</w:sectPr>

```

---



---

Listing 6.5: Excerpt of example document B, showing the XML of a paragraph break from document A

```

<w:p w:rsidR="004C3AF7" w:rsidRPr="00EB6C6A" w:rsidRDefault="004C3AF7"
w:rsidP="004C3AF7">

```

```

...
<w:t>This is my last sentence</w:t>
...
</w:p>

```

When pasting content from another source into a document, Microsoft Word has functionality that makes it possible to specify whether or not any style preferences of the original content should be used in the receiving document. This functionality is provided with a popup menu that appears when content is pasted, or through having a pre-defined default action for every time content is pasted. Our experiments determined that if *Merge formatting* or *Keep text only* is selected, any revision identifiers that normally would survive the copying will not be preserved in the new document.

### Revision identifier types stored in settings.xml

As described in Section 6.3.1, an experiment was executed in order to determine what types of revision identifiers are stored in *settings.xml* when a document is edited. Figure 20 shows the experiment document and its relevant parts of *document.xml* and *settings.xml*.

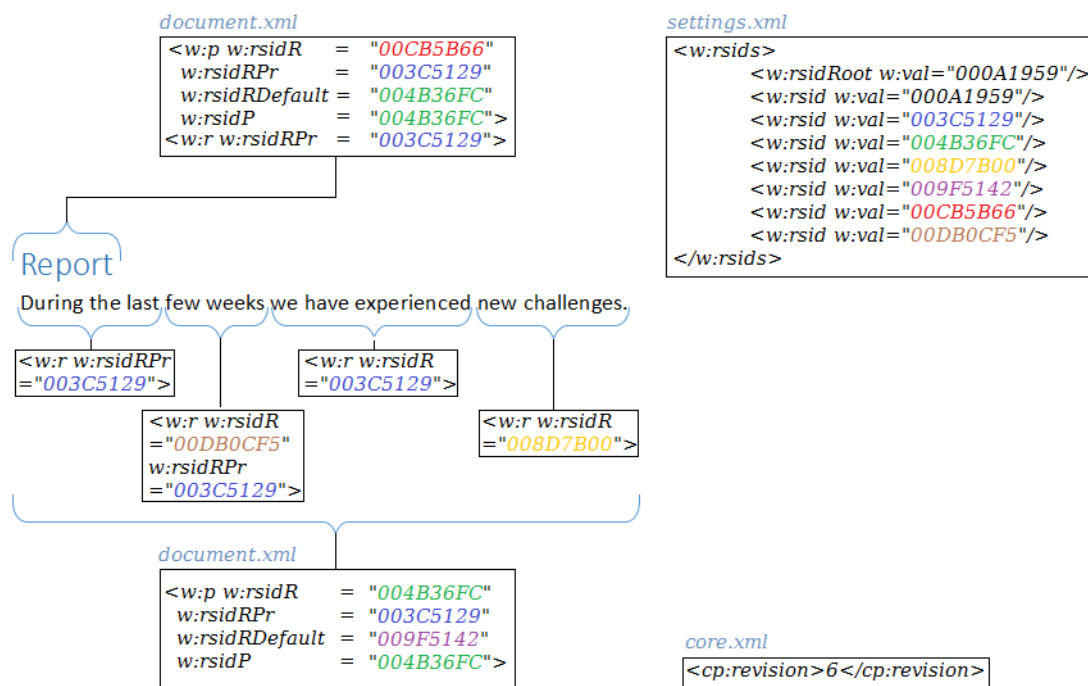


Figure 20: Example document with six revisions, and its revisions recorded in settings.xml.

As Figure 20 shows, a total of 6 revisions were recorded in *settings.xml* and *core.xml*. This number might at first seem to be erroneous, as only 5 saves appear to have been performed. However, this document was created by using the *New Microsoft Document* feature that appears when right

clicking a directory in Windows, as opposed to opening Microsoft Word and using *Save* or *Save as*. Therefore, the root revision identifier in *settings.xml*, with the value *000A1959*, is not recorded in *document.xml* or anywhere else in the package.

As can be extrapolated from the figure, the result of the experiment was that the *paragraph rsidR* and *paragraph rsidRDefault* attributes were recorded in *settings.xml*. These values were still intact even if the content of the paragraph or run was deleted, or the document was entirely blanked and rewritten with different content.

Lastly, the result of performing copying from document A and pasting into document B, as shown in Figure 20, under requirements that preserves the revision identifiers, as listed in Section 6.3.2, shows that while the original revision identifiers are preserved in *document.xml* in document B, they were not recorded in *settings.xml*. Therefore, any surviving revision identifiers will not be preserved in the package if the content is removed, since they are not recorded in *settings.xml*.

### 6.3.3 Experiment #2 analysis and discussion

We argue that the research presented by Fu et al. [5][p. 4-5] must be fine read: At a first glance, it does seem that copy-pasting material between documents will always preserve the revision identifiers from the original document in the new document. While this does hold true when following their experimental setup, which consisted of typing some arbitrary text and changing its style afterwards, our experiments have determined that it does not hold true in all other situations.

Our experiments have determined that there are certain requirements that must be fulfilled in order for the original revision identifiers to be preserved in document B when content is copied from document A, most importantly a *rsidRPr* associated with the *run* is required, and this attribute will be added e.g. when the style of the text is changed *after* it has been written. The revision identifiers will therefore always be preserved when following the same execution steps as Fu et al. [5].

These traces could have significant value in a forensic investigation, since they provide a hidden relationship between a seized document and one or several reference documents if the content is copied under the situations and with the requirements we have identified. As presented in Section 4.5, preserved revision identifiers can in some cases be used to detect plagiarism, uncover social networks and detect unauthorized distribution of sensitive documents.

Large amounts of run *rsidRPr* could indicate that the content has been copied from some other source, even though a reference document might not be available. This could in particular be apparent if the text style is identical to other text style of other parts of the document, i.e. the run *rsidRPr* attribute is not appended because of any change of properties. Determining that some content appears to be copy-pasted from another source might be of interest in forensic investigations involving e.g. plagiarism



## 6.4 Experiment #3: Forensic difference between office suites

Garfinkel et al. [9][p. 2] briefly demonstrate that Microsoft Office 2008 and NeoOffice for Macintosh store a thumbnail of the first page of the OOXML documents they edit, and that the two office suites store the thumbnails in different file formats. The fact that these two example office suites perform the same task slightly differently motivates more research to be performed on the difference between office suites, with respect to what forensically interesting information they record. We therefore utilize experimental research techniques to determine if there is a forensic difference between popular office suites Office 2007, 2010, 2013, 365, Word Online, Google Docs and LibreOffice.

### Implementation of revision identifiers

As presented in Section 4.2.3, revision identifiers were introduced as more privacy-friendly alternative to full change tracking, and is used by the word processor to provide a more accurate result when merging or comparing two documents that origin from the same source. The purpose of this experiment was to determine how the various office suites supporting OOXML implement and utilize revision identifiers.

### Original path preservation in image insertion

Preserved original file paths of inserted images could be very useful in a forensic investigation, in particular since they might include names, usernames and other identifiers that may help answer the “who” question, as discussed in Section 4.3. Furthermore, preserved original file paths may reveal that removable media such as a USB flash drive at some point has been used by the suspect, which could indicate that the investigators have not been able seize all evidence if a corresponding device has not yet been seized [10][Appendix E].

Upon previous inspection of OOXML files, we have observed that there might be a difference between the various available methods of inserting images into OOXML documents in Microsoft Office, with respect to whether or not the original file paths are preserved in the document. We therefore performed an experiment in order to attempt formalizing what situations produce what information when images are inserted into a document.

### Thumbnail creation and their readability

Thumbnails of the first page of the document could be important in a forensic investigation. Garfinkel et al. identified two potential uses of a thumbnail: i) determining if there is a mismatch between the last rendered first page and the actual first page in the document, which could be an indication of an attempt of malicious alteration of either the document or the thumbnail, ii) determining what the document was about, in the case where other parts of the document is corrupted but the thumbnail is intact or partly recoverable [9][p. 3]. This motivates an experiment to determine whether thumbnails are saved by default, and to what extent the thumbnails are readable.

### 6.4.1 Experiment #3 execution

#### Implementation of revision identifiers

This experiment was executed by first performing the following steps.

- Create a document with arbitrary content in the current office suite, and revise it 5 times;
- Extract the document package, inspect the content of *word/document.xml* and *word/settings.xml* and attempt to locate revision identifiers;
- Note how revision identifiers are implemented, compared to the implementation in Microsoft Office 2007;
- Repeat process for every office suite.

Furthermore, a second execution was performed as part of this experiment. The steps of this part of the experiment is provided in the following list.

- Create a document with arbitrary content in Microsoft Word 2007, and revise it 5 times;
- Open the document in the current office suite, make an arbitrary change and save the document;
- Extract the document package, inspect the content of *word/document.xml* and *word/settings.xml* and attempt to locate revision identifiers;
- Note if any revision identifiers are altered, compared to the original document;
- Repeat process for every office suite.

#### Original path preservation in image insertion

This experiment was executed by performing the following the steps provided in the following list.

- Insert image into document with each method available, i.e.:
  - Via *Insert -> Image*
  - Via drag-and-drop
  - From clipboard
  - From URL
  - From Facebook (Office 2013)
  - From Bing (Office 2013 and 365)
  - From Office.com (Office 2013 and 365)
  - From OneDrive (Office 2013 and 365)
- Extract the document package and inspect the resulting *word/document.xml* and locate the image reference;
- Repeat process for every office suite.

### Thumbnail creation and their readability

This experiment was executed by performing the following the steps provided in the following list.

- Create dummy document containing text and an image;
- Save document;
- Inspect document package to determine if a thumbnail image is present, and if not, go back to last step and enable thumbnail saving;
- Inspect thumbnail to determine its readability;
- Repeat process for every office suite.

### 6.4.2 Experiment #3 results

#### Implementation of revision identifiers

The results of this experiment are provided in Table 8 and Table 9.

Table 8: Implementation of revision identifiers in office suites; creating new OOXML document

Office suite	Interpretation of implementation
MS Office 2007	Identical to 2007
MS Office 2010	Identical to 2007
MS Office 2013	Identical to 2007
MS Office 365	Identical to 2007 in <i>word/document.xml</i> , but more revision identifiers are added in <i>word/settings.xml</i> ; these are not found in <i>word/document.xml</i>
Office Online	Mostly identical to 2007, but adds some additional revision identifier attributes: <i>w14:paraId</i> and <i>w14:textId</i>
LibreOffice	Does not use revision identifiers
Google Docs	Appears to use revision identifiers, but all numbers are nulled (“00000000”)

Table 9: Implementation of revision identifiers in office suites; editing OOXML document made in Office 2007

Office suite	Interpretation of implementation
MS Office 2007	Identical to 2007
MS Office 2010	Mostly identical to 2007; removes a <i>sectPr rsidSect</i> attribute

MS Office 2013	Mostly identical to 2007; removes a <i>sectPr rsidSect</i> attribute
MS Office 365	Mostly identical to 2007; removes a <i>sectPr rsidSect</i> attribute
Office Online	Mostly identical to 2007; removes a <i>sectPr rsidSect</i> attribute, and adds some additional revision identifier attributes: <i>w14:paraId</i> and <i>w14:textId</i> to each paragraph
LibreOffice	All original revision identifiers are removed, both attributes and their respective values. All identifiers are removed from <i>word/settings.xml</i>
Google Docs	All original revision identifiers are nulled, i.e. replaced with "00000000". All identifiers are removed from <i>word/settings.xml</i>

### Original path preservation in image insertion

Table 10 shows the result of the experiment consisting of attempting to determine whether or not the original path of inserted images is preserved when inserted into the different office suites, performed by using each of the available methods for insertion. For completeness, an extended version is provided in Appendix A, which includes performing image insertion via Bing, Facebook, Office.com and OneDrive.

Table 10: Original path preservation results of image insertion

Application	Insertion method	Result
Word 2007	Insert -> Image	Only original filename with extension, e.g. <i>vacation.png</i>
Word 2010	Insert -> Image	Only original filename with extension, e.g. <i>vacation.png</i>
Word 2013	Insert -> Image	Only original filename with extension, e.g. <i>vacation.png</i>
Word 365	Insert -> Image	Only original filename with file extension, e.g. <i>vacation.png</i>
Word On-line	Insert -> Image	Neither original path nor filename
LibreOffice Writer	Insert -> Picture -> From file	Neither original path nor filename
Google Docs	Insert via upload	Only original filename with file extension, e.g. <i>vacation.png</i>
Word 2007	Drag-and-drop	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 2010	Drag-and-drop	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 2013	Drag-and-drop	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>

Word 365	Drag-and-drop	Only original filename without file extension, e.g. <i>vacation</i>
Word On-line	Drag-and-drop	Not supported
LibreOffice Writer	Drag-and-drop	Neither original path nor filename
Google Docs	Drag-and-drop	Neither original path nor filename
Word 2007	From clipboard	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 2010	From clipboard	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 2013	From clipboard	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 365	From clipboard	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word On-line	From clipboard	Not supported
LibreOffice Writer	From clipboard	Neither original path nor filename
Google Docs	From clipboard	Neither original path nor filename
Word 2007	From URL	Only original filename with extension, e.g. <i>vacation.png</i>
Word 2010	From URL	Only original filename with extension, e.g. <i>vacation.png</i>
Word 2013	From URL	Only original filename with extension, e.g. <i>vacation.png</i>
Word 365	From URL	Only original filename file extension, e.g. <i>vacation.png</i>
Word On-line	From URL	Neither original path nor filename
LibreOffice Writer	From URL	Not supported
Google Docs	From URL	Only original filename file extension, e.g. <i>vacation.png</i>
Word 2013	From Facebook	Only original filename with extension, e.g. <i>10009314_10152851812487578_617485243_n.jpg</i>

### Thumbnail creation and their readability

Table 11 shows the results of this experiment.

Table 11: Thumbnail creation and their readability

Application	Thumbnail saved?	Location	Readability
Word 2007	Must check “Save thumbnail” when saving	<i>docProps\thumbnail.wmf</i>	Very poor quality; not possible to read but possible to see text and image structure. Appendix B.1.1 shows thumbnail, Appendix B.1.2 shows screenshot for reference
Word 2010	Must check “Save thumbnail” when saving	<i>docProps\thumbnail.emf</i>	Ok quality; possible to read although slightly poorly resized. Appendix B.2.1 shows thumbnail, Appendix B.2.2 shows screenshot for reference
Word 2013	Must check “Save thumbnail” when saving	<i>docProps\thumbnail.emf</i>	Ok quality; possible to read although slightly poorly resized. Appendix B.3.1 shows thumbnail, Appendix B.3.2 shows screenshot for reference
Word 365	Must check “Save thumbnail” when saving	<i>docProps\thumbnail.emf</i>	Ok quality; possible to read although slightly poorly resized. Appendix B.4.1 shows thumbnail, Appendix B.4.2 show screenshot for reference
Word On-line	Not supported <sup>2</sup>	Not supported	Not supported
LibreOffice Writer	Not supported	Not supported	Not supported
Google Docs	Not supported	Not supported	Not supported

### 6.4.3 Experiment #3 analysis and discussion

#### Implementation of revision identifiers

This experiment determined that Office 2007, 2010 and 2013 appear to implement revision identifiers in a practically identical way, with the exception of one attribute removed in 2010 and 2013. The revision identifier implementation in Office 365 is also approximately the same, with the exception of one attribute removed from *word/document.xml* and several additional revision identifier values added to *word/settings.xml*. These additional values could not be located in *word/document.xml* nor other files in the package, and therefore appear exclusively in *word/settings.xml*. The implementation in Office Online appeared to also be almost identical to Office 2007, with the exception of two additional revision identifiers added to each paragraph in the document.

<sup>2</sup>We observed that a 3 kb blank thumbnail (*docProps\thumbnail.jpeg*) was created when an image was inserted into the document via URL; does not have any value.

We found that both LibreOffice and Google Docs do not use revision identifiers; LibreOffice strips all existing identifiers from *word/document.xml* and *word/settings.xml* when it saves an edited document, and Google Docs replaces all existing identifiers with a null sequence. The practical implication of these implementations is that OOXML documents made or edited in LibreOffice or Google Docs cannot be used in a revision identifier comparison process for the purposes described in Section 4.5 and Section 5, therefore drastically reducing the documents' forensic usefulness. All versions of Office use revision identifiers, and can therefore be used in forensic investigations as long as they are not edited in LibreOffice, Google Docs or any other office suite that removes the identifiers.

### Original path preservation in image insertion

The experiment yielded some forensically interesting results. The preservation of original paths when performing image insertion could potentially be a very rich source of information. As presented in Section 2.1.2, Buchholz et al. identified six keywords for questions that forensic investigators may seek to get answered in an investigation: *Who, what, when, how, where* and *why*.

Original path preservation traces could be of particular support when attempting to answer the “who” keyword, i.e. providing an indication of who performed the actions. In this context, “actions” refers to inserting images into a document, which on a higher level can be regarded as a subset of document editing. Table 10 presents an example path of an inserted image, “C:\Users\Mallory\vacation.png”, and interpreting such path provides forensically interesting information: *Users* indicates that the document was edited on a machine using Windows Vista, 7 or 8 [53], *Mallory* indicates that the document was edited by a user account of that name.

Another interesting result of the experiment was the functionality in Word 2013 enabling the user to insert an image from a Facebook account. In our experiment, we observed that performing insertion of images from Facebook in Word 2013 requires the user to authenticate to Facebook and accept giving Word the permissions required to download the user's images. We observed that the original filenames of the inserted images are preserved in the document, e.g. *10009314\_10152851812487578\_617485243\_n.jpg*.

We observe that these filenames of Facebook images contain an identifier that can be connected to the Facebook user account used to publish the images, which currently is available by visiting <https://www.facebook.com/photo.php?fbid=<identifier>>, where *<identifier>* refers to the sequence of digits after the first underscore in the filename, e.g. 10152851812487578. Appendix C provides an example of the result of viewing a Facebook image, based on the identifier extracted from the original filename of an image inserted into a document. It should be noted that in order to be able to view the image online, the image must be set to public or the requester must have sufficient permissions to view it.

### Thumbnail creation and their readability

Thumbnails created in Word 2007 are unreadable, but it is possible to see how the content was structured. Word 2010, 2013 and 365 produce readable thumbnails, while Word Online, LibreOffice Writer and Google Docs do not support thumbnails. Thumbnails produced in Word 2010, 2013 and 365 are therefore more forensically useful than the other office suites.

## 6.5 Experiment #4: Uniqueness of revision identifiers

Revision identifiers can be used for purposes such as document movement tracking, uncovering social networks and detecting plagiarism. These possibilities are based on the assumption that revision identifiers are unique enough, in that the intersecting documents' revision identifiers are not identical merely by chance, but that they actually have some kind of relationship. A document based on another document is one example of such relationship, in addition to a document containing content copied from the other. In these situations, detected intersecting revision identifiers can be considered true positives, since there is a clear relationship between the documents.

Garfinkel et al. claim that *“there is, of course, a one in four billion chance that two of these 32-bit numbers will be the same”* [9][p. 4], likewise does Fu et al. provide the same statement [5][p. 4]. Due to the fact that neither of the studies describe any experimental research performed on the revision identifier number generator nor do they mention ECMA-376's requirement for generating revision identifiers, it can be assumed that their statement is based only on the number of possible combinations of a 32-bit number. ECMA-376 specified that *“Revision save IDs should be randomly generated based on the current time (to minimize the chance that two disparate editing sessions starting with the same immediate predecessor are assigned the same revision save ID)”* [1][p. 1049]. It is possible that the algorithm follows other routines that are not publicly known.

An experiment was performed with the purpose of attempting to determine if there are any false positives in the independent data set, i.e. documents with intersecting revision identifiers without a relationship, which could provide some knowledge about the uniqueness of the revision identifiers.

### 6.5.1 Experiment #4 execution

This experiment was executed by performing the steps provided in the following list. This process was repeated until the number of inspected document pairs equaled the pre-specified sample size ( $n = 100$ ).

1. Select 3000 documents from the independent data set collected with the setup described in Section 6.1;
2. Extract the revision identifiers from every document by using the extraction process of OOFAT, which is described in Section 5.2.3;
3. Perform comparison and present the output in a tree graph layout with OOFAT, as described in Section 5.2.4;
4. Perform manual inspection of those document pairs with intersecting revision identifiers;



5. If the inspection fails to detect any connection between the documents, it is considered a false positive;

### 6.5.2 Experiment #4 results

As shown in Appendix H.3, the result of this experiment was that of 100 performed inspections, 2 document pairs with no apparent actual connection were detected. This means that there was a 2% false positive rate among the chosen set of sample documents. For the first intersecting document pair, an excerpt of the XML of document A is provided in Listing 6.6, and an excerpt of the XML of document B is provided in Listing 6.7. The presented XML shows an intersecting *rsidRDefault* revision identifier, whose value “00392C21” is found 36 places in document A and three places in document B. Since it cannot be excluded that the documents are not intended for the public, the actual content associated with these revisions is not included in this thesis. Figure 21 shows a screenshot of the tree graph layout of OOFAT displaying the result of the first iteration of revision identifier comparison.



Figure 21: Tree graph layout of OOFAT showing the first iteration of revision identifier comparison, used for inspecting document pairs to determine their relationship.

It can, however, be noted that the documents were of completely different subjects, no common content was detected, the authors and companies extracted from the metadata were different, different versions of Office were used, and the documents were written in different years. Due to these reasons, the revision identifier intersection of these two documents were therefore classified as a false positive.

Listing 6.6: Excerpt of XML of document A

---

```
<w:p w:rsidR="00701F5A" w:rsidRPr="003148F0" w:rsidRDefault="00392C21"
  w:rsidP="000D63D1">
...
</w:p>
```

---

Listing 6.7: Excerpt of XML of document B

---

```
<w:p w:rsidR="00315A0E" w:rsidRPr="00CA4831" w:rsidRDefault="00392C21"
  w:rsidP="00315A0E">
...
</w:p>
```

---

### 6.5.3 Experiment #4 analysis and discussion

One important aspect of using the revision identifiers for e.g. determining the history of a document or detecting its unauthorized distribution, is to determine the uniqueness of the identifiers. This is important since a high number of false positives could reduce its forensic usefulness: If many matches without any real connection between the documents with intersecting identifiers occur, true positive alarms may be ignored or wrong conclusions may be drawn if the identifiers are used for e.g. uncovering social networks. In this context, false positives refers to when two documents with intersecting revision identifiers do not come from the same source, but have intersecting identifiers merely by chance.

This experiment yielded some interesting results: 2% (2 of 100) documents were likely false positive detections, which we consider to be quite low, yet higher than we initially expected. We note that in both of the false positive cases, the documents shared only one revision identifier number. These experiment results indicate that the revision identifiers are likely unique enough in that the false positive rate is low, and the identifiers can likely still be considered useful in a digital forensic context. However, since only a limited number of inspections were performed, we cannot exclude that a higher false positive rate may be detected if the sample size is increased.

## 7 Conclusions

This section presents conclusions for each of the research questions, based on the material presented in this thesis.

### 7.1 RQ1: What is the forensic value of OOXML documents, and how can they be used in forensic investigations?

Buchholz et al. identified the six keywords for questions that forensic investigators may seek to get answered in an investigation: *Who, what, when, how, where* and *why* [3][p. 5]. We have used these keywords as a basis for defining what could be forensically interesting; the more the piece of information could be used to answer the questions identified by Buchholz et al., the more forensically interesting we consider it to be.

In this thesis, we have seen that OOXML documents contain a large amount of forensically interesting metadata which in particular is found in the files *docProps/app.xml* and *docProps/core.xml*, which includes information such as the name of the document's creator, the name of the person who last modified the document, the application used to edit the document, timestamps of creation and modification. This information is considered to be of the main sources of evidence in many of the cases of National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway (ØKOKRIM) [8]. Timestamps are commonly extracted from files' metadata to build timelines of events that have occurred in a case [8, 10][Appendix D E].

Preserved file paths of images inserted into OOXML documents can reveal forensically interesting information, as they could include e.g. the name or username of the document's author, the operating system the author used, or a unique path on the system that could prove that the suspect's machine was used to create the document. If the inserted image has a preserved path showing that it originated from an external unit, this could be an indication that the forensic investigators might not have seized all units from the suspect [10][Appendix E].

OOXML documents produced by Microsoft Office contain unique revision identifiers, which are appended to the content of the documents in the XML files *word/document.xml* and *word/settings.xml*. These hexadecimal numbers are used by the word processor to provide a more accurate result when merging or comparing two versions of a document. Previous research has determined that the original paragraph creation revision identifiers are preserved in *word/settings.xml* even if parts of or all of the associated content is removed.

The revision identifiers extracted from a document can be used to track the source of a document if the forensic investigators have access to a reference document, and if the seized document is a result of copying the original document or copying content from the original document in certain

situations. This tracking is based on comparing the unique revision identifiers from the seized document and the reference document; if there are any intersecting numbers, they are likely from the same source. These numbers can therefore be valuable in investigations involving actions such as copyright infringement.

Another potential use of the revision identifiers can be used to uncover previously unknown social networks, such as a network of extremists [10][Appendix E]. In a similar manner as tracking the source of a document, a social network mapping could be performed by comparing the revision identifiers extracted from a seized document with the revision identifiers extracted from a set of documents that previously have been seized. In case any intersecting revision identifiers are detected, there is reason to believe that the person associated with the newly seized document in some way has communicated with the person associated with the document found in the database of previously seized documents.

## 7.2 RQ2: Can the metadata of OOXML document be trusted?

This research question was introduced because it is crucial to know if evidence should be trusted or not, since erroneous or otherwise uncertain evidence could lead to wrong conclusions in an investigation and evidence exclusion in a court of a law.

An OOXML document is a ZIP package containing a set of files in a file structure. The document's body and metadata are stored as XML files, and their contents can be modified by using a regular text editor. After the files have been modified, the adversary could create a ZIP package of the file structure. As long as the modifications comply with the OOXML standard, the document can be opened and edited in a word processor after it has been altered. Opening the document afterwards is unproblematic, and does not display any warnings unless the alteration damages the document, e.g. by removing an XML element the word processor expects.

Files extracted from OOXML documents created by Microsoft Word by default have a modification timestamp reflecting the start of the FAT epoch, *01.01.1980 00:00*, instead of the real modification time. If a file in the extracted package has been manually altered, e.g. by an adversary attempting to falsify evidence, the altered file gets a modification timestamp reflecting the actual last modification time. This timestamp remains the same even if the documents has been edited after the alteration occurred. Since files in an OOXML package normally has the modification timestamp of *01.01.1980 00:00*, a file with another timestamp found within the package is an indication that manual alteration has been performed.

Since the OOXML standard is an open standard, a determined adversary who seeks to falsify evidence can build his own office suite or a script that produces or modifies a document with falsified data. As long as the program follows the standard and mimics Microsoft Office or another office suite, the output document could be credible and the evidence falsification might not be possible to detect. The program could take a document as input and change only e.g. a revision identifier and the name of the document's creator to put the blame of an action on somebody

else, while the modification timestamps would show *01.01.1980 00:00*. Since this timestamp is what investigators would expect in a non-altered document, they would not be able to detect that the evidence was falsified.

Although the simplicity of manually altering the files in a OOXML package was one of the biggest fears the forensic community had about the format when it was released [8][Appendix D], experience The Norwegian National Authority for Investigation and Prosecution of Economic and Environmental Crime have gained show that criminals normally do not attempt to manually alter the files, but instead e.g. change the clock of the computer to falsify recorded timestamps [8][Appendix D].

We argue that technically speaking, the metadata of OOXML files cannot be trusted since it is so easy to alter it and falsify evidence. However, most criminals will not go through the trouble such alteration entails. Practically speaking, the metadata of OOXML files can in most cases therefore be trusted, unless there is reason to believe that alteration has been performed. Forensic investigators must, however, be aware that performing alteration and evidence falsification is a trivial task for a technical person.

Through analysis of documents created in Word Online, we have determined that the creation timestamps of these documents are erroneous and should not be trusted in a forensic investigation.

### **7.3 RQ3: Are there differences from version to version of the popular office suites, with respect to what forensically interesting data they record in the files? Does performing certain actions in different ways affect the recorded forensically interesting data?**

Garfinkel et al. [9] observed that the two office suites NeoOffice and Microsoft Office 2008 perform the task of storing thumbnails differently. This motivated our introduction of this research question, since there could be other differences between what forensically interesting information the various office suites store in the OOXML files they handle. From previous experience, we also observed when the user performs certain tasks differently in Microsoft Office, this could affect the forensically interesting information being stored.

We chose to inspect some of the differences between Microsoft Office (2007, 2010, 2013, 365 and Online), LibreOffice and Google Docs with respect to what forensically interesting information they store. We focused on the following:

**Implementation of revision identifiers:** The purpose of this experiment was to determine if the office suites implement revision identifiers differently. We found that Office 2007, 2010, 2013, 365 and Office Online use revision identifiers in an almost identical manner. LibreOffice and Google Docs do not use revision identifiers in OOXML documents, and even remove existing identifiers from documents produced in Microsoft Office. This drastically reduces their forensic

usefulness, since documents cannot be compared based on the identifiers, to e.g. track the source of a document.

**Original path preservation of inserted images:** Preserved original filepaths of inserted images could reveal forensically interesting information, such as the name of the author, the operating system used by the author, or a reference to a unit not yet seized. The purpose of this experiment was to determine if there is a difference between how original filenames are preserved, in addition to determining if different ways of inserting images lead to different information being stored. We found that for Office 2007, 2010 and 2013, the full original path of the image is preserved if the image is inserted by dragging and dropping the image into the document, or if the image is inserted from the clipboard. If an image is inserted via Facebook in Word 2013, the original filename can be used to track the profile of the user.

**Thumbnail creation and their readability:** A thumbnail of the first page of a document could be used to get an indication of what the document was about if the document is damaged, and the thumbnail is recoverable. We found that Word 2007's thumbnail is unreadable, but it is possible to see how the content was structured. The thumbnail made by 2010, 2013 and 365 is of decent quality, and can easily be read. Word Online, LibreOffice and Google Docs do not support thumbnails in OOXML documents.

#### **7.4 RQ4: In what ways can the revision identifiers be useful in a forensic investigation, and in what situations are they preserved?**

Since these numbers are 32-bit and unique, there are theoretically  $2^{32} = 4,294,967,296$  possible revision identifiers. We note that this is merely theoretical since Microsoft Office's revision identifier generator algorithm is not public. However, our initial analysis of the documents in the collected data set with intersecting revision identifiers detected 2% (2 of 100) false positives; meaning that 98% of the inspected documents actually shared some common content, and 2% were had intersecting revision identifiers merely by chance. Despite our analysis, we argue that the uniqueness of revision identifiers needs further research to be conclusive.

The literature has shown that revision identifiers can be used in visualization, as the identifiers associated with paragraph creation can visually represent the document as e.g. a barcode. This provides an overview of the how the document is put together, which is knowledge that can be difficult to gain based on manually reading the XML of the document body.

Documents that share unique revision identifiers are likely from the same source, contain content from the same source, alternatively one or both document contain content that has been copied from one document to another in certain situations. Comparing two documents to determine if they came from the same source, based on revision identifiers, was first implemented by Fu et al. [5]. They showed that these characteristics could be used to detect plagiarism in some cases. We have demonstrated that this implementation could be extended, by supporting the comparison of more than two documents and implementing visualization.

In the case where the comparison process results in any matching documents, the output is visualized in the form of a tree graph. The tree graph graphically displays the relationship between two or more documents, with respect to intersecting unique revision identifiers. By comparing the creation dates extracted from the XML metadata of the documents with intersecting revision identifiers, the document appearing to be oldest is set as the source node in the graph. This makes it possible for the investigator to quickly determine which of the documents are likely the original, and which appear to contain content copied from the source documents.

Our implementation has shown one way of presenting intersecting revision identifiers between documents, by displaying the identical revision identifiers in a “tooltip” window when hovering the node-connecting edges (i.e. the lines between the nodes, which represent the documents), and providing a more detailed view if the edges are clicked. This detailed view displays the metadata extracted from both documents, and categorized lists of intersecting revision identifiers. Interviews with forensic experts in *National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway* (Norwegian: *Økokrim*) and *NCIS Norway* (Norwegian: *Kripos*) have determined that this implementation could be useful in large cases, or cases with a large amount of documents [8, 10][Appendix D, E].

Through experiments, we determined that Fu et al. [5]’s research on preservation of revision identifiers when copying content between documents is slightly imprecise. Based on their research, it is easy to conclude that the original revision identifiers are preserved in all situations of copying and pasting. We have determined that the following two requirements must be fulfilled in order for the identifiers to be preserved in the receiving document:

- The receiving document must be open when the content is copied, if the original document is closed before the content is pasted.
- Content in a *run*, which is content sharing the same set of properties, must have a *rsidRPr* attribute specified.

We determined that a *run* gets the necessary *rsidRPr* attributed in the following situations.

- The properties of the the text, which refers to font, color, boldface, italics, underline etc., must be changed *after* the text has been written.
- The content of the original document is pasted from any another source than the document itself, e.g. another document, a web browser, Notepad and all other sources.

In addition, the original revision identifiers are preserved when the very last paragraph break is copy-pasted, which will happen if the last paragraph is highlighted and copied. Lastly, we determined that revision identifiers being preserved when copy-pasting are not recorded in *word/settings.xml*, and will be deleted from the document if all content associated with the *run* is removed.

## 8 Future work

We have several recommendations for those who seek to perform additional research on using these types of documents in the context of digital forensics.

### 8.1 Using visualization techniques to support forensic investigators

During the pre-project phase of this thesis, we originally planned to focus more on using visualization techniques to aid forensic investigators in handling evidence from OOXML documents. The horizontal barcode visualization of Langweg [2] was originally planned to be extended by implementing it with zoom and showing details when specific revisions in the barcode were clicked. Additionally, we wanted to evaluate such implementation by interviewing forensic investigators to determine if they would benefit from it, compared to the methods they currently use when performing analysis.

Due to prioritizing other experiments, this was not performed. However, we believe that this could be beneficial. Further research should therefore consider looking more into visualization of evidence from OOXML documents.

### 8.2 Optimizing the comparison process

We have presented a prototype for comparing the revision identifiers extracted from documents. We have not prioritized optimizing this comparison process, and it likely does not scale very well when a large amount of identifiers from a large amount of documents are input to the comparison process. The efficiency of the comparison process can likely be optimized greatly by storing the revision identifiers and other information extracted from documents in an SQL database, and perform comparison with SQL queries.

### 8.3 OOXML spreadsheets and presentations in digital forensics

This thesis has focused on OOXML documents, and has not dealt with OOXML spreadsheets (named Excel in Microsoft Office) nor OOXML presentations (named Powerpoint in Microsoft Office). These files do get collected in forensic investigations, in e.g. white collar crime, and share some similarities with OOXML documents. Further research should inspect these files to determine how they can be used in forensic investigations.

### 8.4 OpenDocument files in digital forensics

The Open Document Format for Office Applications (ODF) is an XML-based file format similar to OOXML. ODF is supported in a number of office suites, including LibreOffice, OpenOffice, Microsoft Office and NeoOffice. Our initial inspection of ODF files created by LibreOffice has determined that they use revision identifiers in a similar manner as Microsoft Office in OOXML files. Since ODF files likely contain forensically interesting information, further research should



inspect these files to determine their usefulness.

### **8.5 Microsoft Office’s revision identifier generator algorithm**

Previous research has claimed that there is a  $2^{32} = 4,294,967,296$  chance that two revision identifiers are identical by chance. This number only reflects the total number of possible combinations of a 32-bit number, and does not take Microsoft Office’s revision identifier generator algorithm into consideration. Details about this algorithm have not been published, except that ECMA-376 specifies that the revision identifiers should be “*randomly generated based on the current time*” [1][p. 1049].

We inspected 100 document pairs with intersecting revision identifiers by using a combination of our prototype’s comparison process and manual XML inspection. The purpose of this experiment was to determine if any document pair shared any revision identifier values without sharing any content, i.e. merely by chance. We found that 2% of the inspected documents had a false positive match, i.e. no actual shared content could be detected. If the false positive rate is too high, it drastically reduces their usefulness in a forensic context. Our experiment showed a relatively low false positive rate, but the revision identifier generator algorithm needs more research in order to be conclusive.

Future researchers should ideally attempt to reverse engineer the algorithm to determine exactly how it functions, alternatively perform experiments similar to ours with a larger sample set. Our experience is that determining if the documents with intersecting revision identifiers actually share some content needs human interpretation, and the task is therefore hard to automate.

## Bibliography

- [1] ECMA. December 2012. Standard ECMA-376 Office Open XML File Formats. <http://www.ecma-international.org/publications/standards/Ecma-376.htm>. [Online; accessed 28. January-2014].
- [2] Langweg, H. 2012. OOXML File Analysis of the July 22nd Terrorist Manual. *Communications and Multimedia Security. Springer Berlin Heidelberg*.
- [3] Buchholz, F. & Spafford, E. 2004. On the role of file system metadata in digital forensics. *Digital Investigation 1.4 (2004): 298-309*.
- [4] ECMA. December 2012. Standard ECMA-376 Office Open XML File Formats - Open Packaging Conventions. <http://www.ecma-international.org/publications/standards/Ecma-376.htm>. [Online; accessed 26. May-2014].
- [5] Fu, Z., Sun, X., Liu, Y., & Li, B. 2011. Forensic investigation of OOXML format documents. *Digital Investigation 8. 1*.
- [6] Microsoft. 2013. Office 365 for business FAQ. <http://office.microsoft.com/en-us/business/microsoft-office-365-for-business-faq-FX103030232.aspx>. [Online; accessed 26. May-2014].
- [7] Microsoft. 2013. Microsoft Office Online - Word, Excel, and PowerPoint on the web. <http://office.com>. [Online; accessed 26. May-2014].
- [8] Walmann, T. 2014. The Norwegian National Authority for Investigation and Prosecution of Economic and Environmental Crime (Norwegian: Økokrim). *Personal communication*.
- [9] Garfinkel, S. & Migletz, J. 2009. New XML-Based Files Implications for Forensics. *IEEE Security & Privacy 7. 2*.
- [10] Flølo, T. S. 2014. National Criminal Investigation Service Norway (Norwegian: Kripos). *Personal communication*.
- [11] Palmer, G. 2001. A road map for digital forensic research. *First Digital Forensic Research Workshop, Utica, New York*.
- [12] Raghavan, S. 2013. Digital forensic research: current state of the art. *CSI Transactions on ICT 1.1 (2013): 91-114*.
- [13] Casey, E. 2011. Digital evidence and computer crime: forensic science, computers and the internet. *Academic Press. ISBN 978-0-12-374268-1*.

- 
- [14] NISO Press. 2004. Understanding metadata. *National Information Standards 20*.
- [15] Gudjonsson, K. 2009. Office 2007 Metadata. <http://computer-forensics.sans.org/blog/2009/07/10/office-2007-metadata/>. [Online; accessed 20. January-2013].
- [16] Guidance Software. 2014. Overview. <https://www.guidancesoftware.com/products/Pages/encase-forensic/overview.aspx?cmpid=nav>. [Online; accessed 26. May-2014].
- [17] AccessData. 2014. FTK - Forensic Toolkit. <http://www.accessdata.com/products/digital-forensics/ftk>. [Online; accessed 26. May-2014].
- [18] Leedy, P. D. & Ormrod, J. E. 2013. Practical research: Planning and design, 10th edition. *Pearson Education Inc. ISBN 978-0-13-269324-0*.
- [19] Onwuegbuzie, Anthony J., N. L. L. & Collins, K. M. 2012. Qualitative Analysis Techniques for the Review of the Literature. *Qualitative Report 17*.
- [20] Google. 2014. Google Scholar. <http://scholar.google.com>. [Online; accessed 26. May-2014].
- [21] ScienceDirect. 2014. Search through over 11 million science, health, medical journal full text articles and books. <http://sciencedirect.com>. [Online; accessed 26. May-2014].
- [22] IEEE. 2014. IEEE Xplore Digital Library. <http://ieeexplore.ieee.org>. [Online; accessed 26. May-2014].
- [23] ACM. 2014. ACM Digital Library. <http://dl.acm.org>. [Online; accessed 26. May-2014].
- [24] Springer. 2014. SpringerLink. <http://link.springer.com>. [Online; accessed 26. May-2014].
- [25] Jones, B. 2007. History of office XML formats (1998-2006). [http://blogs.msdn.com/b/brian\\_jones/archive/2007/01/25/office-xml-formats-1998-2006.aspx](http://blogs.msdn.com/b/brian_jones/archive/2007/01/25/office-xml-formats-1998-2006.aspx). [Online; accessed 30. March-2014].
- [26] International Standards Organization. 2008. ISO/IEC DIS 29500 receives necessary votes for approval as an International Standard. <http://www.iso.org/iso/news.htm?refid=Ref1123>. [Online; accessed 30. March-2014].
- [27] Microsoft. 2011. Office Open XML, ECMA-376, and ISO/IEC 29500. [http://msdn.microsoft.com/en-us/library/gg607163\(v=office.14\).aspx](http://msdn.microsoft.com/en-us/library/gg607163(v=office.14).aspx). [Online; accessed 26. May-2014].
- [28] Bhorat, Z. 2008. A renewed wish for open document standards. <http://googleblog.blogspot.com/2008/02/renewed-wish-for-open-document.html>. [Online; accessed 30. March-2014].
- [29] Macnaghten, E. 2007. ODF/OOXML technical white paper. [http://www.freesoftwaremagazine.com/articles/odf\\_ooxml\\_technical\\_white\\_paper](http://www.freesoftwaremagazine.com/articles/odf_ooxml_technical_white_paper). [Online; accessed 30. March-2014].

- [30] Sayer, P. 2008. Changes to OOXML draft standard waved through. <http://www.infoworld.com/t/applications/changes-ooxml-draft-standard-waved-through-414>. [Online; accessed 30. March-2014].
- [31] MSDN. 2011. Understanding Office Binary File Formats. [http://msdn.microsoft.com/en-us/library/gg615407\(v=office.14\).aspx](http://msdn.microsoft.com/en-us/library/gg615407(v=office.14).aspx). [Online; accessed 26. May-2014].
- [32] ISO/IEC. 2012. Information technology – Document description and processing languages – Office Open XML File Formats – Part 1: Fundamentals and Markup Language Reference. <http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>. [Online; accessed 28. January-2014].
- [33] Hillmann, D. 2005. Using Dublin Core - The Elements. <http://dublincore.org/documents/usageguide/elements.shtml>. [Online; accessed 26. May-2014].
- [34] Jones, B. 2006. What's up with all those "rsids"? [http://blogs.msdn.com/b/brian\\_jones/archive/2006/12/11/what-s-up-with-all-those-rsids.aspx](http://blogs.msdn.com/b/brian_jones/archive/2006/12/11/what-s-up-with-all-those-rsids.aspx). [Online; accessed 26. May-2014].
- [35] Willassen, S. Y. 2008. Timestamp evidence correlation by model based clock hypothesis testing. *Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)*.
- [36] MSDN. 2014. extLst. [http://msdn.microsoft.com/en-us/library/dd905977\(v=office.12\).aspx](http://msdn.microsoft.com/en-us/library/dd905977(v=office.12).aspx). [Online; accessed 26. May-2014].
- [37] MSDN. 2014. oleSize. [http://msdn.microsoft.com/en-us/\library/documentformat.openxml.spreadsheet.olesize\(v=office.14\).aspx](http://msdn.microsoft.com/en-us/\library/documentformat.openxml.spreadsheet.olesize(v=office.14).aspx). [Online; accessed 26. May-2014].
- [38] U.S. Securities and Exchange Commission. 2014. Insider trading. <http://www.sec.gov/answers/insider.htm>. [Online; accessed 26. May-2014].
- [39] Microsoft. 2013. Incorporate revisions with track changes. <http://office.microsoft.com/en-001/word-help/video-incorporate-revisions-with-track-changes-VA104072631.aspx>. [Online; accessed 26. May-2014].
- [40] Garfinkel, S. L. 2014. tcpflow 1.3. <https://github.com/simsong/tcpflow>. [Online; accessed 09. February-2014].
- [41] Norwegian Police Security Service. 2014. Terrorisme. <http://www.pst.no/trusler/terrorisme/>. [Online; accessed 26. May-2014].
- [42] MSDN. 2014. WordprocessingDocument.Open Method (String, Boolean). [http://msdn.microsoft.com/en-us/library/cc562234\(v=office.14\).aspx](http://msdn.microsoft.com/en-us/library/cc562234(v=office.14).aspx). [Online; accessed 26. May-2014].

- 
- [43] MSDN. OpenXmlValidator.Validate Method (OpenXmlPackage). [http://msdn.microsoft.com/en-us/library/ee862967\(v=office.14\).aspx](http://msdn.microsoft.com/en-us/library/ee862967(v=office.14).aspx). [Online; accessed 26. May-2014].
- [44] Garfinkel, S. 2012. Lessons learned writing digital forensics tools and managing a 30TB digital evidence corpus. *Digital Investigation 9: S80-S89*.
- [45] MSDN. 2014. List<T>.Intersect Method. [http://msdn.microsoft.com/en-us/library/vstudio/bb910215\(v=vs.90\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/bb910215(v=vs.90).aspx). [Online; accessed 26. May-2014].
- [46] MSDN. 2014. Microsoft Automatic Graph Layout. <http://research.microsoft.com/en-us/projects/msagl/>. [Online; accessed 26. May-2014].
- [47] Sugiyama, Kozo, S. T. & Toda, M. 1981. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11.2: 109-125.
- [48] MSDN. 2011. Automatic Graph Layout. <http://research.microsoft.com/en-us/downloads/f1303e46-965f-401a-87c3-34e1331d32c5/>. [Online; accessed 26. May-2014].
- [49] Microsoft. 2013. Microsoft Office Professional Plus 2013. <http://technet.microsoft.com/en-us/evalcenter/jj192782.aspx>. [Online; accessed 23. March-2014].
- [50] Microsoft. 2013. Office 365 home. <http://office.microsoft.com/en-us/office365home/>. [Online; accessed 26. May-2014].
- [51] LibreOffice. 2014. What is LibreOffice? <http://www.libreoffice.org/discover/libreoffice/>. [Online; accessed 26. May-2014].
- [52] Microsoft. 2014. Bing. <http://bing.com>. [Online; accessed 26. May-2014].
- [53] Wikipedia. 2014. Default home directory per operating system. [http://en.wikipedia.org/wiki/Home\\_directory#Default\\_home\\_directory\\_per\\_operating\\_system](http://en.wikipedia.org/wiki/Home_directory#Default_home_directory_per_operating_system). [Online; accessed 26. May-2014].

## A Path preservation results table

Table 12: Original path preservation results of image insertion (extended version)

Application	Insertion method	Result
Word 2007	Insert -> Image	Only original filename with extension, e.g. <i>vacation.png</i>
Word 2010	Insert -> Image	Only original filename with extension, e.g. <i>vacation.png</i>
Word 2013	Insert -> Image	Only original filename with extension, e.g. <i>vacation.png</i>
Word 365	Insert -> Image	Only original filename file extension, e.g. <i>vacation.png</i>
Word On-line	Insert -> Image	Neither original path nor filename
LibreOffice Writer	Insert -> Picture -> From file	Neither original path nor filename
Google Docs	Insert via upload	Only original filename with file extension, e.g. <i>vacation.png</i>
Word 2007	Drag-and-drop	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 2010	Drag-and-drop	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 2013	Drag-and-drop	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 365	Drag-and-drop	Only original filename without file extension, e.g. <i>vacation</i>
Word On-line	Drag-and-drop	Not supported
LibreOffice Writer	Drag-and-drop	Neither original path nor filename
Google Docs	Drag-and-drop	Neither original path nor filename
Google Docs	From clipboard	Neither original path nor filename
Word 2007	From clipboard	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 2010	From clipboard	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 2013	From clipboard	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word 365	From clipboard	Full original path, e.g. <i>C:\Users\Mallory\vacation.png</i>
Word On-line	From clipboard	Not supported

LibreOffice Writer	From clipboard	Neither original path nor filename
Google Docs	From clipboard	Neither original path nor filename
7 Google Docs	From URL	Only original filename file extention, e.g. <i>vacation.png</i>
Word 2007	From URL	Only original filename with extention, e.g. <i>vacation.png</i>
Word 2010	From URL	Only original filename with extention, e.g. <i>vacation.png</i>
Word 2013	From URL	Only original filename with extention, e.g. <i>vacation.png</i>
Word 365	From URL	Only original filename file extention, e.g. <i>vacation.png</i>
Word On-line	From URL	Neither original path nor filename
LibreOffice Writer	From URL	Not supported
Google Docs	From URL	Only original filename file extention, e.g. <i>vacation.png</i>
Word 2007	From Bing	Not supported
Word 2010	From Bing	Not supported
Word 2013	From Bing	Only original filename with extention, e.g. <i>vacation.png</i>
Word 365	From Bing	Only original filename file extention, e.g. <i>vacation.png</i>
Word On-line	From Bing	Not supported
LibreOffice Writer	From Bing	Not supported
Google Docs	From Bing	Not supported
Word 2007	From Facebook	Not supported
Word 2010	From Facebook	Not supported
Word 2013	From Facebook	Only original filename with extention, e.g. <i>10009314_10152851812487578_617485243_n.jpg</i>
Word 365	From Facebook	Not supported
Word On-line	From Facebook	Not supported
LibreOffice Writer	From Facebook	Not supported
Google Docs	From Facebook	Not supported
Word 2007	From Office.com	Not supported

Word 2010	From Office.com	Not supported
Word 2013	From Office.com	Only original filename with extention, e.g. <i>vacation.jpg</i>
Word 365	From Office.com	Only original filename with extention, e.g. <i>vacation.jpg</i>
Word On-line	From Office.com	Not supported
LibreOffice Writer	From Office.com	Not supported
Google Docs	From Office.com	Not supported
Word 2007	From OneDrive <sup>1</sup>	Not supported
Word 2010	From OneDrive	Not supported
Word 2013	From OneDrive	Only original filename with extention, e.g. <i>vacation.jpg</i>
Word 365	From OneDrive	Only original filename with extention, e.g. <i>vacation.jpg</i>
Word On-line	From OneDrive	Not supported
LibreOffice Writer	From OneDrive	Not supported
Google Docs	From OneDrive	Not supported

<sup>1</sup><http://onedrive.live.com>; Microsoft's cloud storage service.



## B Thumbnail creation and readability experiment

### B.1 Word 2007

#### B.1.1 Thumbnail of document



Figure 22: Thumbnail produced by Office 2007

## B.1.2 Screenshot for comparison

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quae cum dixisset, finem ille. Illud dico, ea, quae dicat, praeciare inter se cohaerere. Tenent mordicus. Nemo igitur esse beatus potest. Sed tamen intellego quid velit. Haec paradoxica illi, nos admirabilia dicamus. Duo Reges: constructio interrete.

Mene ergo et Triarium dignos existimas, apud quos turpiter loquere? Miserum hominem! Si dolor summum malum est, dici aliter non potest. Quid enim ab antiquis ex eo genere, quod ad disserendum valet, praetermissum est? Nec vero sum nescius esse utilitatem in historia, non modo voluptatem. Sed nunc, quod agimus; Sed nimis multa. Putabam equidem satis, inquit, me dixisse.

Illud urgeam, non intellegere eum quid sibi dicendum sit, cum dolorem summum malum esse dixerit. Nunc de hominis summo bono quaeritur; Causa autem fuit huc veniendi ut quosdam hinc libros promerem.

Consequatur summam voluptatem non modo parvo, sed per me nihilo, si potest; Utrum igitur tibi litteram videor an totas paginas commovere? Minime vero istorum quidem, inquit. Quorum sine causa fieri nihil putandum est. Eademne, quae restincta sili?

Eam stabilem appellas. Pauca mutat vel plura sane; Nemo igitur esse beatus potest. Nostil, credo, illud: Nemo plus est, qui pietatem-; Traditur, inquit, ab Epicuro ratio neglegendi doloris. Sed nimis multa.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quae cum dixisset, finem ille. Illud dico, ea, quae dicat, praeciare inter se cohaerere. Tenent mordicus. Nemo igitur esse beatus potest. Sed tamen intellego quid velit. Haec paradoxica illi, nos admirabilia dicamus. Duo Reges: constructio interrete.

Mene ergo et Triarium dignos existimas, apud quos turpiter loquere? Miserum hominem! Si dolor summum malum est, dici aliter non potest. Quid enim ab antiquis ex eo genere, quod ad disserendum valet, praetermissum est? Nec vero sum nescius esse utilitatem in historia, non modo voluptatem. Sed nunc, quod agimus; Sed nimis multa. Putabam equidem satis, inquit, me dixisse.

Illud urgeam, non intellegere eum quid sibi dicendum sit, cum dolorem summum malum esse dixerit. Nunc de hominis summo bono quaeritur; Causa autem fuit huc veniendi ut quosdam hinc libros promerem.

Consequatur summam voluptatem non modo parvo, sed per me nihilo, si potest; Utrum igitur tibi litteram videor an totas paginas commovere? Minime vero istorum quidem, inquit. Quorum sine causa fieri nihil putandum est. Eademne, quae restincta sili?

Eam stabilem appellas. Pauca mutat vel plura sane; Nemo igitur esse beatus potest. Nostil, credo, illud: Nemo plus est, qui pietatem-; Traditur, inquit, ab Epicuro ratio neglegendi doloris. Sed nimis multa.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quae cum dixisset, finem ille. Illud dico, ea, quae dicat, praeciare inter se cohaerere. Tenent mordicus. Nemo igitur esse beatus potest. Sed tamen intellego quid velit. Haec paradoxica illi, nos admirabilia dicamus. Duo Reges: constructio interrete.

Mene ergo et Triarium dignos existimas, apud quos turpiter loquere? Miserum hominem! Si dolor summum malum est, dici aliter non potest. Quid enim ab antiquis ex eo genere, quod ad disserendum valet, praetermissum est? Nec vero sum nescius esse utilitatem in historia, non modo voluptatem. Sed nunc, quod agimus; Sed nimis multa. Putabam equidem satis, inquit, me dixisse.

Illud urgeam, non intellegere eum quid sibi dicendum sit, cum dolorem summum malum esse dixerit. Nunc de hominis summo bono quaeritur; Causa autem fuit huc veniendi ut quosdam hinc libros promerem.



Figure 23: Screenshot of first page of Office 2007 document

## B.2 Word 2010

### B.2.1 Thumbnail of document

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Te ipsum, dignissimum maioribus tuis, voluptasne induxit, ut adolescentulus eriperes P. An me, inquam, nisi te audire vellem, censes haec dicturum fuisse? Quae cum dixisset paulumque institisset, Quid est? Mihi enim erit isdem istis fortasse iam utendum. Duo Reges: constructio interrete. Quo igitur, inquit, modo? Quae cum essent dicta, discessimus. Non igitur potestis voluptate omnia dirigentes aut tueri aut retinere virtutem. Sed ad bona praeterita redeamus. Sic consequentibus vestris sublati prima tolluntur. Igitur neque stultorum quisquam beatus neque sapientium non beatus.

Bonum negas esse divitias, praepositum esse dicis? Quod autem satis est, eo quicquid accessit, nimium est, Urgent tamen et nihil remittunt. Quod quidem iam litetiam in Academia. Quid ergo?

Qui ita affectus, beatum esse numquam probabis; Polemoni etiam ante Aristoteli ea prima visa sunt, quae paulo ante dixi. Atqui reperies, inquit, in hoc quidem pertinacem; Quid censes in Latino fore? Quo modo? Consequentia exquirere, quoad sitid, quod volumus, effectum.

Quamquam id quidem licebitis exis timare, qui legerint. Addidisti ad extremum etiam indoctum fuisse. Tu vero, inquam, ducas licet, si sequetur; Quem Tiberina descensio festo illo die tanto gaudio affecit, quanto L. Nihil ad rem! Ne sitsane;

Ut in voluptate sit, qui epuletur, in dolore, qui torqueatur. Eadem nunc mea adversum te oratio est. Eadem nunc mea adversum te oratio est. Nulla profecto est, quin suam vim retineata primo ad extremum. Illa tamen simplicia, vestra versuta. Quis non odit sordidos, vanos, leves, futiles?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Te ipsum, dignissimum maioribus tuis, voluptasne induxit, ut adolescentulus eriperes P. An me, inquam, nisi te audire vellem, censes haec dicturum fuisse? Quae cum dixisset paulumque institisset, Quid est? Mihi enim erit isdem istis fortasse iam utendum. Duo Reges: constructio interrete. Quo igitur, inquit, modo? Quae cum essent dicta, discessimus. Non igitur potestis voluptate omnia dirigentes aut tueri aut retinere virtutem. Sed ad bona praeterita redeamus. Sic consequentibus vestris sublati prima tolluntur. Igitur neque stultorum quisquam beatus neque sapientium non beatus.

Bonum negas esse divitias, praepositum esse dicis? Quod autem satis est, eo quicquid accessit, nimium est, Urgent tamen et nihil remittunt. Quod quidem iam litetiam in Academia. Quid ergo?

Qui ita affectus, beatum esse numquam probabis; Polemoni etiam ante Aristoteli ea prima visa sunt, quae paulo ante dixi. Atqui reperies, inquit, in hoc quidem pertinacem; Quid censes in Latino fore? Quo modo? Consequentia exquirere, quoad sitid, quod volumus, effectum.

Quamquam id quidem licebitis exis timare, qui legerint. Addidisti ad extremum etiam indoctum fuisse. Tu vero, inquam, ducas licet, si sequetur; Quem Tiberina descensio festo illo die tanto gaudio affecit, quanto L. Nihil ad rem! Ne sitsane;

Ut in voluptate sit, qui epuletur, in dolore, qui torqueatur. Eadem nunc mea adversum te oratio est. Eadem nunc mea adversum te oratio est. Nulla profecto est, quin suam vim retineata primo ad extremum. Illa tamen simplicia, vestra versuta. Quis non odit sordidos, vanos, leves, futiles?



Figure 24: Thumbnail produced by Office 2010

## B.2.2 Screenshot for comparison

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Te ipsum, dignissimum maioribus tuis, voluptasne induxit, ut adolescentulus eriperes P. An me, inquam, nisi te audire vellem, censes haec dicturum fuisse? Quae cum dixisset paulumque institisset, Quid est? Mihi enim erit isdem istis fortasse iam utendum. Duo Reges: constructio interrete. Quo igitur, inquit, modo? Quae cum essent dicta, discessimus. Non igitur potestis voluptate omnia dirigentes aut tueri aut retinere virtutem. Sed ad bona praeterita redeamus. Sic consequentibus vestris sublatis prima tolluntur. Igitur neque stultorum quisquam beatus neque sapientium non beatus.

Bonum negas esse divitias, praepositum esse dicis? Quod autem satis est, eo quicquid accessit, nimium est; Urgent tamen et nihil remittunt. Quod quidem iam fit etiam in Academia. Quid ergo?

Qui ita affectus, beatum esse numquam probabis; Polemoni et iam ante Aristoteli ea prima visa sunt, quae paulo ante dixi. Atqui reperies, inquit, in hoc quidem pertinacem; Quid censes in Latino fore? Quo modo? Consequentia exquirere, quoad sit id, quod volumus, effectum.

Quamquam id quidem licebit iis existimare, qui legerint. Addidisti ad extremum etiam indoctum fuisse. Tu vero, inquam, ducas licet, si sequetur; Quem Tiberina descensio festo illo die tanto gaudio affecit, quanto L. Nihil ad rem! Ne sit sane;

Ut in voluptate sit, qui epuletur, in dolore, qui torqueatur. Eadem nunc mea adversum te oratio est. Eadem nunc mea adversum te oratio est. Nulla profecto est, quin suam vim retineat a primo ad extremum. Illa tamen simplicia, vestra versuta. Quis non odit sordidos, vanos, leves, futtiles?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Te ipsum, dignissimum maioribus tuis, voluptasne induxit, ut adolescentulus eriperes P. An me, inquam, nisi te audire vellem, censes haec dicturum fuisse? Quae cum dixisset paulumque institisset, Quid est? Mihi enim erit isdem istis fortasse iam utendum. Duo Reges: constructio interrete. Quo igitur, inquit, modo? Quae cum essent dicta, discessimus. Non igitur potestis voluptate omnia dirigentes aut tueri aut retinere virtutem. Sed ad bona praeterita redeamus. Sic consequentibus vestris sublatis prima tolluntur. Igitur neque stultorum quisquam beatus neque sapientium non beatus.

Bonum negas esse divitias, praepositum esse dicis? Quod autem satis est, eo quicquid accessit, nimium est; Urgent tamen et nihil remittunt. Quod quidem iam fit etiam in Academia. Quid ergo?

Qui ita affectus, beatum esse numquam probabis; Polemoni et iam ante Aristoteli ea prima visa sunt, quae paulo ante dixi. Atqui reperies, inquit, in hoc quidem pertinacem; Quid censes in Latino fore? Quo modo? Consequentia exquirere, quoad sit id, quod volumus, effectum.

Quamquam id quidem licebit iis existimare, qui legerint. Addidisti ad extremum etiam indoctum fuisse. Tu vero, inquam, ducas licet, si sequetur; Quem Tiberina descensio festo illo die tanto gaudio affecit, quanto L. Nihil ad rem! Ne sit sane;

Ut in voluptate sit, qui epuletur, in dolore, qui torqueatur. Eadem nunc mea adversum te oratio est. Eadem nunc mea adversum te oratio est. Nulla profecto est, quin suam vim retineat a primo ad extremum. Illa tamen simplicia, vestra versuta. Quis non odit sordidos, vanos, leves, futtiles?



Figure 25: Screenshot of first page of Office 2010 document



## B.3 Word 2013

### B.3.1 Thumbnail of document

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quorum sine causa fieri nihil putandum est. Duo Reges: constructio interrete. Proclivi currit oratio. Sed erat aequius Triarium aliquid de dissensione nostra iudicare. Etsi qui potest intellegi aut cogitari esse aliquod animal, quod se oderit? Quod quidem iam lit etiam in Academia. Quod ea non occurrentia fingunt, vincunt Aristonem; Ergo id est convenienter naturae vivere, a natura discedere. Haec para/docca illi, nos admirabilia dicamus. A quibus propter discendi cupiditatem videmus ultimas terras esse peragratas.

Atqui reperies, inquit, in hoc quidem pertinacem; Ego vero is li, inquam, permitto. Addidisti ad extremum etiam indoctum fuisse. Quicquid enim a sapientia proficiscitur, id continuo debet expletum esse omnibus suis partibus; Animum autem reliquis rebus ita perfecit, ut corpus, iam enim adesse poterit.

An nisi populari fama? Ergo et avarus erit, sed finite, et adulter, verum habebit modum, et luxuriosus eodem modo. Eadem nunc mea adversum te oratio est. Videamus igitur sententias eorum, tum ad verba redeamus. Non minor, inquit, voluptas percipitur ex vilissimis rebus quam ex pretiosissimis. Ex quo, id quod omnes expetunt, beate vivendi ratio inveniri et comparari potest. Tum ille: Tu autem cum ipse tantum librorum habeas, quos hic tandem requiris? An est aliquid, quod te sua sponte delectet? Non est igitur summum malum dolor. Tum, Quintus et Pomponius cum idem se velle dixissent, Piso exorsus est.

Ut placet, inquit, etsi enim illud erat aptius, aequum cuique concedere. Quod cum dixissent, ille contra. Restinguet citius, si ardentem acceperit. Hanc quoque iucunditatem, si vis, transfer in animum; Roges enim Aristonem, bonae ei videantur haec: vacuitas doloris, divitiae, validudo; Intellegi quidem, ut propter aliam quam plam rem, verbi gratia propter voluptatem, nos amemus; Videsne, ut haec concinant? Negatesse eam, inquit, propter se expetendam.

Quid adiuvas? Sed haec nihil sane ad rem; Sed emolumenta communia esse dicuntur, recte autem facta et peccata non habentur communia. Nos commodius agimus. Ne claphi suavitatem acupenserii Galloni Laelius anteponere, sed suavitatem ipsam neglebat; Quam ob rem tandem, inquit, non satisfacit? Minime vero istorum quidem, inquit. Eaedem enim utilitates poterunt eas labefactare atque pervertere. Quam ad modum quis ambulet, sedeat, qui ductus oris, qui vultus in quoque sit? Nonne videmus quanta perturbatio rerum omnium consequatur, quanta confusio?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quorum sine causa fieri nihil putandum est. Duo Reges: constructio interrete. Proclivi currit oratio. Sed erat aequius Triarium aliquid de dissensione nostra iudicare. Etsi qui potest intellegi aut cogitari esse aliquod animal, quod se oderit? Quod quidem iam lit etiam in Academia. Quod ea non occurrentia fingunt, vincunt Aristonem; Ergo id est convenienter naturae vivere, a natura discedere. Haec para/docca illi, nos admirabilia dicamus. A quibus propter discendi cupiditatem videmus ultimas terras esse peragratas.

Atqui reperies, inquit, in hoc quidem pertinacem; Ego vero is li, inquam, permitto. Addidisti ad extremum etiam indoctum fuisse. Quicquid enim a sapientia proficiscitur, id continuo debet expletum esse omnibus suis partibus; Animum autem reliquis rebus ita perfecit, ut corpus, iam enim adesse poterit.

An nisi populari fama? Ergo et avarus erit, sed finite, et adulter, verum habebit modum, et luxuriosus eodem modo. Eadem nunc mea adversum te oratio est. Videamus igitur sententias eorum, tum ad verba redeamus. Non minor, inquit, voluptas percipitur ex vilissimis rebus quam ex pretiosissimis. Ex quo, id quod omnes expetunt, beate vivendi ratio inveniri et comparari potest. Tum ille: Tu autem cum ipse tantum librorum habeas, quos hic tandem requiris? An est aliquid, quod te sua sponte delectet? Non est igitur summum malum dolor. Tum, Quintus et Pomponius cum idem se velle dixissent, Piso exorsus est.



Figure 26: Thumbnail produced by Office 2013

### B.3.2 Screenshot for comparison

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quorum sine causa fieri nihil putandum est. Duo Reges: constructio interrete. Proclivi currit oratio. Sed erat aequius Triarium aliquid de dissensione nostra iudicare. Etsi qui potest intellegi aut cogitari esse aliquod animal, quod se oderit? Quod quidem iam fit etiam in Academia. Quod ea non occurrentia fingunt, vincunt Aristonem; Ergo id est convenienter naturae vivere, a natura discedere. Haec para/docia illi, nos admirabilia dicamus. A quibus propter discendi cupiditatem videmus ultimas terras esse peragratas.

Atqui reperies, inquit, in hoc quidem pertinacem; Ego vero isti, inquam, permitto. Addidisti ad extremum etiam indoctum fuisse. Quioquid enim a sapientia proficiscitur, id continuo debet expletum esse omnibus suis partibus; Animum autem reliquis rebus ita perfecit, ut corpus; lam enim adesse poterit.

An nisi populari fama? Ergo et avarus erit, sed finite, et adulter, verum habebit modum, et luxuriosus eodem modo. Eadem nunc mea adversum te oratio est. Videamus igitur sententias eorum, tum ad verba redeamus. Non minor, inquit, voluptas percipitur ex vilissimis rebus quam ex pretiosissimis. Ex quo, id quod omnes expetunt, beate vivendi ratio inveniri et comparari potest. Tum ille: Tu autem cum ipse tantum librorum habeas, quos hic tandem requiris? An est aliquid, quod te sua sponte delectet? Non est igitur summum malum dolor. Tum, Quintus et Pomponius cum idem se velle dixissent, Piso exorsus est.

Ut placet, inquit, etsi enim illud erat aptius, aequum cuique concedere. Quod cum dixissent, ille contra. Restinguet citius, si ardentem acceperit. Hanc quoque iucunditatem, si vis, transfer in animum; Roges enim Aristonem, bonane ei videantur haec: vacuitas doloris, divitiae, validudo; Intellegi quidem, ut propter aliam quampiam rem, verbi gratia propter voluptatem, nos amemus; Videsne, ut haec concinant? Negat esse eam, inquit, propter se expetendam.

Quid adiuvas? Sed haec nihil sane ad rem; Sed emolumenta communia esse dicuntur, recte autem facta et peccata non habentur communia. Nos commodius agimus. Nec Iapathi suavitatem acupenseri Galloni Laelius anteponerat, sed suavitatem ipsam neglegebat; Quam ob rem tandem, inquit, non satisfacit? Minime vero istorum quidem, inquit. Eaedem enim utilitates poterunt eas labefactare atque pervertere. Quem ad modum quis ambulet, sedeat, qui ductus oris, qui vultus in quoque sit? Nonne videmus quanta perturbatio rerum omnium consequatur, quanta confusio?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quorum sine causa fieri nihil putandum est. Duo Reges: constructio interrete. Proclivi currit oratio. Sed erat aequius Triarium aliquid de dissensione nostra iudicare. Etsi qui potest intellegi aut cogitari esse aliquod animal, quod se oderit? Quod quidem iam fit etiam in Academia. Quod ea non occurrentia fingunt, vincunt Aristonem; Ergo id est convenienter naturae vivere, a natura discedere. Haec para/docia illi, nos admirabilia dicamus. A quibus propter discendi cupiditatem videmus ultimas terras esse peragratas.

Atqui reperies, inquit, in hoc quidem pertinacem; Ego vero isti, inquam, permitto. Addidisti ad extremum etiam indoctum fuisse. Quioquid enim a sapientia proficiscitur, id continuo debet expletum esse omnibus suis partibus; Animum autem reliquis rebus ita perfecit, ut corpus; lam enim adesse poterit.

An nisi populari fama? Ergo et avarus erit, sed finite, et adulter, verum habebit modum, et luxuriosus eodem modo. Eadem nunc mea adversum te oratio est. Videamus igitur sententias eorum, tum ad verba redeamus. Non minor, inquit, voluptas percipitur ex vilissimis rebus quam ex pretiosissimis. Ex quo, id quod omnes expetunt, beate vivendi ratio inveniri et comparari potest. Tum ille: Tu autem cum ipse tantum librorum habeas, quos hic tandem requiris? An est aliquid, quod te sua sponte delectet? Non est igitur summum malum dolor. Tum, Quintus et Pomponius cum idem se velle dixissent, Piso exorsus est.



Figure 27: Screenshot of first page of Office 2013 document

## B.4 Word 365

### B.4.1 Thumbnail of document

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quid nunc honeste dicit? Non ego tecum iam ita iocabor, ut isdem his de rebus, cum L. Mihi quidem Antiochum, quem audis, satis belle videris attendere. Ut nemo dubitet, eorum omnia officia quo spectare, quid sequi, quid fugere debeant? Duo Reges: constructio interrete.

Negat enim summo bono afferre incrementum diem. Portenta haec esse dicit, neque ea ratione ullo modo posse vivi; itaque his sapiens semper vacabit. Si quicquam extra virtutem habeatur in bonis. Nemo nostrum istius generis asotos iucunde putat vivere.

Illis videtur, qui illud non dubitant bonum dicere; illud dico, ea, quae dicat, praeclare inter se cohaerere. Non est enim vitium in oratione solum, sed etiam in moribus. Quid de Pythagora? Ne tum quidem te respicies et cogitabis sibi quemque natum esse etsuis voluptatibus? Quae cum dixisset paulumque institisset, Quid est? An potest, inquit ille, quicquam esse suavius quam nihil dolere? Nam, ut sint illa vendibiliora, haec uberiora certe sunt.

Videamus animi partes, quarum est conspectus illustrior; Sedulo, inquam, faciam. Tum mihi Piso: Quid ergo? Qui vere falsone, quaerere mitimus-dicitur oculis se privasse; Que Manilium, ab iisque M. Ad eas enim res ab Epicuro praecepta dantur. Quid igitur, inquit, eos responsuros putas? Aliter enim explicari, quod quaeritur, non potest.

Dempta enim aeternitate nihilo beatorum Iuppiter quam Epicurus; An hoc usque quaque, aliter in vita? Rhetorice igitur, inquam, nos magis quam dialectice disputare? Comoda autem et incommoda in eo genere sunt, quae praeposita et reiecta diximus; Sed quae tandem ista ratio est? Expectoque quid ad id, quod quaerebam, respondeas. Rhetorice igitur, inquam, nos magis quam dialectice disputare? Primum in nostrane potestate est, quid meminerimus?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quid nunc honeste dicit? Non ego tecum iam ita iocabor, ut isdem his de rebus, cum L. Mihi quidem Antiochum, quem audis, satis belle videris attendere. Ut nemo dubitet, eorum omnia officia quo spectare, quid sequi, quid fugere debeant? Duo Reges: constructio interrete.

Negat enim summo bono afferre incrementum diem. Portenta haec esse dicit, neque ea ratione ullo modo posse vivi; itaque his sapiens semper vacabit. Si quicquam extra virtutem habeatur in bonis. Nemo nostrum istius generis asotos iucunde putat vivere.

Illis videtur, qui illud non dubitant bonum dicere; illud dico, ea, quae dicat, praeclare inter se cohaerere. Non est enim vitium in oratione solum, sed etiam in moribus. Quid de Pythagora? Ne tum quidem te respicies et cogitabis sibi quemque natum esse etsuis voluptatibus? Quae cum dixisset paulumque institisset, Quid est? An potest, inquit ille, quicquam esse suavius quam nihil dolere? Nam, ut sint illa vendibiliora, haec uberiora certe sunt.

Videamus animi partes, quarum est conspectus illustrior; Sedulo, inquam, faciam. Tum mihi Piso: Quid ergo? Qui vere falsone, quaerere mitimus-dicitur oculis se privasse; Que Manilium, ab iisque M. Ad eas enim res ab Epicuro praecepta dantur. Quid igitur, inquit, eos responsuros putas? Aliter enim explicari, quod quaeritur, non potest.

Dempta enim aeternitate nihilo beatorum Iuppiter quam Epicurus; An hoc usque quaque, aliter in vita? Rhetorice igitur, inquam, nos magis quam dialectice disputare? Comoda autem et incommoda in eo genere sunt, quae praeposita et reiecta diximus; Sed quae tandem ista ratio est? Expectoque quid ad id, quod quaerebam, respondeas. Rhetorice igitur, inquam, nos magis quam dialectice disputare? Primum in nostrane potestate est, quid meminerimus?



Figure 28: Thumbnail produced by Office 365

## B.4.2 Screenshot for comparison

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quid nunc honeste dicit? Non ego tecum iam ita iocabor, ut isdem his de rebus, cum L. Mihi quidem Antiochum, quem audis, satis belle videris attendere. Ut nemo dubitet, eorum omnia officia quo spectare, quid sequi, quid fugere debeant? Duo Reges: constructio interrete.

Negat enim summo bono afferre incrementum diem. Portenta haec esse dicit, neque ea ratione ullo modo posse vivi; Itaque his sapiens semper vacabit. Si quicquam extra virtutem habeatur in bonis. Nemo nostrum istius generis asotos iucunde putat vivere.

Illis videtur, qui illud non dubitant bonum dicere -; Illud dico, ea, quae dicat, praeclare inter se cohaerere. Non est enim vitium in oratione solum, sed etiam in moribus. Quid de Pythagora? Ne tum quidem te respicies et cogitabis sibi quemque natum esse et suis voluptatibus? Quae cum dixisset paulumque institisset, Quid est? An potest, inquit ille, quicquam esse suavius quam nihil dolere? Nam, ut sint illa vendibilia, haec uberiora certe sunt.

Videamus animi partes, quarum est conspectus illustrior; Sedulo, inquam, faciam. Tum mihi Piso: Quid ergo? Qui-vere falsone, quaerere mittimus-dicitur oculis se privasse; Que Manilium, ab iisque M. Ad eas enim res ab Epicuro praecepta dantur. Quid igitur, inquit, eos responsuros putas? Aliter enim explicari, quod quaeritur, non potest.

Dempta enim aeternitate nihilo beatior Iuppiter quam Epicurus; An hoc usque quaque, aliter in vita? Rhetorice igitur, inquam, nos mavis quam dialectice disputare? Commoda autem et incommoda in eo genere sunt, quae praeposita et reiecta diximus; Sed quae tandem ista ratio est? Expectoque quid ad id, quod quaerebam, respondeas. Rhetorice igitur, inquam, nos mavis quam dialectice disputare? Primum in nostrane potestate est, quid meminerimus?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quid nunc honeste dicit? Non ego tecum iam ita iocabor, ut isdem his de rebus, cum L. Mihi quidem Antiochum, quem audis, satis belle videris attendere. Ut nemo dubitet, eorum omnia officia quo spectare, quid sequi, quid fugere debeant? Duo Reges: constructio interrete.

Negat enim summo bono afferre incrementum diem. Portenta haec esse dicit, neque ea ratione ullo modo posse vivi; Itaque his sapiens semper vacabit. Si quicquam extra virtutem habeatur in bonis. Nemo nostrum istius generis asotos iucunde putat vivere.

Illis videtur, qui illud non dubitant bonum dicere -; Illud dico, ea, quae dicat, praeclare inter se cohaerere. Non est enim vitium in oratione solum, sed etiam in moribus. Quid de Pythagora? Ne tum quidem te respicies et cogitabis sibi quemque natum esse et suis voluptatibus? Quae cum dixisset paulumque institisset, Quid est? An potest, inquit ille, quicquam esse suavius quam nihil dolere? Nam, ut sint illa vendibilia, haec uberiora certe sunt.

Videamus animi partes, quarum est conspectus illustrior; Sedulo, inquam, faciam. Tum mihi Piso: Quid ergo? Qui-vere falsone, quaerere mittimus-dicitur oculis se privasse; Que Manilium, ab iisque M. Ad eas enim res ab Epicuro praecepta dantur. Quid igitur, inquit, eos responsuros putas? Aliter enim explicari, quod quaeritur, non potest.

Dempta enim aeternitate nihilo beatior Iuppiter quam Epicurus; An hoc usque quaque, aliter in vita? Rhetorice igitur, inquam, nos mavis quam dialectice disputare? Commoda autem et incommoda in eo genere sunt, quae praeposita et reiecta diximus; Sed quae tandem ista ratio est? Expectoque quid ad id, quod quaerebam, respondeas. Rhetorice igitur, inquam, nos mavis quam dialectice disputare? Primum in nostrane potestate est, quid meminerimus?



Figure 29: Screenshot of first page of Office 365 document



## C Facebook user identification based on inserted image

```
<pic:nvPicPr>
  <pic:cNvPr id="1" name="10009314_10152851812487578_617485243_n[1].jpg"/>
  <pic:cNvPicPr/>
</pic:nvPicPr>
```

Mobile Uploads

Back to Album · Espen's photos · Espen's Timeline Previous · Next

**Forensic Laboratory**

**Espen Didriksen**  
where work cutting-edge digital forensic research is in progress!

Album: Mobile Uploads  
Shared with: Public

Figure 30: Screenshot of Facebook user's published image, identified based on inserted image's original filename

## D Transcription of workshop with National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway (ØKOKRIM), 14/3-2014

*Context: A group of Master's students and one PhD student from Gjøvik University College specializing in digital forensics were invited to present our projects to forensic investigators working for National Authority for Investigation and Prosecution of Economic and Environmental Crime in Norway (ØKOKRIM), in order to get feedback from experts.*

**John-Erik (GUC):** Have you detected any collisions in your data set?

**Me:** No, not yet. That's part of my further work. ... To see if there any collisions. They shouldn't be there.

**John-Erik (GUC):** So the linked documents you've already got there [*refers to slide graphically displaying documents with intersecting revision identifiers*], have you actually looked at them, do they seem similar?

**Me:** Obviously I have not looked at all of them, but those I've looked at share some values. [*Changes to graph slide*] I should probably have mentioned that these are some collected document found through [*search engine*] searches. So I performed some keyword searches, like "CV", "Møtereferat" etc., so obviously some of them will be made of templates, so some of them will probably share some of the values. [*Points to a group of nodes pointing to each other*] I think these are some company specific documents. I think it's common to just use templates or, for instance "Ukeplan", will obviously not be completely changed from week to week.

**John-Erik (GUC):** Also, you said you were going to check the id created from one session, is that id unique for each time you start Word on the same machine, or for every login session or is it unique for every machine...

**Me:** I'm sorry, which one?

**John-Erik (GUC):** You said that Word created a unique id when starting editing.

**Me:** Right.

**John-Erik (GUC):** But is that unique id the same every time for the same machine or...

**Me:** No, it's not. It's probably made in the same way as the other identifiers, so, what's inter-

esting here is that if you open one instance of Word, you type something, save it and close Word. When you open a new document, it will generate some other new value. But if you have several instances of Word open and you save documents, you could have for example 100 instances of Word open, saving, saving, saving... they all get the same value. I think that's interesting, because it probably means that the documents are from the same source, and that they are made at the same time.

**Thomas (Økokrim):** Everything in the format is well documented?

**Me:** Yeah, it's 6000 pages...

**Thomas (Økokrim):** You haven't found anything that's not documented?

**Me:** Not yet, no. Obviously, some details are not extremely well documented. Like how do they generate the revision identifiers? It's mostly up to the producer. If I make some similar tool as Word, I could do it in my way. The important thing for Microsoft when they made this standard, was that it should be this number of hexadecimal characters, and it should be unique, and it should be incremented. You could do it in any way you want if you make some office suite. So some [*forensically interesting*] details have been spared.

**Thomas (Økokrim):** One of the fears that the forensics community had when they went for this XML format, was that it's plaintext. So if you open a file and you don't use Word, and just sort of unzip it, and you try to use Notepad and try to change stuff, then it should sort of be easier to change the content of a file in an XML format, than in the old proprietary binary format. But according to you, it isn't?

**Me:** No, even if you change just one character, remove something, add something, and try to open it as a document again, it will not run in Word. I think it's absolutely possible, but it's not that easy. [*Note: This statement is erroneous; see Section 4.6.1.*]

**André (GUC):** Unless you can create that one once again... the hashsum or whatever it is, there is some integrity going on, probably. So if you could generate that one, if you know how to generate it, you could manipulate it.

**Me:** It's probably easy, yeah.

**Katrin (GUC):** Would it be possible to just change an identifier?

**Me:** Probably, yeah, it could be...

**Katrin (GUC):** A normal XML parser can understand... but they are randomly generated, so I could hide activities ...

**Me:** Yes, and you could for example get a document from Thomas and you could want him to get in trouble, and you could change some identifiers you have specified. It's probably easy if you want to do some bad things.

**Katrin (GUC):** So we have input for Microsoft for the next workshop.

**Thomas (Økokrim):** What's really interesting is that it's possible to dig more into the new formats, so it's possible to investigate further to see if there might be infringement or contracts or ...

**Me:** But I guess in most cases people using these kind of programs will not be aware of what's going in the documents.

**Thomas (Økokrim):** No. They would probably for instance try to change the time on the computer, but if you're then able to figure out how the timing comes into these... hashed values [*referring to revision identifiers*] ... there are new possibilities, at least.

**Me:** So there are some tiny details that could be useful, yeah.

**Me:** So I had some questions... but we could probably take them later.

**Thomas (Økokrim):** If you put them up, we could try to provide some input.

**Me:** [*Changing to slide with research questions*] So the first one, could you think of any other scenarios where these things could be useful?

**Thomas (Økokrim):** A lot of, for instance, if you copy and paste from different documents and you can trace identifiers that actually one person was in writing something into a document then at least you have reason to think that he had seen the document. We use metadata from Office documents a lot, it's one of the main evidence sources, where we find proof. Even lastOpened, lastPrinted and stuff like that. In the old Word format, you [*use*] "Save as" and you keep the old printed date and ... so we spend quite a lot of time investigating metadata in Word files.

**Me:** Have you ever used for example the revision identifiers for something, or do you mainly focus on timestamps, names etc?

**Thomas (Økokrim):** It's partly [*important*] being aware that it's possible to dig out things like that. As you say, it's a 6000 pages manual, and we haven't read it. I'm pretty sure that soon there will be someone claiming that someone had changed an identifier or something like that to blame it on someone else. Just having tools to check if someone obviously have done something, extract the identifiers etc etc [*is useful*], I'm not aware of any tools like that or at least none of

the tools we have at the moment have those possibilities.

**Katrin (GUC):** One could create a set of new security features like hashsums of identifiers and save them somewhere else to see if one character or identifier is changed.

**Thomas (Økokrim):** It's typically trying to reverse engineer ... for instance if you find different versions of the same document in different sources, then we have to sit down and try to reverse engineer to see how and who changed it, which we can then probably use the tags [*revision identifiers*] to help us.

**Me:** So, tools like EnCase, what kind of functionality do they have? I looked at FTK, but it was from 2006, so it was not so helpful.

**Thomas (Økokrim):** I once proposed that if you have some 10-15 files then you can send them and we can export the metadata with the tools we use. Typically, they only show what they showed from the old versions [*of Word*], prior to 2007, because that's what they've built into their databases. There has not been much focus on the metadata on Office documents in the forensic community. There has been more focused on what people are used to look at, for instance what's still kept in the registry. There are some most recent used stuff that's for some reason not in the XML documents, but in the registry. Forensic examiners are used to look into the registry and spend a lot of time digging around there, and they find something related to Office documents, and that becomes the focus. But very few have spent a lot of time digging into the XML structure of the Office files. We know that there are some things there that could help us in investigations, but we have not had a high-priority case yet where it was really important to dig in.

**Me:** You did mention that in most cases you actually look into the XML source manually, why?

**Thomas (Økokrim):** Because the tools... if it's important, then we have to look at it manually, because the tools do not extract sufficient information, none of the tools we have can extract [*for example*] the identifiers. So we basically have to go in and use your tool or do it manually. So as soon as we have a case where we need to dig into it, be sure that we will call you.

**Me:** That's good.

**Thomas (Økokrim):** But I'm not very aware of what's going in the open source community, because we haven't yet had the need to dig in.

**Me:** At least there aren't many good tools published. There are some tools that extract the metadata from app.xml and core.xml.

**Thomas (Økokrim):** So it's a business case to make a tool and prove that it's relevant, and

then AccessData or Guidance will buy your company.

**Me:** So it's not a good idea to open source it then.

**Thomas (Økokrim):** It's always a question of what you like. If you want to contribute to the open source community...

**Me:** And if you want money, then...

**Katrin (GUC):** Or you crowd fund it.

**Me:** Yeah, but it's mostly done [*completed*].

**Thomas (Økokrim):** But if you make a commercial thing, then you should be certain that you don't use open source code, you should make sure that you code things yourself or make sure that you use libraries that are okay to sell copies of.

**NN (Økokrim):** So you have a program that makes these metadata, or...?

**Me:** No, what you see here [*pointing to slide showing metadata extraction*] is extraction of metadata. So we have a input document, and it extract the metadata from it.

**Katrin (GUC):** Can you show the document?

**Me:** Do you want to see the XML?

**Katrin (GUC):** Just to get a feeling for it.

**Me:** Alright, so I don't have the original document from the screenshot, but this is an equivalent. [*Shows core.xml*] So this is core.xml, which contains creator information, modified, title, subject, number of revisions etc., and timestamps. [*Shows app.xml*] App.xml contains some information about the document, like how many characters, lines, what application was used etc. [*Shows document.xml*] So this is the document itself, and these are the identifiers. We can see that the paragraphs that have the same identifiers, are made within the same editing session.

**NN (Økokrim):** Does it take a long time to process the metadata from these documents?

**Me:** No, I'm [*currently*] processing an average of 4 documents per second. That's documents from about 20 kilobytes to up to 1 megabyte in the dataset, presented in the graph form. It's quite fast, I think. That includes unzipping and extracting. And the comparison, I think takes around 10 seconds if you have 300 documents. It depends also what you compare. If you want to compare all the attributes, there are 6 [*particularly interesting*] revision attributes [*types*], if

you want to compare all of them it takes a lot more time. [*Pointing to a run rsidRPr attribute on slide*] But it's mostly interesting to compare this attribute, because it's the one being preserved when you copy and paste content.

**(From session afterwards (not recorded))**

**Me:** What features would be important or interesting for Økokrim?

**Thomas (Økokrim):** i) Flagging something that appears to be strange in a document. ii) Having a master document, and comparing the identifiers to other reference documents to see their relationship. The tools we use today are sometimes a bit disappointing, they don't extract all metadata. None of the tools use the revision identifiers for anything. Would also like you to look some more into how easy it is to tamper with metadata.

**Me:** What kind of export format would Økokrim want?

**Thomas (Økokrim):** Simply CSV.

## **E Interview with Tom Sørensen Flølo from National Criminal Investigation Service (Norway) (Kripos)**

*Context: The thesis author was invited to visit Kripos' headquarters after requesting the possibility of interviewing a special investigator and perform document analysis with commercial forensic tools. During the interview, key points were written down and sentences were fully formed later on; the quotes are not exact wording.*

### **1. How do investigators from Kripos usually analyse OOXML documents today?**

Since we usually do not collect or analyse many documents in our typical investigations, we usually inspect them manually. It is, though, not unthinkable that we at some point in the future will have a case where we need to analyse such a large number of documents that doing it manually will be unfeasible.

### **2. What types of information do you usually look for and use from these documents?**

Of course, it really depends on the case. First and foremost, we are of course most interested in the actual contents of the document, and then of course who wrote it, where it came from etc. Furthermore, we could use details such as the preserved original paths of inserted images in order to determine if there might be units we have not yet seized from the suspect, for example units that may be hidden somewhere in the suspect's house. In one particular case, for example, we determined that 3 different installations likely were associated with editing the document, and some files appeared to origin from USB flash drives. The preserved original image paths could then indicate that there was some evidence we had not yet been able to seize.

We also use the document's metadata as an additional source of timestamps. For example, we could use these to build a timeline. If we for example determine that the suspect in fact appears to have been editing the document at the same time as a homicide was committed at some other place, it might be used as supporting alibi. Or the other way around; if the suspect claims he was at some other place and we determine that he in fact was at home editing the document.

### **3. What tools do you currently use for handling these documents?**

We often use EnCase and FTK, in addition to a lot of self-written scripts written to do a specific task. We usually use EnCase to extract the files from the forensic images, then inspect them manually or use `read_open_xml.pl` or similar scripts. We are also often interested in carving documents from images where the suspect has attempted to delete them. `bulk_extractor` is often useful.



**4. Is there any functionality that lacks in the commercial tools?**

The commercial tools almost don't have any functionality by default, but it is usually sufficient for us to analyse them manually. EnCase, for example, does not display XML properly. It is, however, possible to write scripts that extend the default functionality. EnCase, for example, uses a language called "EnScript", and developers distribute these scripts on (private) community forums. Analysis is currently lacking in most tools, but they are starting to implement various analysis possibilities, such as generating timelines, displaying connections etc. We see the problem in ever-increasing amounts of data seized, and doing things manually is hopeless when faced with a lot of information. There are too many results, and we need to sort based on e.g. time or do other analysis to shrink the amount of data needed to be analysed manually.

**5. Have the revision identifier in OOXML documents ever been used in any cases in Kripes?**

No, not that I am aware of. At least not the last 5 years.

**6. Could the analysis functionality I have presented be useful for Kripes?**

As of today, we currently haven't had any big cases where manually analysis didn't suffice. ØKOKRIM likely handle a lot more documents than we do. But it is as previously mentioned not unthinkable that a larger case or a case with a large amount of documents comes up at some point, then the functionality presented could be useful. The graph showing documents with relationships could typically be something for example the Norwegian Police Security Service (Politiets Sikkerhetstjeneste) could be interested in, to e.g. map out criminal networks, determining who has been communicating.

**7. Is there any functionality in particular Kripes would find beneficial?**

Preserved original filepaths of inserted objects is very interesting to have extracted, in order to e.g. know what computers the document has been edited in. Automating this process to perform bulk extraction would be beneficial, since unzipping a large amount of documents and manually extracting or "grep'ing" them out takes time. It is also interesting to determine any changes performed over time within a document. When what content was added, what was changed, has the document looked different earlier?

**8. What output format of an analysis tool would Kripes want?**

In most cases, simply textual output so we can handle it in any way we want. When it comes to showing relationships between entities, a graphical visualization would be beneficial, especially when faced with a large amount of information.

## F EnCase Forensic functionality

### F.1 EnCase metadata extraction

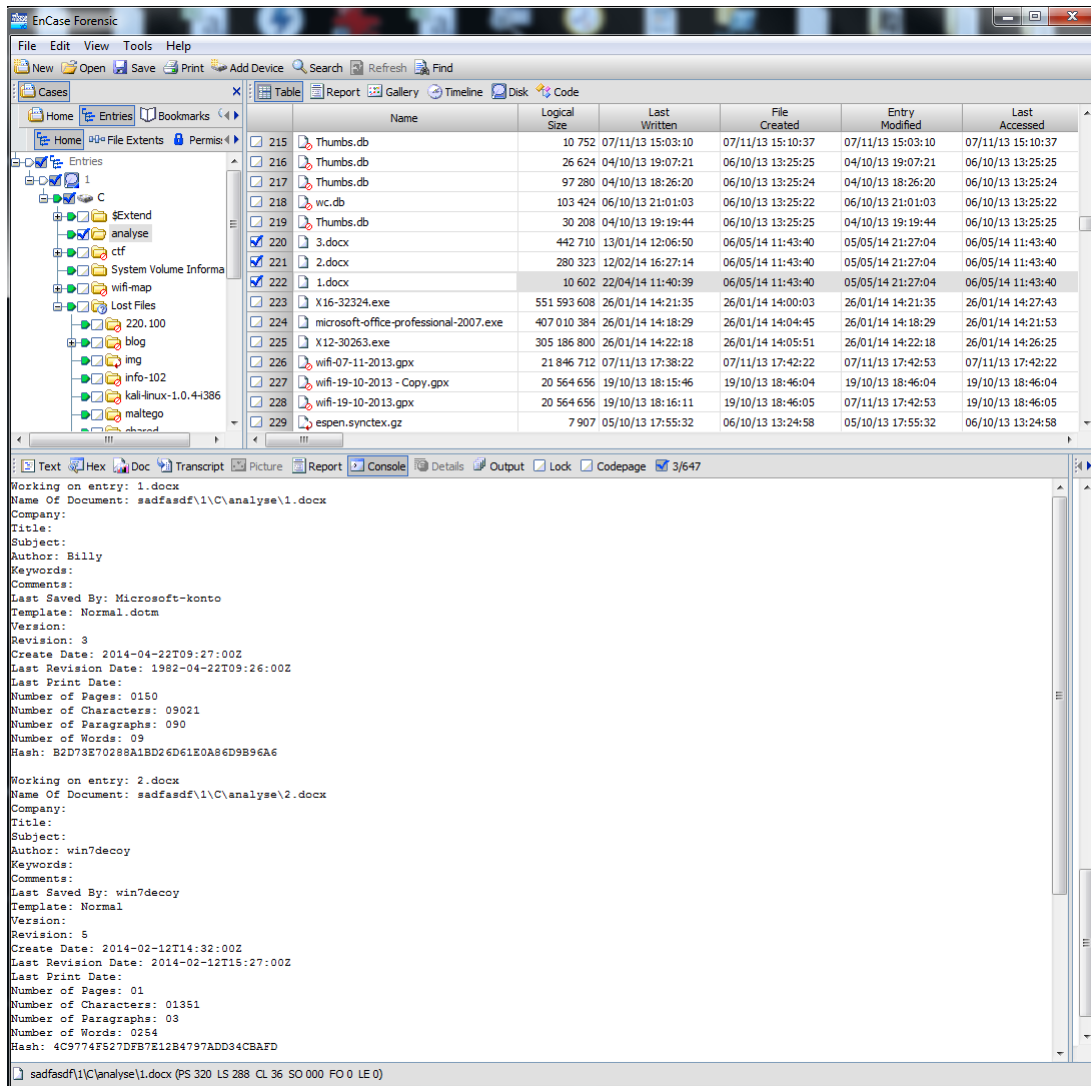


Figure 31: Result of EnCase extracting metadata from sample OOXML documents

## F.2 EnCase image information

Text Hex Doc Transcript Picture Report Console Details Output Lock

Name image1.JPG  
File Ext JPG  
File Type JPEG  
File Category Picture  
Description File  
Last Written 01/01/80 00:00:00  
Entry Modified 01/01/80 00:00:00  
Logical Size 245 437  
Initialized Size 245 437  
Physical Size 245 437  
Starting Extent 0Office 2007 Volume-B14481  
File Extents 1  
References 0  
Physical Location 14 481  
Physical Sector 0  
Evidence File 1  
File Identifier 0  
Code Page 0  
Full Path sadfasdf1\C\analyse\2.docx\Office 2007 Volume\word\media\image1.JPG

**Compressed Entry Metadata**

Encrypted No

**Compression**

Method NO COMPRESSION  
Supported Yes

**Hash Properties**

Name	Value
Hash Set	
Hash Category	

Picture



Figure 32: Result of EnCase extracting information from sample image

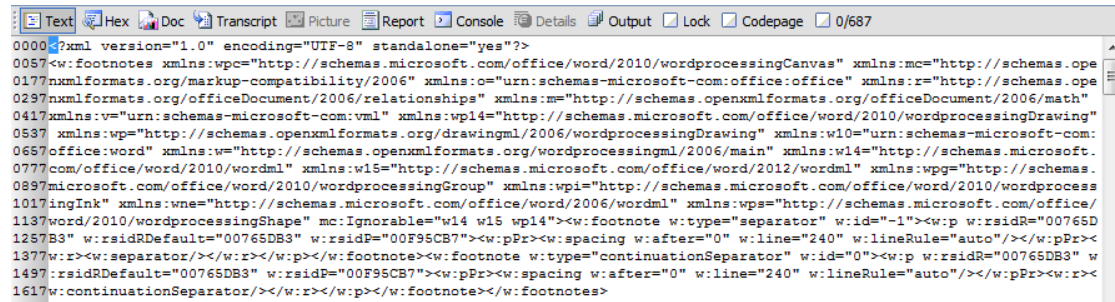
### F.3 EnScript output, extracting Exif metadata



Filnavn: image1.JPG  
Opprettet dato:  
Sist lagret dato: 01/01/80 00:00:00  
Modifisert dato: 01/01/80 00:00:00  
Aksessert dato:  
Slettet dato:  
Path:...word\media  
Beslagsid: 2.docx  
Størrelse: 245437  
Exif dato: 2014:02:12 15:59:43  
Exif program: 14.2.A.1.136\_9\_f500  
Exif modell: C6903  
Exif merke: Sony  
Exif tittel: N/A  
Exif artist: N/A  
Exif copyright: N/A  
Exif os: N/A  
Exif GPS: N 60 47.000 E 10 40.000  
Exif org dato: 2014:02:12 15:59:43  
Exif digital dato: 2014:02:12 15:59:43

Figure 33: Output of sample EnScript extracting Exif metadata from sample inserted image

## F.4 EnCase displaying XML



```

0000<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
0057<w:footnotes xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas" xmlns:mc="http://schemas.mpe
0177nxmlformats.org/markup-compatibility/2006" xmlns:o="urn:schemas-microsoft-com:office:office" xmlns:r="http://schemas.mpe
0297nxmlformats.org/officeDocument/2006/relationships" xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
0417xmlns:v="urn:schemas-microsoft-com:vml" xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing"
0537 xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing" xmlns:w10="urn:schemas-microsoft-com:
0657office:word" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main" xmlns:w14="http://schemas.microsoft.
0777com/office/word/2010/wordml" xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml" xmlns:wpg="http://schemas.
0897microsoft.com/office/word/2010/wordprocessingGroup" xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocess
1017ingInk" xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml" xmlns:wps="http://schemas.microsoft.com/office/
1137word/2010/wordprocessingShape" mc:Ignorable="w14 w15 wp14"><w:footnote w:type="separator" w:id="-1"><w:p w:rsidR="00765D
1257B3" w:rsidRDefault="00765DB3" w:rsidP="00F95CB7"><w:pPr><w:spacing w:after="0" w:line="240" w:lineRule="auto"/></w:pPr><
1377w:r><w:separator/></w:r></w:p></w:footnote><w:footnote w:type="continuationSeparator" w:id="0"><w:p w:rsidR="00765DB3" w
1497:rsidRDefault="00765DB3" w:rsidP="00F95CB7"><w:pPr><w:spacing w:after="0" w:line="240" w:lineRule="auto"/></w:pPr><w:r><
1617w:continuationSeparator/></w:r></w:p></w:footnote></w:footnotes>

```

Figure 34: EnCase displaying XML of sample file in document package

## F.5 EnCase showing manually altered values in *docProps/app.xml*

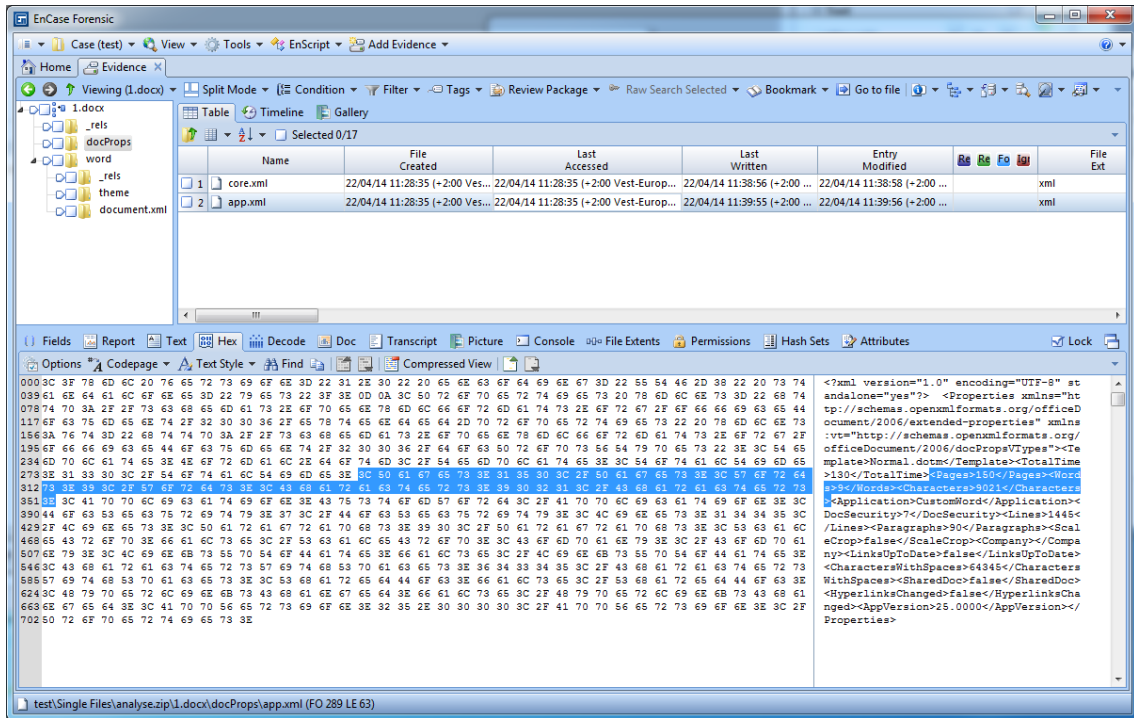


Figure 35: EnCase showing manually altered values in *docProps/app.xml*

## F.6 ExifTool output of sample image

Table 13: ExifTool output of sample image

Description	Value
File Name	image1.JPG
Directory	.
File Size	240 kB
File Modification Date/Time	1980:01:01
File Access Date/Time	2014:05:07 17:32:12+02:00
File Creation Date/Time	2014:05:07 17:32:12+02:00
File Permissions	rw-rw-rw-
File Type	JPEG
MIME Type	image/jpeg
JFIF Version	1.01
Exif Byte Order	Big-endian (Motorola, MM)
Make	Sony
Camera Model Name	C6903
Orientation	Horizontal (normal)
X Resolution	72
Y Resolution	72
Resolution Unit	inches
Software	14.2.A.1.136_9_f500
Modify Date	2014:02:12 15:59:43
Y Cb Cr Positioning	Centered
Exposure Time	1/200
F Number	2.0
ISO	50
Exif Version	0220
Date/Time Original	2014:02:12 15:59:43
Create Date	2014:02:12 15:59:43
Components Configuration	Y, Cb, Cr, -
Shutter Speed Value	1/199
Exposure Compensation	0

---

Metering Mode	Center-weighted average
Light Source	Unknown
Flash	Off, Did not fire
Focal Length	4.9 mm
Flashpix Version	0100
Color Space	sRGB
Exif Image Width	5248
Exif Image Height	3936
Interoperability Index	R98 - DCF basic file (sRGB)
Interoperability Version	0100
Custom Rendered	Normal
Exposure Mode	Auto
White Balance	Auto
Digital Zoom Ratio	1
Scene Capture Type	Standard
Subject Distance Range	Unknown
Offset Schema	12
GPS Version ID	2.2.0.0
GPS Latitude Ref	North
GPS Longitude Ref	East
GPS Altitude Ref	Above Sea Level
GPS Time Stamp	14:59:10
GPS Status	Measurement Active
GPS Map Datum	WGS-84
GPS Date Stamp	2014:02:12
Compression	JPEG (old-style)
Thumbnail Offset	3850
Thumbnail Length	2987
Image Width	1288
Image Height	430
Encoding Process	Baseline DCT, Huffman coding
Bits Per Sample	8
Color Components	3



Y Cb Cr Sub Sampling	YCbCr4:2:0 (2 2)
Aperture	2.0
GPS Altitude	251 m Above Sea Level
GPS Date/Time	2014:02:12
GPS Latitude	60 deg 47' 20.22" N
GPS Longitude	10 deg 40' 15.24" E
GPS Position	60 deg 47' 20.22" N, 10 deg 40' 15.24" E
Image Size	1288x430
Shutter Speed	1/200
Thumbnail Image	(Binary data 2987 bytes)
Focal Length	4.9 mm
Light Value	10.6

---

## G Forensic Toolkit (FTK) functionality

### G.1 FTK metadata extraction

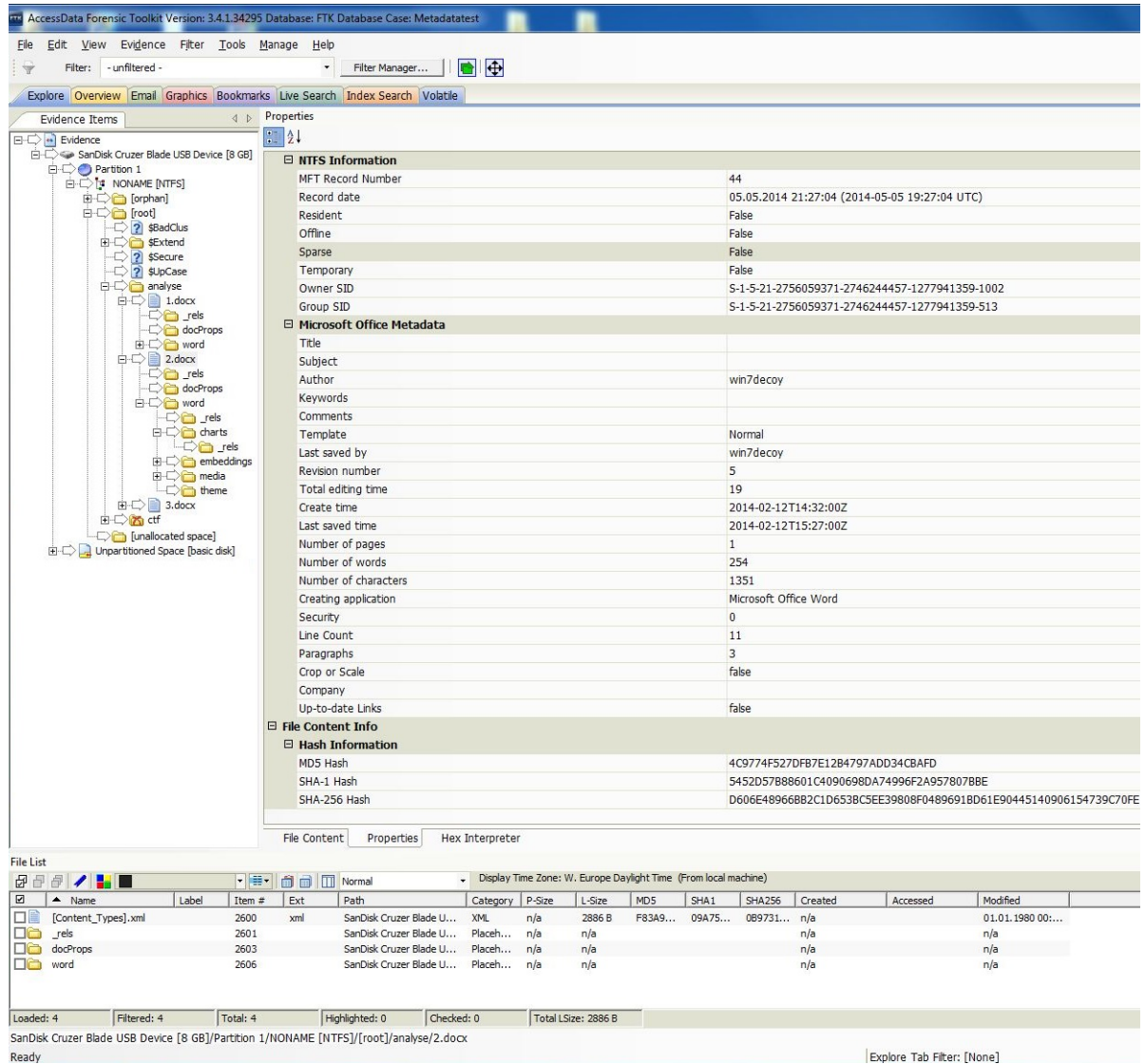


Figure 36: Screenshot of FTK extracting metadata from an OOXML document.

## G.2 FTK viewing individual XML file

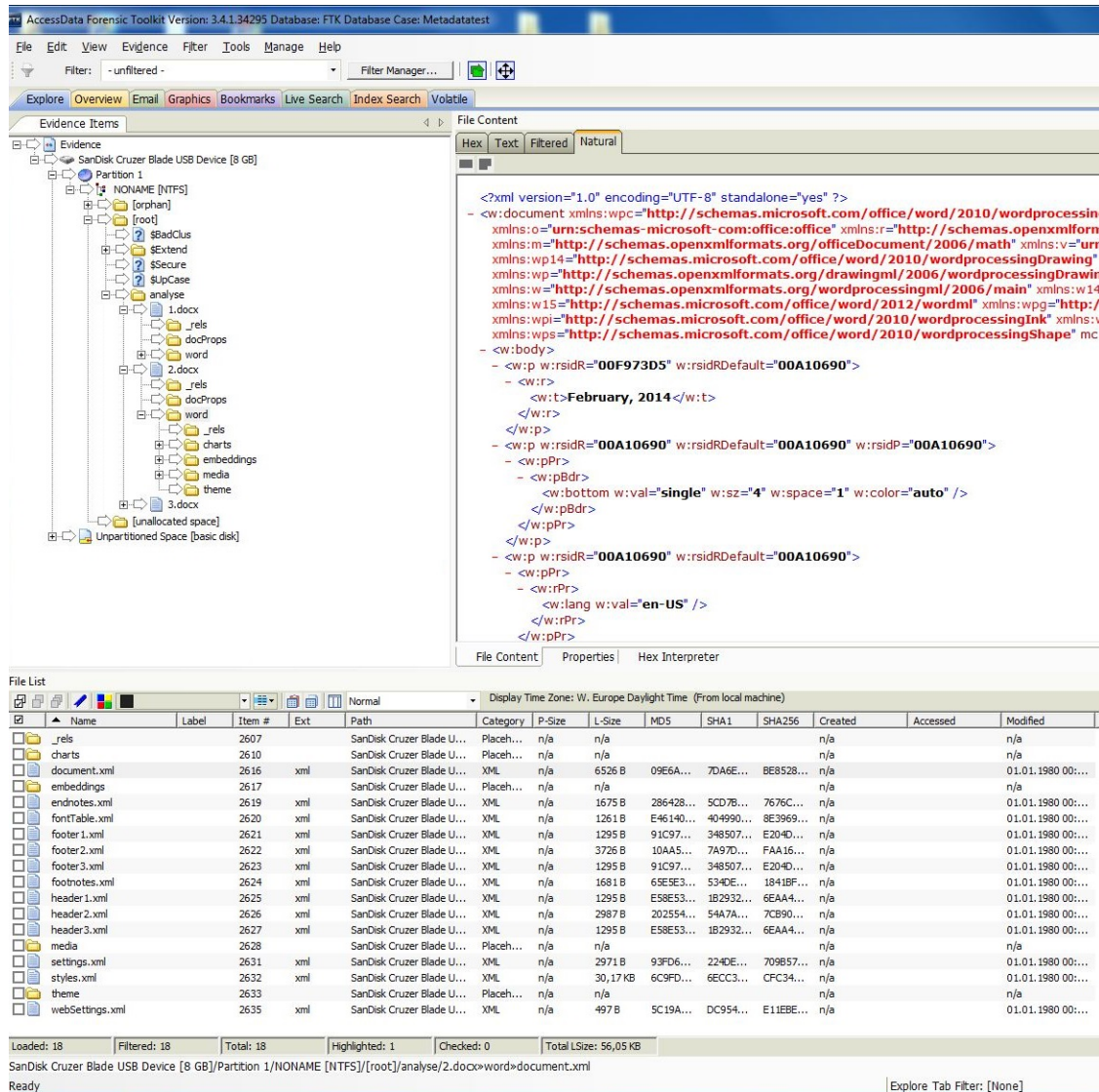


Figure 37: Screenshot of FTK viewing individual XML file.

### G.3 Sample document

#### G.3.1 Screenshot of document

Summary of R&D activities

February, 2014

As presented on the seminar in early January, the R&D department has the last six months been working on a revolutionary system for increasing the effective frequency of advertisement exposure, based on multi-platform user behavior advertisement spreading via Internet-connected devices. As we all know, smart phone popularity has rapidly increased since 2007, and is believed to already have taken over the market when it comes to accessing social media.

In short, our proposed system in its current form receives input from our smart phone application developing partners, who actively collect user information, application usage statistics and browsing habits from approximately 5.2 million users (as of late January, 2014). Our system processes this raw input and performs data mining in order to discover trends that might not be apparent at the first glance. By correlating our rapidly growing database of product information keywords with the user's web search keywords, we are able to expose the user with customized advertisements in the partnering applications, along with customized e-mail marketing for users that are significantly likely to purchase specific products.

The following graph provides a comparison of the current and the new approach, in terms of revenue (Y axis represents millions of Euro):

Year	Current approach (Millions of Euro)	New approach (Millions of Euro)
2010	12	15
2011	15	20
2012	10	35
2013	18	45
2014	20	72

Note: This is an internal document containing confidential information intended only for authorized employees. Any unauthorized distribution or access will lead to dismissal and legal actions.

SearchLinkGrid Solutions

Figure 38: Screenshot of sample document, with paragraph revisions marked.

### G.3.2 Paragraph creation revision identifiers

Listing G.1: XML showing the logical structure of textual content in an OOXML document

```
[+] <w:p w:rsidR="00D653F5" w:rsidRPr="00931AFE" w:rsidRDefault="00931AFE"
w:rsidP="00931AFE">
[+] <w:p w:rsidR="00931AFE" w:rsidRDefault="00931AFE" w:rsidP="00931AFE">
[+] <w:p w:rsidR="00931AFE" w:rsidRDefault="00931AFE" w:rsidP="00931AFE">
[+] <w:p w:rsidR="00931AFE" w:rsidRDefault="00931AFE" w:rsidP="00931AFE">
[+] <w:p w:rsidR="00E5398C" w:rsidRDefault="00E5398C" w:rsidP="00931AFE">
[+] <w:p w:rsidR="00E5398C" w:rsidRDefault="00E5398C" w:rsidP="00931AFE">
[+] <w:p w:rsidR="00A91664" w:rsidRDefault="00A91664" w:rsidP="00931AFE">
[+] <w:p w:rsidR="008A384C" w:rsidRDefault="008A384C" w:rsidP="00931AFE">
[+] <w:p w:rsidR="008A384C" w:rsidRDefault="008A384C" w:rsidP="00931AFE">
[+] <w:p w:rsidR="008A384C" w:rsidRDefault="008A384C" w:rsidP="008A384C">
[+] <w:p w:rsidR="008A384C" w:rsidRPr="008A384C" w:rsidRDefault="003A2162"
w:rsidP="00931AFE">
```

### G.3.3 docProps/app.xml

Listing G.2: XML showing the logical structure of textual content in an OOXML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Properties
  xmlns="http://schemas.openxmlformats.org/officeDocument/2006/extended-properties"
  xmlns:vt="http://schemas.openxmlformats.org/officeDocument/2006/docPropsVTypes">
  <Template>Normal</Template>
  <TotalTime>86</TotalTime>
  <Pages>1</Pages>
  <Words>247</Words>
  <Characters>1314</Characters>
  <Application>Microsoft Office Word</Application>
  <DocSecurity>0</DocSecurity>
  <Lines>10</Lines>
  <Paragraphs>3</Paragraphs>
  <ScaleCrop>>false</ScaleCrop>
  <HeadingPairs>
    <vt:vector size="2" baseType="variant">
      <vt:variant>
        <vt:lpstr>Title</vt:lpstr>
      </vt:variant>
      <vt:variant>
        <vt:i4>1</vt:i4>
      </vt:variant>
    </vt:vector>
  </HeadingPairs>
  <TitlesOfParts>
    <vt:vector size="1" baseType="lpstr">
      <vt:lpstr/>
    </vt:vector>
  </TitlesOfParts>
  <Company/>
  <LinksUpToDate>>false</LinksUpToDate>
  <CharactersWithSpaces>1558</CharactersWithSpaces>
  <SharedDoc>>false</SharedDoc>
  <HyperlinksChanged>>false</HyperlinksChanged>
  <AppVersion>15.0000</AppVersion>
</Properties>
```

### G.3.4 docProps/core.xml

Listing G.3: XML showing the logical structure of textual content in an OOXML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cp:coreProperties
  xmlns:cp="http://schemas.openxmlformats.org/package/2006/metadata/core-properties"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:dcmitype="http://purl.org/dc/dcmitype/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dc:title/>
  <dc:subject/>
  <dc:creator>win7decoy</dc:creator>
  <cp:keywords/>
  <dc:description/>
  <cp:lastModifiedBy>win7decoy</cp:lastModifiedBy>
  <cp:revision>6</cp:revision>
  <cp:lastPrinted>2014-02-12T12:39:00Z</cp:lastPrinted>
  <dcterms:created
    xsi:type="dcterms:W3CDTF">2014-02-12T11:13:00Z</dcterms:created>
  <dcterms:modified
    xsi:type="dcterms:W3CDTF">2014-02-12T13:40:00Z</dcterms:modified>
</cp:coreProperties>
```

### G.3.5 word/document.xml

Listing G.4: XML showing the logical structure of textual content in an OOXML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document
  xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
  xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
  xmlns:v="urn:schemas-microsoft-com:vml"
  xmlns:wp14="http://schemas.microsoft.com/office/word/2010/wordprocessingDrawing"
  xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
  xmlns:w10="urn:schemas-microsoft-com:office:word"
  xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
  xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
  xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
  xmlns:wpg="http://schemas.microsoft.com/office/word/2010/wordprocessingGroup"
  xmlns:wpi="http://schemas.microsoft.com/office/word/2010/wordprocessingInk"
  xmlns:wne="http://schemas.microsoft.com/office/word/2006/wordml"
  xmlns:wps="http://schemas.microsoft.com/office/word/2010/wordprocessingShape"
  mc:Ignorable="w14_w15_wp14">
<w:body>
  <w:p w:rsidR="00D653F5" w:rsidRPr="00931AFE" w:rsidRDefault="00931AFE"
    w:rsidP="00931AFE">
    <w:pPr>
      <w:pStyle w:val="Heading1"/>
      <w:rPr>
        <w:sz w:val="40"/>
        <w:szCs w:val="40"/>
        <w:lang w:val="en-US"/>
      </w:rPr>
    </w:pPr>
    <w:r w:rsidRPr="00931AFE">
      <w:rPr>
        <w:sz w:val="40"/>
        <w:szCs w:val="40"/>
        <w:lang w:val="en-US"/>
      </w:rPr>
      <w:t>Summary of R&D activities</w:t>
    </w:r>
  </w:p>
  <w:p w:rsidR="00931AFE" w:rsidRDefault="00931AFE" w:rsidP="00931AFE">
    <w:pPr>
      <w:rPr>
        <w:lang w:val="en-US"/>
      </w:rPr>
    </w:pPr>
    <w:r>
```



```

    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
    <w:t>February, 2014</w:t>
  </w:r>
</w:p>
<w:p w:rsidR="00931AFE" w:rsidRDefault="00931AFE" w:rsidP="00931AFE">
  <w:pPr>
    <w:pBdr>
      <w:bottom w:val="single" w:sz="4" w:space="1" w:color="auto"/>
    </w:pBdr>
    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
  </w:pPr>
</w:p>
<w:p w:rsidR="00931AFE" w:rsidRDefault="00931AFE" w:rsidP="00931AFE">
  <w:pPr>
    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
  </w:pPr>
  <w:r>
    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
    <w:t>As presented on the seminar in early January, the R&D
      department has the last six months been working on a revolutionary
      system for increasing the effective frequency of advertisement
      exposure, based on multi-platform user behavior advertisement
      spreading via Internet-connected devices. As we all know, smart
      phone popularity has rapidly increased since 2007, and is believed
      to already have taken over the marked when it comes to accessing
      social media.</w:t>
  </w:r>
</w:p>
<w:p w:rsidR="00E5398C" w:rsidRDefault="00E5398C" w:rsidP="00931AFE">
  <w:pPr>
    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
  </w:pPr>
  <w:r>
    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
    <w:t>In short, our proposed system in its current form receives input
      from our smart phone application developing partners, who actively
      collect user information, application usage statistics and browsing

```

habits from approximately 5.2 million users (as of late January, 2014). Our system processes this raw input and performs data mining in order to discover trends that might not be apparent at the first glance. By correlating our rapidly growing database of product information keywords with the user's web search keywords, we are able to expose the user with customized advertisements in the partnering applications, along with customized e-mail marketing for users that are significantly likely to purchase specific products.

The following graph provides a comparison of the current and the new approach, in terms of revenue

(Y axis represents millions of Euro)

:

noProof

nb-NO

drawing

```

<wp:inline distT="0" distB="0" distL="0" distR="0">
  <wp:extent cx="5486400" cy="3200400"/>
  <wp:effectExtent l="0" t="0" r="0" b="0"/>
  <wp:docPr id="1" name="Chart_1"/>
  <wp:cNvGraphicFramePr/>
  <a:graphic
    xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
    <a:graphicData
      uri="http://schemas.openxmlformats.org/drawingml/2006/chart">
      <c:chart
        xmlns:c="http://schemas.openxmlformats.org/drawingml/2006/chart"
        xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
        r:id="rId6"/>
      </a:graphicData>
    </a:graphic>
  </wp:inline>
</w:drawing>
</w:r>
</w:p>
<w:p w:rsidR="008A384C" w:rsidRDefault="008A384C" w:rsidP="00931AFE">
  <w:pPr>
    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
  </w:pPr>
</w:p>
<w:p w:rsidR="008A384C" w:rsidRDefault="008A384C" w:rsidP="00931AFE">
  <w:pPr>
    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
  </w:pPr>
  <w:bookmarkStart w:id="0" w:name="_GoBack"/>
  <w:bookmarkEnd w:id="0"/>
</w:p>
<w:p w:rsidR="008A384C" w:rsidRDefault="008A384C" w:rsidP="008A384C">
  <w:pPr>
    <w:pBdr>
      <w:bottom w:val="single" w:sz="4" w:space="1" w:color="auto"/>
    </w:pBdr>
    <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
  </w:pPr>
</w:p>
<w:p w:rsidR="008A384C" w:rsidRPr="008A384C" w:rsidRDefault="003A2162"
  w:rsidP="00931AFE">
  <w:pPr>
    <w:rPr>

```

```

    <w:i/>
    <w:lang w:val="en-US"/>
  </w:rPr>
</w:pPr>
<w:r>
  <w:rPr>
    <w:i/>
    <w:noProof/>
    <w:lang w:eastAsia="nb-NO"/>
  </w:rPr>
  <w:drawing>
    <wp:anchor distT="0" distB="0" distL="114300" distR="114300"
      simplePos="0" relativeHeight="251658240" behindDoc="1" locked="0"
      layoutInCell="1" allowOverlap="1" wp14:anchorId="0B52682B"
      wp14:editId="7372E287">
      <wp:simplePos x="0" y="0"/>
      <wp:positionH relativeFrom="margin">
        <wp:align>right</wp:align>
      </wp:positionH>
      <wp:positionV relativeFrom="paragraph">
        <wp:posOffset>538518</wp:posOffset>
      </wp:positionV>
      <wp:extent cx="1738800" cy="367200"/>
      <wp:effectExtent l="0" t="0" r="0" b="0"/>
      <wp:wrapNone/>
      <wp:docPr id="2" name="Picture_2"/>
      <wp:cNvGraphicFramePr>
        <a:graphicFrameLocks
          xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
          noChangeAspect="1"/>
      </wp:cNvGraphicFramePr>
      <a:graphic
        xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
        <a:graphicData
          uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
          <pic:pic
            xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
            <pic:nvPicPr>
              <pic:cNvPr id="2" name="searchlinkgridlogo.png"/>
              <pic:cNvPicPr/>
            </pic:nvPicPr>
            <pic:blipFill>
              <a:blip r:embed="rId7">
                <a:extLst>
                  <a:ext uri="{28A0092B-C50C-407E-A947-70E740481C1C}">
                    <a14:useLocalDpi
                      xmlns:a14="http://schemas.microsoft.com/office/drawing/2010/main"
                      val="0"/>
                  </a:ext>
                </a:extLst>
              </a:blip>
            </pic:blipFill>
          </a:graphicData>
        </a:graphic>
      </wp:cNvGraphicFramePr>
    </wp:drawing>
  </w:r>
</w:p>

```

```

        </a:extLst>
        </a:blip>
        <a:stretch>
          <a:fillRect/>
        </a:stretch>
      </pic:blipFill>
    <pic:spPr>
      <a:xfrm>
        <a:off x="0" y="0"/>
        <a:ext cx="1738800" cy="367200"/>
      </a:xfrm>
      <a:prstGeom prst="rect">
        <a:avLst/>
      </a:prstGeom>
    </pic:spPr>
  </pic:pic>
</a:graphicData>
</a:graphic>
<wp14:sizeRelH relativeFrom="margin">
  <wp14:pctWidth>0</wp14:pctWidth>
</wp14:sizeRelH>
<wp14:sizeRelV relativeFrom="margin">
  <wp14:pctHeight>0</wp14:pctHeight>
</wp14:sizeRelV>
</wp:anchor>
</w:drawing>
</w:r>
<w:r w:rsidR="008A384C">
  <w:rPr>
    <w:i/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t>Note: This is an internal document containing confidential
    information intended only for authorized employees. Any
    unauthorized distribution or access will lead to dismissal and
    legal actions.</w:t>
</w:r>
<w:r w:rsidRPr="003A2162">
  <w:rPr>
    <w:i/>
    <w:noProof/>
    <w:lang w:val="en-US" w:eastAsia="nb-NO"/>
  </w:rPr>
  <w:t xml:space="preserve"/>
</w:r>
</w:p>
<w:sectPr w:rsidR="008A384C" w:rsidRPr="008A384C">
  <w:headerReference w:type="even" r:id="rId8"/>
  <w:headerReference w:type="default" r:id="rId9"/>

```

```
<w:footerReference w:type="even" r:id="rId10"/>
<w:footerReference w:type="default" r:id="rId11"/>
<w:headerReference w:type="first" r:id="rId12"/>
<w:footerReference w:type="first" r:id="rId13"/>
<w:pgSz w:w="11906" w:h="16838"/>
<w:pgMar w:top="1417" w:right="1417" w:bottom="1417" w:left="1417"
  w:header="708" w:footer="708" w:gutter="0"/>
<w:cols w:space="708"/>
<w:docGrid w:linePitch="360"/>
</w:sectPr>
</w:body>
</w:document>
```

---

### G.3.6 word/settings.xml

Listing G.5: XML showing the logical structure of textual content in an OOXML document

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:settings xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships"
  xmlns:m="http://schemas.openxmlformats.org/officeDocument/2006/math"
  xmlns:v="urn:schemas-microsoft-com:vml"
  xmlns:w10="urn:schemas-microsoft-com:office:word"
  xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main"
  xmlns:w14="http://schemas.microsoft.com/office/word/2010/wordml"
  xmlns:w15="http://schemas.microsoft.com/office/word/2012/wordml"
  xmlns:sl="http://schemas.openxmlformats.org/schemaLibrary/2006/main"
  mc:Ignorable="w14_w15">
  <w:zoom w:percent="70"/>
  <w:proofState w:spelling="clean" w:grammar="clean"/>
  <w:defaultTabStop w:val="708"/>
  <w:hyphenationZone w:val="425"/>
  <w:characterSpacingControl w:val="doNotCompress"/>
  <w:hdrShapeDefaults>
    <o:shapedefaults v:ext="edit" spidmax="2050"/>
    <o:shapelayout v:ext="edit">
      <o:idmap v:ext="edit" data="2"/>
    </o:shapelayout>
  </w:hdrShapeDefaults>
  <w:footnotePr>
    <w:footnote w:id="-1"/>
    <w:footnote w:id="0"/>
  </w:footnotePr>
  <w:endnotePr>
    <w:endnote w:id="-1"/>
    <w:endnote w:id="0"/>
  </w:endnotePr>
  <w:compat>
    <w:compatSetting w:name="compatibilityMode"
      w:uri="http://schemas.microsoft.com/office/word" w:val="15"/>
    <w:compatSetting w:name="overrideTableStyleFontSizeAndJustification"
      w:uri="http://schemas.microsoft.com/office/word" w:val="1"/>
    <w:compatSetting w:name="enableOpenTypeFeatures"
      w:uri="http://schemas.microsoft.com/office/word" w:val="1"/>
    <w:compatSetting w:name="doNotFlipMirrorIndents"
      w:uri="http://schemas.microsoft.com/office/word" w:val="1"/>
    <w:compatSetting w:name="differentiateMultirowTableHeaders"
      w:uri="http://schemas.microsoft.com/office/word" w:val="1"/>
  </w:compat>
  <w:rsids>
    <w:rsidRoot w:val="00BB122F"/>
    <w:rsid w:val="003A2162"/>
  </w:rsids>
</w:settings>
```

---

```

    <w:rsid w:val="008A384C"/>
    <w:rsid w:val="00931AFE"/>
    <w:rsid w:val="00A91664"/>
    <w:rsid w:val="00BB122F"/>
    <w:rsid w:val="00CD3453"/>
    <w:rsid w:val="00D653F5"/>
    <w:rsid w:val="00DB04C1"/>
    <w:rsid w:val="00DB47D6"/>
    <w:rsid w:val="00E5398C"/>
  </w:rsids>
  <m:mathPr>
    <m:mathFont m:val="Cambria_Math"/>
    <m:brkBin m:val="before"/>
    <m:brkBinSub m:val="--"/>
    <m:smallFrac m:val="0"/>
    <m:dispDef/>
    <m:lMargin m:val="0"/>
    <m:rMargin m:val="0"/>
    <m:defJc m:val="centerGroup"/>
    <m:wrapIndent m:val="1440"/>
    <m:intLim m:val="subSup"/>
    <m:naryLim m:val="undOvr"/>
  </m:mathPr>
  <w:themeFontLang w:val="nb-NO"/>
  <w:clrSchemeMapping w:bg1="light1" w:t1="dark1" w:bg2="light2"
    w:t2="dark2" w:accent1="accent1" w:accent2="accent2" w:accent3="accent3"
    w:accent4="accent4" w:accent5="accent5" w:accent6="accent6"
    w:hyperlink="hyperlink" w:followedHyperlink="followedHyperlink"/>
  <w:shapeDefaults>
    <o:shapedefaults v:ext="edit" spidmax="2050"/>
    <o:shapelayout v:ext="edit">
      <o:idmap v:ext="edit" data="1"/>
    </o:shapelayout>
  </w:shapeDefaults>
  <w:decimalSymbol w:val=","/>
  <w:listSeparator w:val=";"/>
  <w15:chartTrackingRefBased/>
  <w15:docId w15:val="{49B638EA-E327-461B-96AA-3676BCB565F0}"/>
</w:settings>

```

---



## H Change tracking example

### H.1 Screenshot of document edited with change tracking enabled

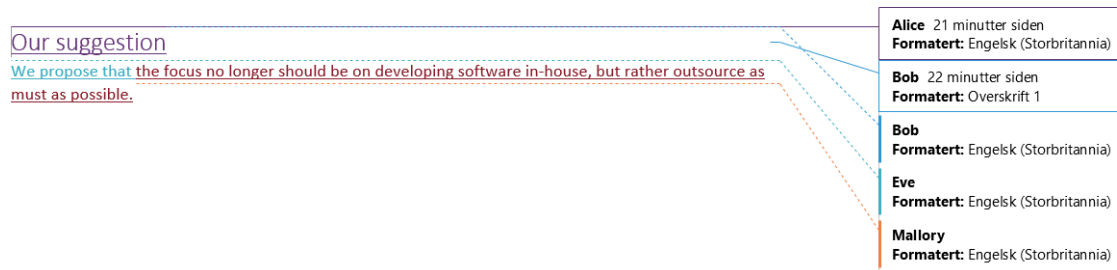


Figure 39: Screenshot of document edited with change tracking enabled

### H.2 XML of document edited with change tracking enabled

Listing H.1: 123

```

<w:ins w:id="4" w:author="Alice" w:date="2014-05-29T15:14:00Z">
  <w:r w:rsidRPr="00E71DAE">
    <w:rPr>
      <w:lang w:val="en-GB"/>
      <w:rPrChange w:id="5" w:author="Alice"
        w:date="2014-05-29T15:15:00Z">
        <w:rPr>
          <w:lang w:val="nb-NO"/>
        </w:rPr>
      </w:rPrChange>
    </w:rPr>
    <w:t>Our suggestion</w:t>
  </w:r>
</w:ins>

```

### H.3 Uniqueness of revision identifiers result table

Table 14: Classification table; description of each number in Table 15

Number	Description
1	One common rsidRDefault value; no common content at all; different authors; different companies; different Office versions; different year = no connection
2	Common root rsid; many shared rsids; common content = clear connection; one document is based on the other
3	Shared run rsidRPr values; common content = clear connection; content is likely copied from one document to the other
4	One common rsidR/rsidP/rsidRDefault value; no common content at all; different authors; different companies; different Office versions; different year = no connection
5	Common root rsid; several shared rsids; common content = clear connection; one document is based on the other
6	Last paragraph is the same = clear connection
7	Common template; common company = clear connection
8	Shared root rsid; same author = clear connection; documents likely written simultaneously
9	Common footer; common company = clear connection
10	Many shared rsids; common content = clear connection; one document is based on the other

Table 15: Result of uniqueness of revision identifiers experiment. Column name “CI” refers to “Classification” (see Table 14); “FP” refers to “false positive”

#	Document A SHA1 hash	Document B SHA1 hash	CI	FP?
1	7232d6caebd461569f67f6179e0887d946c73f9	21cea4c0d35bbc419b6cb4d06c79951d1522dd5d	1	<u>Yes</u>
2	651d10bffa2b98a8dad6ccf59c21b2852c80d6f	928ec22f53affa441a24fbc724dafec25737b6	2	No
3	0f1861cdb86784afb5964118d3918a7c92dfe2bd	e81610efeb6757799233d3b802e409decfaba0c0	2	No
4	c30b27d3945447b1c9c705e0bac279187b3cb925	ff871934aa3968aeced9a442f8f84a9a61d79ef9	2	No
5	450c6fe13355742c2f9712d74ecb8767c895320a	b563f1922d9f12cf1e0befe314677c0438aba909	2	No
6	4331958aef46855b17d1620b6dde9b2018e8333f	90708418358abb7c4f6dc836a3eece28236d8b24	2	No
7	6ee167ad8e800263d44d38b245a70b934482a356	8e5d483cd39a5593957726381929fbd186d2687b	2	No
8	71658ffc646c037b4e3491b5ef0715a2451d219f	d0fa60f87b091048414376228b0cc8141ab94459	3	No
9	d4cb36bbfafbb8f4df9f7922326e6ef39e94a4d2	76c54762a0565ef85884be87072c855109a1c915	3	No
10	97f228fa5737402fb1ff01f297f7abaabc662f0d	c85cf79bc003107014b84747b6169a4105aa2091	2	No

11	993bdb2c1ff462b2757a12885b1d279db0436023	bc92ad1c887315d0c3f0529fd3f560eb6534626e	2	No
12	df635579f3c4282d67dc7cf095991c7b12327ab5	423c99e10c49522f9b81ddd679cf76bed56d7f8d	2	No
13	9a397545d00696120c4a3a3452fbfae048ec1390	7508a7a612d30cd1765aea16e76a253f90b0fe9e	2	No
14	232eb11072bf76ca3a009d19d897b1d37f0bc321	7508a7a612d30cd1765aea16e76a253f90b0fe9e	4	Yes
15	71a9b7b8b5f9853ed24b87200738614af1f19013	3cca95d8a911bf90b9fc3505b279afc0a94fdf30	2	No
16	630963790a60fee0d3585d356365867c551bd871	cc61abb6e1b5851e8763fabea0839f77bf457bbc	2	No
17	5de9922c987f43be5d93454bf8082fd3b87c5700	b374ef5ebec46bd8106e1841b0feec38512085b2	2	No
18	1d826b52e0e19f3d7dc5182f84d89c0b6dedcd4	33020b4ce45352776f1af20f0d11ba23b96c9f37	2	No
19	66e5e0228bab3a08498413a6485f9db2f2bbfa07	7d2c935db84bd7c3fd1480f753485f57e6ce76ba	2	No
20	1c846675ada1c2ff98f78e65958a9c48d95dcf5f	558d007e5f58246a4aac320b889007e542b49123	5	No
21	f4b1dce1032fe0c8a66060a53a4d7f9a395a1018	3eda41dfba2774469fb5349c11810b1cfba1ed23	5	No
22	8e24462f145c27f4a9b5118c52a1018f698a41bc	7287fbff4fe0c735f2bae3dcc1aef623d988f108	5	No
23	4651c8090d06eeecd80c6881093e111613c68d136	f188407f61213e2f1f0b336ec6d119af8b5c477e	5	No
24	7819bfb87d3d4b9af0d77dbbcaaea756bd434536	c1c5d7f027a7ee8a60dec8e0e4d105c456a9ae14	5	No
25	00f664c6b1aeda86586e0db91988a2ea36395adf	71ca8561090587bd479475aa9a5b62152ee874c3	5	No
26	ea1087dbf96f888b06db93f32d29361f3833ab3f	9e3f2037ebb1ff0a2b911a2e39bdd560f4ad8aee	2	No
27	0ce4f0db52b8c44623214fff5f7533f9fe13b668	07fb4d6a1b6369918d23e27e34702f751467c62e	2	No
28	8820ef608f094f7921e28e98adf0c755b0965a33	2b9646abfb0b0b2559ade24700501ddc8a00261b	5	No
29	c4bb4265073441d371bb1f9a220bb3b14d39b3bc	495b49b507c6ee5ea9eaa6538c537e7bb3884f16	5	No
30	64b112dfd3a3470c335a1ada5fc4a87c52c93d67	bb50c2eeaf1080690530d057d680a4ac22345d96	5	No
31	949160588f0ece580932ef6e5507664ff957b314	1eb2d514033004ac515f499c0bc4d18eca75e29e	2	No
32	89df92f2349fdbbe139e9555b606c6bbbed20b9b0	5d17193359b6e8b3be6309a858343f185edf5625	5	No
33	d3a795e177db6a0c275c37ccc6ea25ca4b4a8d93	1bb9c630308106398fe26cb319542b53374f1637	2	No
34	f3e54a23dc5bb210c4df27ae43d608a4466424ed	de07068a18b1026c1e52d016e5a3a10d39fe8747	5	No
35	afbeee0eab394812f36e47c819d96d07cfeaa9c2	369a634ce7a56b489bf57bdc49b2f51516f83673	5	No
36	6154d1c0f008ab5678455f4f4b26b0108e613d65	bfc2e36a4692787a6ae4c2c9a40a08f5e459332e	5	No
37	5880b7f4711818e431fcbebf67b6c32d3d8a086	20e002776ba417bde55514417f888e35edbf8b8e	2	No
38	e17055c449bad7f4bd2a7e7192103be754768df2	fa2aaa9c81d33d2938cc4c7e7af3a671317c6152	6	No
39	23020b8d0f013bde752b8bde753f4088bdbf2fd9	64d15e0c8b70b0365f31ca9bb4d17f48ab5be8ec	2	No
40	7597ef4b3955a0117b81374b598cb9abf5f314e1	d9105bc40aa42ade54bd45fc4369075aef52bbde	2	No
41	70f40d44fd0c54c37ae5d656ba540c04c6e4e00c	2070928e609f33dfaafca81d97a985456a895312	7	No
42	1b4d4fc5001de69068c434c05ed88c71ab36c0dc	fdcb57da1357029c4d238ca1b0773caeb22ba3d4	3	No
43	a7aa32e6895b15753658a4c0fb23f1ed8affedfc	7459c08960c287201a5b42868aacbf28ef81673	3	No
44	5c1555fd176b602aecc0a8acbcacb14586a994cf	5b1c07e54e49b16a31c4f21a13bddde94f66ce5c	10	No
45	7b041ad60e155f1e471d84176d63d16d61174631	d6b05bb8a15974a4aecdd275aefec9e0035582445	3	No
46	af1fa5ad4d2cd1c1ce014b63c0abab24c6e64936	cad3f185bda21a715410a7791007d8ff57d1c580	3	No
47	ce1c5a596cc1b524e744f0507eff3234400732ef	444935747b51a79e191235cc6a52320e054f69e8	10	No
48	289787ed6de66efcf682803507db860bc9d78cc9	677c8c1737906bda53bea858e08861ccd6a3f14	3	No
49	dde87b0b13576b9834a03885dd7970985bdd00db	4640af9ab746a18790f51688b71340c372454d52	3	No

50	049f36d22071ad3b7266ae21b4f39e8c07d746dc	3b3953f1e4b3653e75e39f0be9907963b8e189dd	10	No
51	2b5c967ea392540dd3d256e56865509691c7444a	c59a55fbb424b159fce78c34471646e22ed9d412	3	No
52	1aa0117a9d95ad3c1083ae21003edd6f96ec0902	9b8a75886a81bbe215d48bdb28b8b86a6b317b7c	10	No
53	a602b9195b92f6102ee0aef29be055dd9075ba22	5355f6f17b0be9383e873c4f4b7d7f8f4ac9e777	3	No
54	d9a6f3596193b9460ea535143855dc50ce476b01	311759785e8652a04e50bac58eb1e7e6bf39d3f8	10	No
55	18cf92ea0994071ed33fd2179523bb588d1a42d0	7b2445b2dd6e31f25e0cedb13e61de1979092f3d	10	No
56	1f42fc6037e42fa9295d31187578464b714179f0	0fe698c619ee722957f8ee452cf257926653d0a9	10	No
57	86d008e81d0cfd89f32d425f2f95c4af627fbd0	795aed67aee6bd9ff97f54a01a2c641d5b413f98	10	No
58	540f51b7025106c68f91f0c9ac99ff7851a34caa	b70f5be88457ec86ac3fe3528ca7269847225b5a	7	No
59	d87360b7ac5113366df19898a047b25ab929f209	0d170d7fdbc1e65af192b56bd38a6abf7d4383c4	10	No
60	01807080bd5c5e87cb6bf758ff3ab15f399cb232	8d0e28d158f7d05aaf96f4a7dabce1593efca267	10	No
61	364a8d01484307f671c575d6be583f361a5c3abc	8d0e28d158f7d05aaf96f4a7dabce1593efca267	3	No
62	0fc86edb1dfbc7e454ca3b5b1e5135c3e7e8b126	5610f65c32c0aed705562e5f82d968b04c2e0d9a	3	No
63	d3b47242cb3f0ac1472beaa9ffc5d75aa9496ebf	dd18aacc79dc313eb314caabc35d695ec8f2aa4b	10	No
64	7d05720a40552c42e85c46ec554d4ad3e626c545	38d5051d24a2a6e1b07741a0e842a48be4a00f2b	3	No
65	824c9a122478625b49b39fb129c5a09c46fbe141	0e3c1af516d918584b07ad00ed599a6b69522662	10	No
66	a3e771d2deb658abf604bd91503415ef905cdb0f	e9df8f25bbaa217f41af0aae611f34b392de8b21	3	No
67	49d90f4b3f045378f22ae444b2a9d40704bb1af5	8314b6f57761cd9ce153c78889500c7085a6ed9c	3	No
68	36900e7e814b4ba64cf9cc486b87767930d6a178	468ce96341ea32f290846644d8b7bf4f4450d69d	10	No
69	04585770b49201ba01ffca7de94dc78a7f879117	4ce039de95c289b583fd16d583ccd73919aa6468	10	No
70	a9e6fcc44118c90cdd94ed937b92c67697a85de6	ff995b39f1ba9b841d092392a5f9678ee9288f7b	10	No
71	874ca1bfe328860a54948f7249a2c191836a0fec	4a0783d5d61d40b98fa0da136f867906b5334723	10	No
72	5002ce804d87e1f613f1897ed810b66b11c9df85	21a9af88867118af8dc79af8117d220df1dc033d	10	No
73	712491fc7536a941ecc4d489b50e16eda94085c2	6d0865e715494dd97829c9f96129a71488bcee5f	3	No
74	190b4285c7d06442479a862d52732fefa817cd23	5dd2ecc24c49d655d53d9f54e4b1a678b66528d	8	No
75	500a23ad7846586bc230d85a0b08ef5c314db668	892d6b758db89c92dea6390dc00ceae6f05a74b2	10	No
76	e789a442dfa9f8f4052f5eba0938fb8283cea4b8	50d6f22d6c42c8c2df250e51d6ff5dbcc982d75f	10	No
77	3e99e368a58c27256bbd5e5088f70533e1103513	94ce78b58a7db2ef5ec0f41dc0c91be722ac1d96	10	No
78	9d2d915eb94b2b8ccab81a6d4c4afd2b82196297	3a5c24506c8b2d06456e65f60d4a47ba4980a948	10	No
79	a67d6683ae7fe56700013a28741a864bda53ca8d	aaa3900a8275ecd654d4918279ce80055612f33b	10	No
80	71b4e695bba8f01cd7c83ad46f6e88c3d05dbd7f	57f2fbd347f5475492884b175fb0c5180794dfdb	3	No
81	303dafb79466316943154126d8575db78e24b853	80768f3db0a41cc7628ec6d7a2e1e7aad48271ef	10	No
82	4e03cbfe3d85a024afcb7cfabfa45b70261f4934	754de813a446dff59147bddb6dadff44744d86b9	10	No
83	ebd1d391cdcf0e87d0d107a6214542bc19b627de	2eff9b2e1d574bc8af13f7194d1cec9f6ad25f79	3	No
84	ce507e5060b3e1827acba894cf2b8ab3ca3338d5	30c68bd121773691b26d888286cc4ea16d9536d1	10	No
85	ef434e57ac3c9fe96e63d7957a7d1e113179a5df	1b043c3bf3763253a52e7d304652c020614c96ae	10	No
86	a5755ca3abd3cf1c27456a400c834d5876946628	7ad7ac27099c344df1cb9370919a44b3a852d76d	10	No
87	1b68c752cea1f4ad189b7e60beb886103802df58	5790d726d75d85a8942ad66d3d37c4eec7aacafd	10	No
88	eba7f8c9d506cb63381deb02ab1a9505056c4c0a	2b4705d105c91adac90d0db82ea7d793037e2532	10	No

89	5c240206577b62d921f7a47625b8a8e961d2407d	ea3493e8b2f0376b8ebac056bb97b1214a2172f5	10	No
90	e65e82ccb78aeeb42596caf08722386385f9c929	901360b01ba93b8ea51342dbd8747ff8a27bd3c9	10	No
91	057c1f4b2a41e5f2dde19be8608ca5048edff845	ae36c469e758ef647e4d4bc1e46d9eabc54307e2	9	No
92	869b28768bcdce9b7d5a96e94887b562ba7caa67	5d46edaa9ce554b8b5abc67dcd5396dd941fd312	8	No
93	1b7ae4ed96fc852fb5077681d9b0a4aea0c40684	7c4c8e60cbc6d4be6c54837eaa76b2ad43eda798	10	No
94	fd6787db7139f9097317e9be4b6dec6e6af18c	893517bf69b3272bea32bde5e22ab233d507f0ce	10	No
95	cc06bfe7332a2c6b0d2cbee04ae1aab567d76c4	1289892ceb75b39ab7c9e7fde9074a3226f53775	8	No
96	a89bfce9611e4d90d310314ce0b006e6e22905e8	8467d43430d2a36e3903ad94ed76366a1fc6fd6a	10	No
97	12caceb15966542b9e64ade922ed1cc6a2948c61	325a5906d2839146e39284d4aa1daa82efe67001	10	No
98	1710e44beacfe13bab92e357fffd008b1b5ca841	65f8d295dd579caf7f72d4972f2ee4f413b825fd	10	No
99	cb5d75b5fc61994acba4c39c62e93190c10f432b	67b46dadbeae74477b7cf38b593d96a129261c28	10	No
100	52b13e8c7bad274ff2771490906dba8a74fbae4d	9cd06c2df25ee624245b1fe149c1df0add93464	10	No

## H.4 Application information extracted from data set

Table 16: Application information extracted from data set.

Application	AppVersion	Number of files	Percent
Microsoft Office Word	14.0000	34122	≈ 44.85 %
Microsoft Office Word	12.0000	32308	≈ 42.465 %
Microsoft Office Word	15.0000	1723	≈ 2.265 %
Microsoft Office Word	00.0001	4	≈ 0.005 %
Microsoft Office Word	16.0000	1	≈ 0.001 %
Microsoft Office Word	14.000	1	≈ 0.001 %
Microsoft Word 12.0.0	12.0000	84	≈ 0.110 %
Microsoft Word 12.1.0	12.0256	58	≈ 0.076 %
Microsoft Word 12.0.1	12.0001	1	≈ 0.001 %
Microsoft Word 12.1.1	12.0257	5	≈ 0.006 %
Microsoft Word 12.1.2	12.0258	3	≈ 0.003 %
Microsoft Office Outlook	12.0000	2113	≈ 2.777 %
Microsoft Macintosh Word	14.0000	4159	≈ 5.467 %
Microsoft Macintosh Word	12.0000	1436	≈ 1.887 %
Google Docs	Not specified	59	≈ 0.078 %
LibreOffice	3.5 \$Linux_X86_64	1	≈ 0.001 %
LibreOffice	3.5 \$Windows_x86	1	≈ 0.001 %
LibreOffice	4.1.1.2 \$Windows_x86	1	≈ 0.001 %
WPS Office	??_9.1.0.4468	1	≈ 0.001 %

## I Contents of *docProps/custom.xml* in sample document

Listing I.1: 123

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Properties
  xmlns="http://schemas.openxmlformats.org/officeDocument/2006/custom-properties"
  xmlns:vt="http://schemas.openxmlformats.org/officeDocument/2006/docPropsVTypes">
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="2"
    name="docId">
    <vt:lpwstr>292698</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="3"
    name="templateId">
    <vt:lpwstr>
    </vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="4"
    name="templateFilePath">
    <vt:lpwstr>\\FH-SRV-86\docprod\templates\ABC_Notat.dotm</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="5"
    name="filePathOneNote">
    <vt:lpwstr>\\FH-SRV-86\360users\onenote\ABC\it1</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="6"
    name="comment">
    <vt:lpwstr>Invitation to leader seminar</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="7"
    name="sourceId">
    <vt:lpwstr>
    </vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="8"
    name="module">
    <vt:lpwstr>Document</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="9"
    name="customParams">
    <vt:lpwstr>
    </vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="10"
    name="createdBy">
```

```

    <vt:lpwstr>Rob Johnson</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="11"
    name="modifiedBy">
    <vt:lpwstr>Rob Johnson</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="12"
    name="serverName">
    <vt:lpwstr>fh-srv-84</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="13"
    name="externalUser">
    <vt:lpwstr>
  </vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="14"
    name="BackOfficeType">
    <vt:lpwstr>Searchlink Solutions</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="15"
    name="Server">
    <vt:lpwstr>fh-srv-84</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="16"
    name="Protocol">
    <vt:lpwstr>off</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="17"
    name="Site">
    <vt:lpwstr>/locator.aspx</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="18"
    name="FileID">
    <vt:lpwstr>325024</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="19"
    name="VerID">
    <vt:lpwstr>0</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="20"
    name="FilePath">
    <vt:lpwstr>\\FH-SRV-84\360users\work\ABC\it1</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="21"
    name="FileName">
    <vt:lpwstr>Invitation to leader seminar</vt:lpwstr>
  </property>
  <property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="22"
    name="FullFileName">

```



```
<vt:lpwstr>\\FH-SRV-84\360users\work\ABC\it1\Invitation.DOCX</vt:lpwstr>
</property>
<property fmtid="{D5CDD505-2E9C-101B-9397-08002A2CF9AE}" pid="23"
  name="ContentTypeId">
  <vt:lpwstr>0x010100986B41D86696124FAC199B01E71E8933</vt:lpwstr>
</property>
</Properties>
```

---

## J Source code of OOFAT's most important functionality

### J.1 Document validator

Listing J.1: Document validator process.

```

1  /// <summary>
2  /// Attempt to validate document as a Office 2007, 2010 and 2013 document.
3  /// </summary>
4  /// <param name="documentPath">Path to the document.</param>
5  /// <returns>Returns true if document is a valid Office 2007, 2010 or 2013 ↵
    ↵ document.
6  /// Returns false if invalid.</returns>
7
8  public Boolean ValidateDocument(string documentPath)
9  {
10     Boolean valid = false;
11     Boolean[] errors = new Boolean[3];
12     errors[0] = errors[1] = errors[2] = false;
13
14     try
15     {
16         // Attempt to open document as a wordprocessingML document.
17         using (WordprocessingDocument wpDocument = ↵
            ↵ WordprocessingDocument.Open(documentPath, false))
18         {
19             try
20             {
21                 OpenXmlValidator validator2007 = new ↵
                    ↵ OpenXmlValidator(DocumentFormat.OpenXml.FileFormatVersions.Office2007);
22                 OpenXmlValidator validator2010 = new ↵
                    ↵ OpenXmlValidator(DocumentFormat.OpenXml.FileFormatVersions.Office2010);
23                 OpenXmlValidator validator2013 = new ↵
                    ↵ OpenXmlValidator(DocumentFormat.OpenXml.FileFormatVersions.Office2013);
24
25                 // Attempt to validate document as Office 2007 document.
26                 foreach (ValidationErrorInfo validationError in ↵
                    ↵ validator2007.Validate(wpDocument))
27                 {
28                     errors[0] = true; // Document is not a valid Office 2007 document.
29                 }
30
31                 // Attempt to validate document as Office 2010 document.
32                 foreach (ValidationErrorInfo validationError in ↵
                    ↵ validator2010.Validate(wpDocument))
33                 {
34                     errors[1] = true; // Document is not a valid Office 2010 document.
35                 }
36
37                 // Attempt to validate document as Office 2013 document.
38                 foreach (ValidationErrorInfo validationError in ↵
                    ↵ validator2013.Validate(wpDocument))
39                 {
40                     errors[2] = true; // Document is not a valid Office 2013 document.
41                 }

```

---

```
42     }
43
44     catch (Exception)
45     {
46         MessageBox.Show("Unable to load document, check path.");
47     }
48 }
49 }
50
51 catch (Exception)
52 {
53     errors[0] = errors[1] = errors[2] = true; // Input path is not an OOXML ↵
        ↵ document or is corrupted.
54 }
55
56 // Validation failed in Office 2007, 2010 and 2013.
57 if (errors[0] && errors[1] && errors[2])
58 {
59     valid = false; // Document is invalid.
60 }
61
62 // Validation succeeded in at least one of the validation attempts.
63 else
64 {
65     valid = true; // Document is valid.
66 }
67
68 return valid;
69 }
```

---

## J.2 Document metadata extractor

Listing J.2: Document metadata extraction process.

```

1  /// <summary>
2  /// Extract the metadata contained in docProps\app.xml and docProps\core.xml ↵
   ↵ of an OOXML document.
3  /// </summary>
4  /// <param name="inputFile">The document from which the metadata should be ↵
   ↵ extracted.</param>
5  /// <returns>Delimited string of metadata.</returns>
6  public string ExtractMetadata(InputOutputFile inputFile)
7  {
8      string appXMLpath = inputFile.GetOutputPath() + "\\docProps\\app.xml"; // ↵
   ↵ Path to docProps/app.xml in extracted document cache.
9      string coreXMLpath = inputFile.GetOutputPath() + "\\docProps\\core.xml"; // ↵
   ↵ Path to docProps/core.xml in extracted document cache.
10     //string customXMLpath = inputFile.GetOutputPath() + "\\docProps\\custom.xml";
11     string documentName = inputFile.GetFileName(); // The filename of the ↵
   ↵ document.
12     string documentHash = inputFile.GetHashsum(); // The document's ↵
   ↵ pre-calculated SHA1 hashsum.
13
14     Document document = new Document(); // Create new Document instance. ↵
   ↵ Consists of filename, SHA1 hashsum, created timestamp and revision ↵
   ↵ identifiers.
15     XDocument appXMLDocument = XDocument.Load(appXMLpath); // Read app.xml as ↵
   ↵ XML document.
16     XDocument coreXMLDocument = XDocument.Load(coreXMLpath); // Read core.xml ↵
   ↵ as XML document.
17
18     // Begin namespaces.
19     XNamespace cp = ↵
   ↵ "http://schemas.openxmlformats.org/package/2006/metadata/core-properties";
20     XNamespace dc = "http://purl.org/dc/elements/1.1/";
21     XNamespace dcterms = "http://purl.org/dc/terms/";
22     XNamespace dcmitype = "http://purl.org/dc/dcmitype/";
23     XNamespace xsi = "http://www.w3.org/2001/XMLSchema-instance";
24     XNamespace xmlns = ↵
   ↵ "http://schemas.openxmlformats.org/officeDocument/2006/extended-properties";
25     XNamespace vt = ↵
   ↵ "http://schemas.openxmlformats.org/officeDocument/2006/docPropsVTypes";
26     // End namespaces.
27
28     // Begin app.xml metadata type names.
29     string template;
30     string totalTime;
31     string pages;
32     string words;
33     string characters;
34     string application;
35     string docSecurity;
36     string lines;
37     string paragraphs;
38     string scaleCrop;
39     string company;
40     string linksUpToDate;
41     string charactersWithSpaces;
42     string sharedDoc;
43     string hyperlinksChanged;
44     string appVersion;
45     // End app.xml metadata type names.

```

```
46
47 // Begin LINQ queries for fetching data from app.xml.
48 try { template = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("Template")).Select(o => new { Template = ↵
    ↵ (string)o.Value ?? string.Empty ↵
    ↵ }).FirstOrDefault().Template.ToString(); }
49 catch (System.NullReferenceException) { template = string.Empty; }
50
51 try { totalTime = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("TotalTime")).Select(o => new { TotalTime = ↵
    ↵ (string)o.Value ?? string.Empty ↵
    ↵ }).FirstOrDefault().TotalTime.ToString(); }
52 catch (System.NullReferenceException) { totalTime = string.Empty; }
53
54 try { pages = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("Pages")).Select(o => new { Pages = ↵
    ↵ (string)o.Value ?? string.Empty }).FirstOrDefault().Pages.ToString(); }
55 catch (System.NullReferenceException) { pages = string.Empty; }
56
57 try { words = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("Words")).Select(o => new { Words = ↵
    ↵ (string)o.Value ?? string.Empty }).FirstOrDefault().Words.ToString(); }
58 catch (System.NullReferenceException) { words = string.Empty; }
59
60 try { characters = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("Characters")).Select(o => new { Characters = ↵
    ↵ (string)o.Value ?? string.Empty ↵
    ↵ }).FirstOrDefault().Characters.ToString(); }
61 catch (System.NullReferenceException) { characters = string.Empty; }
62
63 try { application = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("Application")).Select(o => new { Application ↵
    ↵ = (string)o.Value ?? string.Empty ↵
    ↵ }).FirstOrDefault().Application.ToString(); }
64 catch (System.NullReferenceException) { application = string.Empty; }
65
66 try { docSecurity = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("DocSecurity")).Select(o => new { DocSecurity ↵
    ↵ = (string)o.Value ?? string.Empty ↵
    ↵ }).FirstOrDefault().DocSecurity.ToString(); }
67 catch (System.NullReferenceException) { docSecurity = string.Empty; }
68
69 try { lines = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("Lines")).Select(o => new { Lines = ↵
    ↵ (string)o.Value ?? string.Empty }).FirstOrDefault().Lines.ToString(); }
70 catch (System.NullReferenceException) { lines = string.Empty; }
71
72 try { paragraphs = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("Paragraphs")).Select(o => new { Paragraphs = ↵
    ↵ (string)o.Value ?? string.Empty ↵
    ↵ }).FirstOrDefault().Paragraphs.ToString(); }
73 catch (System.NullReferenceException) { paragraphs = string.Empty; }
74
75 try { scaleCrop = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("ScaleCrop")).Select(o => new { ScaleCrop = ↵
    ↵ (string)o.Value ?? string.Empty ↵
    ↵ }).FirstOrDefault().ScaleCrop.ToString(); }
76 catch (System.NullReferenceException) { scaleCrop = string.Empty; }
77
78 try { company = appXMLDocument.Descendants().Where(o => ↵
    ↵ o.Name.LocalName.Equals("Company")).Select(o => new { Company = ↵
    ↵ (string)o.Value ?? string.Empty ↵
```

```

    ↪ }).FirstOrDefault().Company.ToString(); }
79 catch (System.NullReferenceException) { company = string.Empty; }
80
81 try { linksUpToDate = appXMLDocument.Descendants().Where(o => ↪
    ↪ o.Name.LocalName.Equals("LinksUpToDate")).Select(o => new { ↪
    ↪ LinksUpToDate = (string)o.Value ?? string.Empty ↪
    ↪ }).FirstOrDefault().LinksUpToDate.ToString(); }
82 catch (System.NullReferenceException) { linksUpToDate = string.Empty; }
83
84 try { charactersWithSpaces = appXMLDocument.Descendants().Where(o => ↪
    ↪ o.Name.LocalName.Equals("CharactersWithSpaces")).Select(o => new { ↪
    ↪ CharactersWithSpaces = (string)o.Value ?? string.Empty ↪
    ↪ }).FirstOrDefault().CharactersWithSpaces.ToString(); }
85 catch (System.NullReferenceException) { charactersWithSpaces = ↪
    ↪ string.Empty; }
86
87 try { sharedDoc = appXMLDocument.Descendants().Where(o => ↪
    ↪ o.Name.LocalName.Equals("SharedDoc")).Select(o => new { SharedDoc = ↪
    ↪ (string)o.Value ?? string.Empty ↪
    ↪ }).FirstOrDefault().SharedDoc.ToString(); }
88 catch (System.NullReferenceException) { sharedDoc = string.Empty; }
89
90 try { hyperlinksChanged = appXMLDocument.Descendants().Where(o => ↪
    ↪ o.Name.LocalName.Equals("HyperlinksChanged")).Select(o => new { ↪
    ↪ HyperlinksChanged = (string)o.Value ?? string.Empty ↪
    ↪ }).FirstOrDefault().HyperlinksChanged.ToString(); }
91 catch (System.NullReferenceException) { hyperlinksChanged = string.Empty; }
92
93 try { appVersion = appXMLDocument.Descendants().Where(o => ↪
    ↪ o.Name.LocalName.Equals("AppVersion")).Select(o => new { AppVersion = ↪
    ↪ (string)o.Value ?? string.Empty ↪
    ↪ }).FirstOrDefault().AppVersion.ToString(); }
94 catch (System.NullReferenceException) { appVersion = string.Empty; }
95 // End LINQ queries for fetching data from app.xml.
96
97 // Perform LINQ query for fetching data from core.xml.
98 var query = (from ele in coreXMLDocument.Descendants()
99     select new
100     {
101         title = (string)ele.Element(dc + "title") ?? string.Empty,
102         subject = (string)ele.Element(dc + "subject") ?? string.Empty,
103         creator = (string)ele.Element(dc + "creator") ?? string.Empty,
104         keywords = (string)ele.Element(cp + "keywords") ?? string.Empty,
105         description = (string)ele.Element(dc + "description") ?? string.Empty,
106         lastModifiedBy = (string)ele.Element(cp + "lastModifiedBy") ?? ↪
            ↪ string.Empty,
107         lastPrinted = (string)ele.Element(cp + "lastPrinted") ?? string.Empty,
108         revision = (string)ele.Element(cp + "revision") ?? string.Empty,
109         created = (string)ele.Element(dcterms + "created") ?? string.Empty,
110         modified = (string)ele.Element(dcterms + "modified") ?? string.Empty
111     }).FirstOrDefault();
112
113 // Begin LINQ separation queries for fetching data from core.xml.
114 string title = query.title.ToString();
115 string subject = query.subject.ToString();
116 string creator = query.creator.ToString();
117 string keywords = query.keywords.ToString();
118 string description = query.description.ToString();
119 string lastModifiedBy = query.lastModifiedBy.ToString();
120 string lastPrinted = query.lastPrinted.ToString();
121 string revision = query.revision.ToString();
122 string created = query.created.ToString();

```

```
123     string modified = query.modified.ToString();
124     // End LINQ separation queries for fetching data from core.xml.
125
126     string delimiter = ":@@@"; // Delimiter string for separating metadata ↵
        ↵ values. Should not be comma since metadata values could contain commas.
127
128     // String of metadata compiled, delimited.
129     string returnString = documentHash + delimiter + documentName + delimiter + ↵
        ↵ template + delimiter + totalTime + delimiter + pages + delimiter + ↵
        ↵ words + delimiter + characters + delimiter + application + delimiter ↵
        ↵ + docSecurity + delimiter + lines + delimiter + paragraphs + ↵
        ↵ delimiter + scaleCrop + delimiter + company + delimiter + ↵
        ↵ linksUpToDate + delimiter + charactersWithSpaces + delimiter + ↵
        ↵ sharedDoc + delimiter + hyperlinksChanged + delimiter + appVersion + ↵
        ↵ delimiter + title + delimiter + subject + delimiter + creator + ↵
        ↵ delimiter + keywords + delimiter + description + delimiter + ↵
        ↵ lastModifiedBy + delimiter + lastPrinted + delimiter + revision + ↵
        ↵ delimiter + created + delimiter + modified;
130
131     return returnString;
132 }
```

### J.3 Revision identifier extraction

Listing J.3: Document revision identifier extraction process.

```

1  /// <summary>
2  /// Extracts the revision identifiers from a document.
3  /// </summary>
4  /// <param name="filePath">Path to the document.</param>
5  /// <param name="currentDocumentId">Id of current document.</param>
6  /// <param name="makeUnique">Specifies if revision identifier lists should or ↵
   ↵ should not contain duplicates. True if duplicates are not wanted; false ↵
   ↵ if they are wanted.</param>
7  /// <returns>Document object containing revision identifier lists.</returns>
8  public Document ExtractRevisionIdentifiers(string filePath, RichTextBox ↵
   ↵ extractBox, int currentDocumentId, Boolean makeUnique)
9  {
10     Document document = new Document(); // Create new Document instance. ↵
        ↵ Consists of filename, SHA1 hashsum, created timestamp and revision ↵
        ↵ identifiers.
11     XDocument xmlDocument = XDocument.Load(filePath); // Load the main ↵
        ↵ document body as XML.
12     XNamespace w = ↵
        ↵ "http://schemas.openxmlformats.org/wordprocessingml/2006/main"; // ↵
        ↵ Namespace.
13
14     // Query to fetch & separate the paragraph revision identifiers from document.
15     var paragraphRevisions = (from feed in xmlDocument.Descendants(w + "p")
16         select new
17         {
18             rsidR = (string)feed.Attribute(w + "rsidR") ?? String.Empty,
19             rsidRPr = (string)feed.Attribute(w + "rsidRPr") ?? String.Empty,
20             rsidRDefault = (string)feed.Attribute(w + "rsidRDefault") ?? String.Empty,
21             rsidP = (string)feed.Attribute(w + "rsidP") ?? String.Empty
22         }).ToList();
23
24     // Query to fetch & separate the run revision identifiers from document.
25     var runRevisions = (from feed in xmlDocument.Descendants(w + "r")
26         select new
27         {
28             rsidR = (string)feed.Attribute(w + "rsidR") ?? String.Empty,
29             rsidRPr = (string)feed.Attribute(w + "rsidRPr") ?? String.Empty
30         }).ToList();
31
32     var runQuery = runRevisions.Select(rsidR => rsidR); // Perform LINQ selection.
33     var paragraphQuery = paragraphRevisions.Select(rsidR => rsidR); // Perform ↵
        ↵ LINQ selection.
34
35     // Begin lists of revision identifiers.
36     List<string> runRsidRPrList = new List<string>();
37     List<string> runRsidRList = new List<string>();
38     List<string> rsidRList = new List<string>();
39     List<string> rsidRPrList = new List<string>();
40     List<string> rsidRDefaultList = new List<string>();
41     List<string> rsidPList = new List<string>();
42     // End lists of revision identifiers.
43
44     // Go through each run in the document and add each revision identifier ↵
        ↵ type to its respective list.
45     foreach (var run in runQuery)
46     {
47         runRsidRPrList.Add(run.rsidRPr);
48         runRsidRList.Add(run.rsidR);

```



```
49     }
50
51     // Go through each paragraph in the document and add each revision ↵
52     ↵ identifier type to its respective list.
53     foreach (var paragraph in paragraphQuery)
54     {
55         rsidRList.Add(paragraph.rsidR);
56         rsidRPrList.Add(paragraph.rsidRPr);
57         rsidRDefaultList.Add(paragraph.rsidRDefault);
58         rsidPList.Add(paragraph.rsidP);
59     }
60
61     // Begin remove empty entries in the lists of rsids.
62     rsidRList.RemoveAll(string.IsNullOrEmpty);
63     rsidRPrList.RemoveAll(string.IsNullOrEmpty);
64     rsidRDefaultList.RemoveAll(string.IsNullOrEmpty);
65     rsidPList.RemoveAll(string.IsNullOrEmpty);
66     runRsidRPrList.RemoveAll(string.IsNullOrEmpty);
67     runRsidRList.RemoveAll(string.IsNullOrEmpty);
68     // End remove empty entries in the lists of rsids.
69
70     if (makeUnique) // We don't want duplicate rsids.
71     {
72         // Begin specifying the revision identifier type lists in the document ↵
73         ↵ instance, removing duplicates from the lists.
74         document.ParagraphRsidRList = MakeListUnique(rsidRList);
75         document.ParagraphRsidRPrList = MakeListUnique(rsidRPrList);
76         document.ParagraphRsidRDefaultList = MakeListUnique(rsidRDefaultList);
77         document.ParagraphRsidPList = MakeListUnique(rsidPList);
78         document.RunRsidRList = MakeListUnique(rsidRList);
79         document.RunRsidRPrList = MakeListUnique(rsidRPrList);
80         // End specifying the revision identifier type lists in the document ↵
81         ↵ instance, removing duplicates from the lists.
82     }
83
84     else // We want duplicate rsids.
85     {
86         // Begin specifying the revision identifier type lists in the document ↵
87         ↵ instance, removing duplicates from the lists.
88         document.ParagraphRsidRList = rsidRList;
89         document.ParagraphRsidRPrList = rsidRPrList;
90         document.ParagraphRsidRDefaultList = rsidRDefaultList;
91         document.ParagraphRsidPList = rsidPList;
92         document.RunRsidRList = rsidRList;
93         document.RunRsidRPrList = rsidRPrList;
94         // End specifying the revision identifier type lists in the document ↵
95         ↵ instance, removing duplicates from the lists.
96     }
97
98     document.DocumentId = currentDocumentId + 1; // Set id of current document.
99
100    return document;
101 }
```

## J.4 Revision identifier comparison

Listing J.4: Document revision identifier comparison process.

```

1  /// <summary>
2  /// Compare documents.
3  /// </summary>
4  /// <param name="documentList">List of documents to be compared.</param>
5  /// <param name="number">The type of revision identifier to be compared.</param>
6  /// <returns>List of documents with intersecting revision identifiers.</returns>
7  private List<DocumentIntersection> PerformComparison(List<Document> ↵
    ↵ documentList, int number)
8  {
9  List<DocumentIntersection> intersectingDocuments = new ↵
    ↵ List<DocumentIntersection>(); // List of documents with intersecting ↵
    ↵ revision identifiers.
10 string rsidType = string.Empty; // Type of rsid, based on input number.
11
12 // Go through each document.
13 foreach (Document doc in documentList)
14 {
15     if (number == 0) // We are looking for (run) rsidRPr
16         rsidType = doc.RunRsidRPrList.ElementAt(0);
17
18     if (number == 1) // We are looking for (paragraph) rsidR
19         rsidType = doc.ParagraphRsidRList.ElementAt(0);
20
21     if (number == 2) // We are looking for (run) rsidR
22         rsidType = doc.RunRsidRList.ElementAt(0);
23
24     if (number == 3) // We are looking for (paragraph) rsidRPr
25         rsidType = doc.ParagraphRsidRPrList.ElementAt(0);
26
27     if (number == 4) // We are looking for (paragraph) rsidP
28         rsidType = doc.ParagraphRsidPList.ElementAt(0);
29
30     if (number == 5) // We are looking for (paragraph) rsidRDefault
31         rsidType = doc.ParagraphRsidRDefaultList.ElementAt(0);
32
33     if (rsidType != string.Empty) // If the rsid list is empty, there is no ↵
        ↵ point comparing them.
34     {
35         // Inner loop going through the list again in order to compare documents.
36         foreach (Document currentDocument in documentList)
37         {
38             // Find any duplicates in the list of intersecting documents.
39             var docIntersectDuplicate = intersectingDocuments.Find(x => ↵
                ↵ x.FirstDocumentId == currentDocument.DocumentId && ↵
                ↵ x.SecondDocumentId == doc.DocumentId);
40
41             // Remove duplicate.
42             if (docIntersectDuplicate != null)
43             {
44                 intersectingDocuments.RemoveAt(intersectingDocuments. ↵
                    ↵ IndexOf(docIntersectDuplicate));
45             }
46
47             List<string> result = null;
48
49             if (number == 0) // We are looking for (run) rsidRPr
50                 result = ↵
                    ↵ doc.RunRsidRPrList.Intersect(currentDocument.RunRsidRPrList).ToList();

```

---

```

51
52     if (number == 1) // We are looking for (paragraph) rsidR
53         result = ↵
                    ↵ doc.ParagraphRsidRList.Intersect(currentDocument.ParagraphRsidRList) ↵
                    ↵ .ToList();
54
55     if (number == 2) // We are looking for (run) rsidR
56         result = doc.RunRsidRList.Intersect(currentDocument.RunRsidRList).ToList();
57
58     if (number == 3) // We are looking for (paragraph) rsidRPr
59         result = doc.ParagraphRsidRPrList.Intersect(currentDocument ↵
                    ↵ .ParagraphRsidRPrList).ToList();
60
61     if (number == 4) // We are looking for (paragraph) rsidP
62         result = doc.ParagraphRsidPList.Intersect(currentDocument ↵
                    ↵ .ParagraphRsidPList).ToList();
63
64     if (number == 5) // We are looking for (paragraph) rsidRDefault
65         result = doc.ParagraphRsidRDefaultList.Intersect(currentDocument ↵
                    ↵ .ParagraphRsidRDefaultList).ToList();
66
67     // Identical rsids are found.
68     if (result.Any())
69     {
70         if (doc.DocumentId != currentDocument.DocumentId) // We obviously don't ↵
71             ↵ want to compare a document with itself.
72         {
73             DocumentIntersection currentIntersectingDocuments = new ↵
74                 ↵ DocumentIntersection(doc.DocumentId, currentDocument.DocumentId);
75
76             if (number == 0) // We are looking for (run) rsidRPr
77                 currentIntersectingDocuments.RunRsidRPrList = result;
78
79             if (number == 1) // We are looking for (paragraph) rsidR
80                 currentIntersectingDocuments.ParagraphRsidRList = result;
81
82             if (number == 2) // We are looking for (run) rsidR
83                 currentIntersectingDocuments.RunRsidRList = result;
84
85             if (number == 3) // We are looking for (paragraph) rsidRPr
86                 currentIntersectingDocuments.ParagraphRsidRPrList = result;
87
88             if (number == 4) // We are looking for (paragraph) rsidP
89                 currentIntersectingDocuments.ParagraphRsidPList = result;
90
91             if (number == 5) // We are looking for (paragraph) rsidRDefault
92                 currentIntersectingDocuments.ParagraphRsidRDefaultList = result;
93
94             intersectingDocuments.Add(currentIntersectingDocuments); // Add to list ↵
95                 ↵ of intersecting documents.
96         }
97     }
98
99     return intersectingDocuments;
100 }

```

---

## J.5 Tree graph layout output

Listing J.5: Tree graph layout output.

```

1  /// <summary>
2  /// Creates a graph of documents, graphically showing the relationship between ↵
   ↵ documents.
3  /// </summary>
4  /// <param name="gViewer">Graph surface handler, i.e. where the graph is ↵
   ↵ drawn.</param>
5  /// <param name="intersectingDocuments">List of intersecting documents.</param>
6  public void CreateGraph(Microsoft.Glee.GraphViewerGdi.GViewer gViewer, ↵
   ↵ List<List<DocumentIntersection>> intersectingDocumentList, Boolean _showId)
7  {
8      Graph g = new Graph("graph"); // Graph new MSAGL graph surface.
9      showId = _showId;
10     List<Document> docList = LoadExtractedDocuments(); // Load list of documents.
11     graphIntersectingDocuments = intersectingDocumentList[0];
12     //gViewer.GraphHeight = 100;
13     g.GraphAttr.NodeAttr.Padding = 3;
14
15     // Go through each intersecting documents.
16     foreach (DocumentIntersection docIntersect in graphIntersectingDocuments)
17     {
18         string firstDocumentCreated = docList.Find(x => x.DocumentId == ↵
   ↵ docIntersect.FirstDocumentId).Created; // Creation timestamp of ↵
   ↵ document 1.
19         string secondDocumentCreated = docList.Find(x => x.DocumentId == ↵
   ↵ docIntersect.SecondDocumentId).Created; // Creation timestamp of ↵
   ↵ document 2.
20
21         DateTime t1 = DateTime.Parse(firstDocumentCreated); // Parse the timestamp.
22         DateTime t2 = DateTime.Parse(secondDocumentCreated); // Parse the timestamp.
23
24         int result = DateTime.Compare(t1, t2); // Compare timestamps to find the ↵
   ↵ oldest.
25
26         var firstDocumentName = docList.Find(x => x.DocumentId == ↵
   ↵ docIntersect.FirstDocumentId).DocumentName; // Name of document 1.
27         var secondDocumentName = docList.Find(x => x.DocumentId == ↵
   ↵ docIntersect.SecondDocumentId).DocumentName; // Name of document 2.
28
29         if (result < 0) // First document is older than second document.
30         {
31             if (showId)
32             {
33                 g.AddEdge(docIntersect.FirstDocumentId.ToString(), ↵
   ↵ docIntersect.SecondDocumentId.ToString()); // Add edge between ↵
   ↵ document nodes, showing id instead of filename.
34             }
35
36             else
37             {
38                 g.AddEdge(firstDocumentName.ToString(), ↵
   ↵ secondDocumentName.ToString()); // Add edge between document ↵
   ↵ nodes, showing filename instead of id.
39             }
40         }
41
42         else if (result == 0) // Documents are the same age.
43         {
44             if (showId)

```

---

```
45     {
46         g.AddEdge(docIntersect.FirstDocumentId.ToString(), ↵
           ↵ docIntersect.SecondDocumentId.ToString()); // Add edge between ↵
           ↵ document nodes, showing id instead of filename.
47     }
48
49     else
50     {
51         g.AddEdge(firstDocumentName.ToString(), ↵
           ↵ secondDocumentName.ToString()); // Add edge between document ↵
           ↵ nodes, showing filename instead of id.
52     }
53 }
54
55 else // Second document is older than the first document.
56 {
57     if (showId)
58     {
59         g.AddEdge(docIntersect.FirstDocumentId.ToString(), ↵
           ↵ docIntersect.SecondDocumentId.ToString()); // Add edge between ↵
           ↵ document nodes, showing id instead of filename.
60     }
61
62     else
63     {
64         g.AddEdge(firstDocumentName.ToString(), ↵
           ↵ secondDocumentName.ToString()); // Add edge between document ↵
           ↵ nodes, showing filename instead of id.
65     }
66 }
67 }
68
69 gViewer.Graph = g; // Add nodes and edges to graph surface.
70 }
```

---