UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA "RENATO M. CAPOCELLI"

CORSO DI DOTTORATO IN INFORMATICA
XI CICLO – NUOVA SERIE

ANNO ACCADEMICO 2011-2012

TESI DI DOTTORATO IN INFORMATICA

Steganographic Methods for Information Hiding in MS-Office Files

---

Tutor
prof. **Alfredo De Santis**

Candidato
Bonaventura D'Alessio

Coordinatore
prof. **Giuseppe Persiano**

To my father

my role model

Abstract

The simplest container of digital information is "the file" and among the vast array of files currently available, MS-Office files are probably the most widely used. These files, starting from MS-Office 2007, respect the standards 'Office Open XML" (OOXML). One of the benefits of the new format is that it lowers the risk of information leakage. Infact, before MS-Office 2007 was used binary files, called "Microsoft Document File Format" (MCDFF), were often used to host secret information.

In this work, starting from the classification of information hiding adapted from Bauer, methods for embedding data into the OOXML file format are analyzed. It also includes four novel methods. The first one, "*Data Hiding by Different Compression Algorithm in ZIP*", is based on the characteristic that the new MS-Office documents are container of compressed files. The second one, "*Data Hiding by Office Macro*", uses modified macro to store secret messages. The third one, "*Data Hiding by Zero Dimension Image*", is based on the characteristic that it is possible to create invisible images, setting the width and height both equal to zero. The last one, "*Data Hiding by Revision Identifier Value*" hides data by storing it in some attributes of XML elements (*rsid*). The methods presented can be combined in order to extend the amount of data to be hidden in a single cover file.

Analyzing a scenario composed of about $50,000$ MS-Office files, we show how the proposed methods can be helpful in real applications. We have calculated the *capacity* of all stegosystem described, verifying the amount of information that can be hidden. After that, an evaluation of the limits of the proposed methods is carried out by comparing

them with the tool introduced by Microsoft to sanitize MS-Office files. This tool, called *Document Inspector*, was projected to help the users to find and remove hidden data and personal information in MS-Office documents.

Acknowledgements

I would like to thank my family for having made all this possible.Without her encouragement and her love I would not have been able to finish this work. My deepest gratitude goes to my tutor, Alfredo De Santis. He has helped me to grow up my knowledge in digital forensic science, steganography techniques and secure communications. A special thank goes to Pino Persiano for his valuable support. I am very gratful to Pippo Cattaneo and Giuseppe Palo, they have encouraged me to begin this new challenge. I owe a debt of gratitude to Nello Castiglione for supporting and helping me. Finally, I would like to thank my father. His teaching have made it possible for me to abotain great goals in my life.

Contents

List of Figures

Chapter 1

Introduction

To hide data or to communicate in secure way is a requirement that people has felt since the more ancient times. In the past the secret documents were stored in strongbox or in secure places that are only known to the secret's keepers. Instead, there were different techniques used in order to communicate secret information (messages written on the wood, then covered it with wax upon which an innocent covering message was written, message tattooed on the shaved head of a slave, etc.). Also today there are such necessities. The advent of the digital age has considerably changed the scene. The information, asset to protect from the access of unauthorized people, is not written (for example on a paper sheet) but it is stored in electronic devices (hard-disk, USB mass storage, etc.). Moreover, this necessity is more diffused that in the past. Everyone have an user ID, a password and a PIN that must be remembered to access to the several services become of common use (credit card, Internet sites, cellular phone, etc.). Encoding the data is certainly a solution to such problem. In this way, we do not hide our secret but only we make unreadable the information to the unauthorized subjects. The use of steganographic techniques, instead, affords us to conceal, in "apparently innocuous" containers, both the secret and its existence.

1.1 Thesis

The Microsoft Office suite is without a doubt the most widely used word-processing tool when preparing and writing documents, spreadsheets and presentations. Therefore, the

possibility to hide information inside them is a challenge that probably interests many different parties. Moreover, starting from the 2007 version (MS-Office 2007), Microsoft has completely changed the format of its files increasing, among other things, the level of security and thus making it more difficult to hide information inside them. Practically, with the new version it is impossible to apply almost the totality of the methodologies of concealment information used until MS-Office XP. But, the new structure adopted from Microsoft and called OOXML [20], that it has replaced the old binary format, has offered new opportunities. In this direction we have addressed our research. The first step has been a deep study of standard OOXML. Its main characteristic is of being a container ZIP that stores different type of files (mainly XML). This is to identify the weakness of this new format. Then we have:

- verified if the steganographic techniques, already known for the old version of MS-Office, are still applicable to the new structure;
- verified if it is possible to hide data inside the single files, stored in the ZIP container, using technical applicable to the specific file type;
- analyzed the organization of the ZIP container and of the files stored in it, to search innovative methodologies.

These studies have made it possible to obtain complete screening on the steganographic techniques applicable to MS-Office 2007/2010 documents as well as to characterize of ulterior 4 new:

- Data Hiding by Different Compression Algorithm of ZIP;
- Data Hiding by Office Macro;
- Data Hiding by Zero Dimension Image;
- Data Hiding by the Revision Identifier Value.

1.2 The Organization of this Thesis

In the second chapter we will describe distinctive feature of the MS-Office 2007/2010. We will introduce the OOXML standard and we will discuss the obvious implications when dealing with security and privacy. In the third chapter we will introduce the steganography and illustrate its classification.
In the chapter four we will discuss the methodologies that take advantage of the characteristic of producing compressed files introduced by OOXML standard. In the chapter five we will highlight how it is possible to hide data in the XML code contained in MS-Office files. In the chapter six we will analyze the methodologies that use the characteristics and formats of the MS-Office documents. In the chapter seven we will illustrate how the "OLE object" can be used to hide information. In the chapter eight we will describe the methodologies of information hiding based on the characteristic that MS-Office documents are compliant to OOXML standard.
In the chapters nine and ten, we will compare all the methodologies and propose the results of experimental tests. In detail, we will classify the techniques according to the Bauer scheme, we will verify if the *Document Inspector* detect and remove the secret message, we will analyze the resulting behaviour of "save" actions and we will estimate the overhead introduced.
Finally, we will provide some conclusions about our work together with several interesting future research hints.

Chapter 2

OOXML in MS-Office 2007/2010

2.1 Introduction

The Office Open XML (OOXML) format, which is based on the ECMA-376 standard [5], is a document description and processing language. It specifies a family of XML schemes which define the XML vocabularies for word-processing, spreadsheet, and presentation documents, as well as the packaging of documents that conform to these formats. The principal characteristic is that it is based on XML format (XML-based).

The Extensible Markup Language (XML) is used for the representation of structured data and documents. It is a markup language and, thus, composed of instructions defined as tags or markers. Therefore, in XML a document is described, in form and content, by a sequence of elements. Each element is defined by a *tag* or a pair *start-tag*/*end-tag*, which can have one or more attributes. These attributes define the properties of the elements in terms of values.

The OOXML format is based on the principle that even a third party, without necessarily owning product rights, can extract and relocate the contents of the file by only using standard transformation methods. This is possible because XML text is clearly written and therefore visible and modifiable with a common text editor. Moreover, OLE attachments [19] are present in the source file format and, therefore, can be visualized with any compatible viewer.

Starting with the 2007 version, Microsoft has adopted the OOXML format [20] and has begun the transition from the old logic, which used the generation of a binary file format, to a new one that uses XML format. Distinguishing documents produced in this new format is easy due to the file extensions being characterized by an "x" at the end, with

the file Word, Excel and PowerPoint respectively being *.docx*, *.xlsx*, *.pptx*. An additional feature is about the macros which are not activated unless specified by the user. In this case, the extension of the files changes by adding "*m*" rather than "*x*" and thus become *.docm*, *.xlsm*, *.pptm*.

The new structure of an OOXML file uses a container, a ZIP file, inside of which there are a series of files, mostly XML, which are opportunely organized into folders, which describe both the content and the properties and relationships of them. It is highly likely that the ZIP standard was chosen because it is the most commercially well-known, in addition to having characteristics of flexibility and modularity that allow for any eventual expansions in future functionalities [10]. There are three types of files stored in the "container" common to all the applications of MS-Office (Word, Excel and PowerPoint):

- XML files which describe application data, metadata, as well as customer data, stored inside the container file;
- non-XML files, may also be included within the container, including such parts as binary files representing images or OLE objects embedded in the document;
- the relationship parts describe the relation among the parts of MS-Office document, so analyzing these issues it is possible to obtain the file structure scheme.

For example, in a simple Word document the structure [6] of folders and files are shown in Fig. 2.1.

Another key concept related to the OOXML format is the modularity, both inside the files and between the same files, which allows for the easy addition of new elements as well as the removal of old ones. For example, the addition of a new JPEG image inside a Word file could be simply performed by:

- copying the file with the *.jpg* extension in the folder named *media* within the ZIP container;
- adding a group of elements in the *document.xml* file (it contains the XML markups that define the contents of the document) in order to describe the insertion methods within the page;

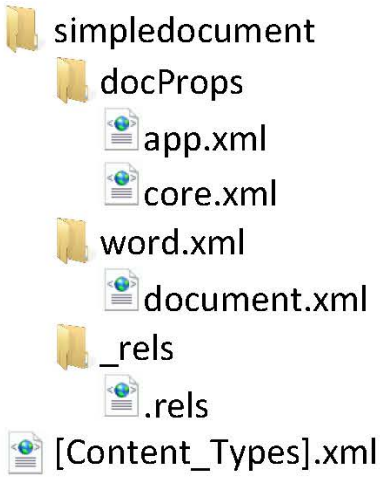

Figure 2.1: Structure of a simple Word document.

- adding, in several files of the relationship, some XML lines which declare the use of an image.

The intention of Microsoft with the introduction of the OOXML format was to give new opportunities to the community [4]. In fact with the new standard:

- it is possible to show just the text of the document. For example, if the file is a Word document only the file *document.xml* will be analyzed without necessarily opening all the files (containing the remaining information about the document);
- the files are compressed, and consequently are shorter and easy to manage;
- it is easy to scan for viruses or malicious contents thanks to its textual form instead of the old binary format;
- the new format does not allow macro and hence guarantee a good level of security;
- if some files in the ZIP container are damaged, the integrity of the entire document could be preserved and, in some cases, the main document could be reconstructed starting from the remaining "untouched" files.

MS-Office 2010, also known as Office 14, maintains formats and interfaces that are similar to the 2007 version. The substantial difference between the two suites is that MS-Office 2010 is much more web-oriented than the previous one ([17], [12]). The new suite, for

example, sends to the user an alert message when transmitting sensitive information via e-mail. It is also able to translate documents and deal with different languages, as well as transform presentations into clips. It makes possible to present a PowerPoint "slideshow" to users connected to the Internet. In [11] Microsoft analyzes the MS-Office 2010 format, describing some of their characteristics, all the features introduced in the new version and highlights the updated parts with respect to the old version.

2.2 Security and Privacy in MS-Office files

The management flexibility offered by the new OOXML format has obvious implications when dealing with security [16]. On one hand, the clear-text offers the seeming impossibility to hide information. While, on the other hand, it offers the possibility to malicious parties to read its content and eventually manipulate it. It is also well-known that MS-Office files contain data that can reveal unwanted personal information, such as people who have collaborated in the writing of the document, network parameters, as well as devices on which it has been edited. In current literature, there are several papers which describe how to extract and reconstruct several different types of information from such documents. While studying the problem of information leakage in MS-Office document, Castiglione et al. [9] introduced also a steganography system which can be applied to all versions before MS-Office 2007. Furthermore, authors analyzed the information leakage issue [7] raised by MS-Office 2007 documents and the unused space inside the file structure adopted by Microsoft until MS-Office XP [18].

The old methodologies exploiting the characteristics of the binary files of MS-Office are no longer applicable to the new XML structure. However, the steganography techniques that take advantage of the functions offered by the Microsoft suite ([1], [16], [14] and [15]), are still valid, and therefore independent from the version of MS-Office adopted. The new format offers new perspectives, as proposed by Garfinkel et al. [2] as well as Park et al. [3]. Both authors describe methodologies that use characteristics that do not conform to the OOXML standard and therefore can be characterized by searching for abnormal content type that is not described in the OOXML specifications inside of the file.

In order to guarantee a higher level of security and privacy, Microsoft (starting from MS-Office 2007 for Windows) has introduced the feature called *Document Inspector* accessible

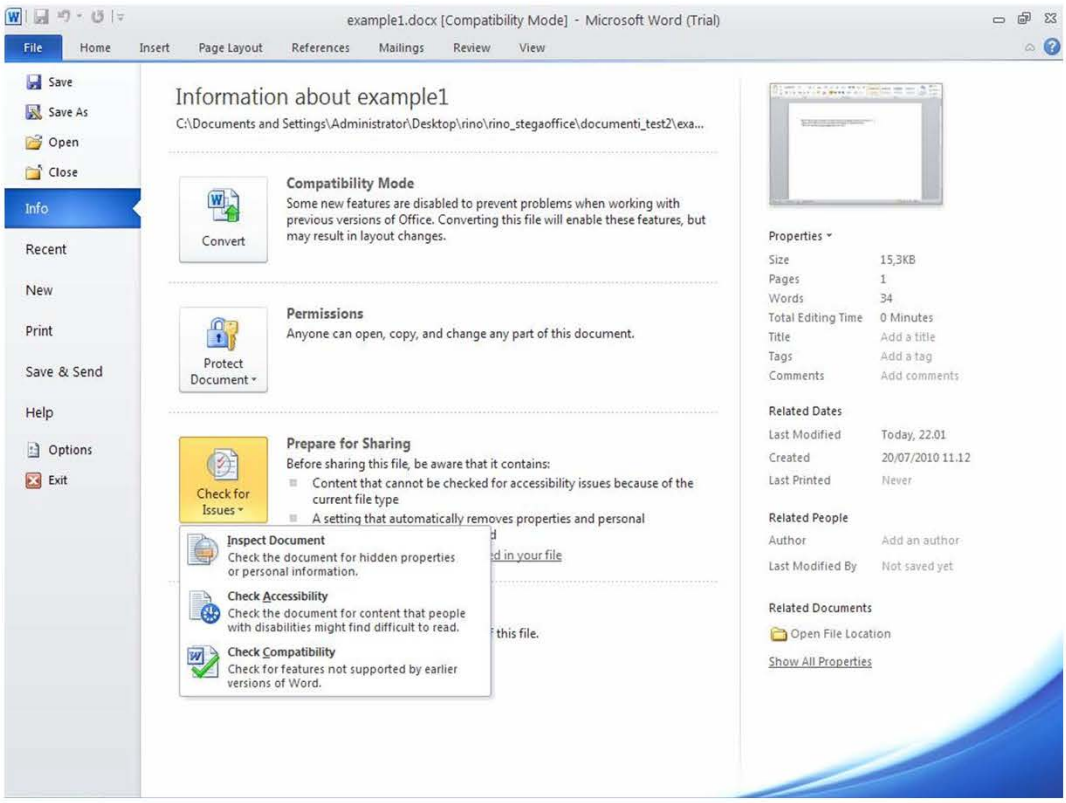

Figure 2.2: How to run the *Document Inspector* feature within Microsoft Word.

from the menu "File" $\rightarrow$ "Info" $\rightarrow$ "Check for Issues" (see Fig. 2.2). The *Document Inspector* makes it possible to find and remove, quickly, personal, sensitive and hidden information (see Fig. 2.3). More details on the *Document Inspector* can be found in the document published by Microsoft [8].

A global solution to privacy and security problem does not exist. Then it is possible to bypass the *Document Inspector* controls and inoculate a secret message inside MS-Office 2007/2010 files. In the following chapters we will analyzed the steganography techniques applicable to Microsoft Office files conform to the OOXML standard. These methodologies will be classified starting from the proprieties that the MS-Office documents are files: based on the ZIP standard, contain XML files, have common characteristics and formats to those of generic MS-Office files (character format, cell properties, collaborative document, etc.), may contain OLE objects (images, audio files, etc.) as well as conform to the ECMA-376 standard, opportunely customized.

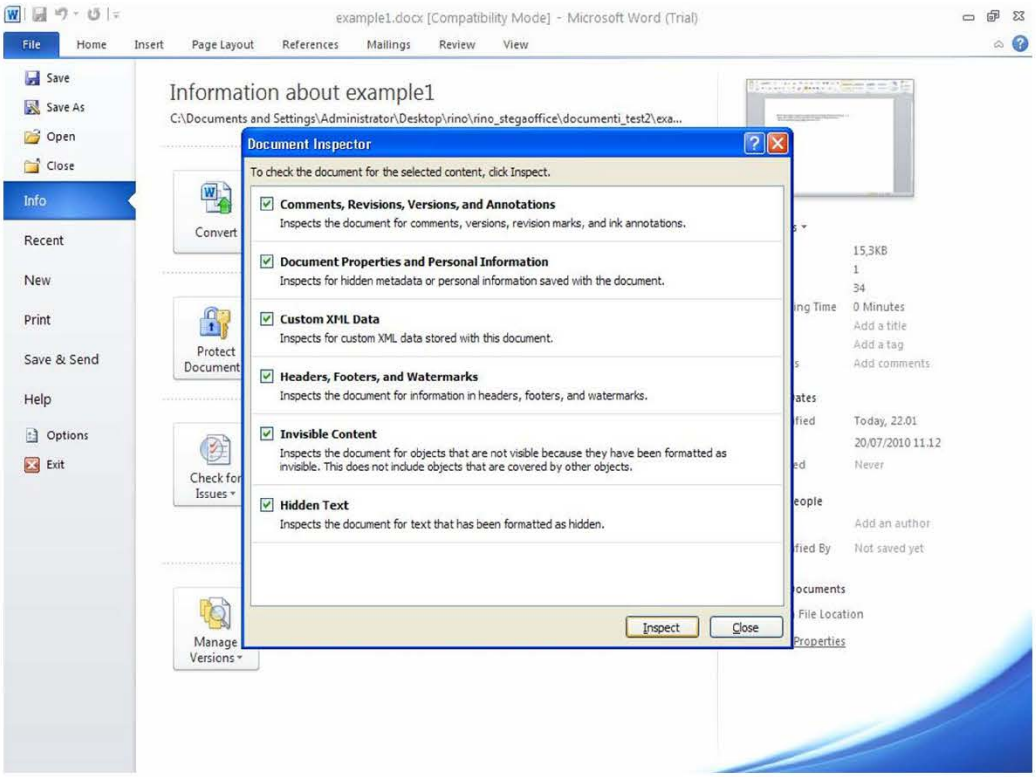

Figure 2.3: What can be found and removed with the *Document Inspector*.

References

[1] T.Y. Liu and W.H. Tsai, "A New Steganographic Method for Data Hiding in Microsoft Word Documents by a Change Tracking Technique". *IEEE Transactions on Information Forensics and Security*, vol.2(1), pages 24-30, 2007.

[2] S.L. Garfinkel and J.J. Migletz, "New XML-Based Files Implications for Forensics". *IEEE Security & Privacy*, vol.7(2), pages 38-44, 2009.

[3] B. Park, J. Park and S. Lee, "Data concealment and detection in Microsoft Office 2007 files". *Digital Investigation*, vol.5(3-4), pages 104-144, 2009.

[4] F. Rice, "Microsoft MSDN. Introducing the Office (2007) Open XML File Formats". http://msdn.microsoft.com/it-it/library/aa338205.aspx, May 2006.

[5] ECMA International, "Final draft standard ECMA-376 Office Open XML File Formats - Part 1". *ECMA International Publication*, December 2008.

[6] E. Ehrli, "Building Server-Side Document Generation Solutions Using the Open XML Object Model". http://msdn.microsoft.com/en-us/library/bb735940%28office.12%29.aspx, August 2007.

[7] S. Kiyomoto and K. M. Martin, "Model for a common notion of privacy leakage on public database". *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol.2(1), pages 50-62, 2011.

[8] Microsoft Corporation, "Remove hidden data and personal information from office documents". http://office.microsoft.com/en-us/excel-help/remove-hidden-data-and-personal-information-from-office-documents-HA010037593.aspx, visited March 2011.

[9] A. Castiglione, A. De Santis and C. Soriente, "Taking advantages of a disadvantage: Digital forensics and steganography using document metadata". *Journal of Systems and Software*, vol.80(5), pages 750-764, 2007

[10] Wikipedia, "ZIP (file format)". http://en.Wikipedia.org/wiki/ZIP_(file_ format), visited March 2011.

[11] Microsoft, "Compare Office Professional Plus 2010 and the 2007 suite". http://office.microsoft.com/en-us/professional-plus/professional-plus-version-comparison-FX101871482.aspx, visited March 2011.

[12] Microsoft TechNet, "What's new for IT professionals in Office 2010". http://technet.microsoft.com/en-us/library/dd188670.aspx, October 2010.

[13] M.Q. Jing, W-C. Yang and L.H. Chen, "A new steganography method via various animation timing effects in PowerPoint files". *International Conference on Machine Learning and Cybernetics 2009*, vol.5, pages 2840-2845, July 2009.

[14] I. C. Lin and P. K. Hsu, "A Data Hiding Scheme on Word Documents Using Multiple-Base Notation System". *Sixth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP2010)*, pages 31-33, October 2010.

[15] H. Zhang, L. Huang, Y. Ye and P. Meng, "A new steganography method via combination in PowerPoint files". *International Conference on Computer Application and System Modeling (ICCASM2010)*, vol.2, pages 62-66, October 2010.

[16] P. Lagadec, "OpenDocument and Open XML security (OpenOffice.org and MS Office 2007)". *Journal in Computer Virology*, vol.4(2), pages 115-125, May 2008.

[17] Microsoft TechNet, "Changes in Office 2010". http://technet.microsoft.com/en-us/library/cc178954.aspx, October 2010.

[18] G. Cantrell and D.D. Dampier, "Experiments in hiding data inside the file structure of common office documents: a stegonography application". *Proceedings of the 2004 international symposium on Information and communication technologies*, ISICT '04, pages 146-151, 2004.

[19] MSDN Library, "[MS-OLEDS]: Object Linking and Embedding (OLE) Data Structures". http://msdn.microsoft.com/en-us/library/dd942265%28v=prot.10%29.aspx, visited February 2012.

[20] F. Rice, "Microsoft MSDN. Introducing the Office (2007) Open XML File Formats". http://msdn.microsoft.com/it-it/library/aa338205.aspx, May 2006.

Chapter 3

Steganography

The Information Hiding is a field of Information Security. This term refers to several techniques used to hide information in different types of "digital containers" (transmission channels, documents, audios, videos, programs, images, etc.). The reasons for hiding information can be various. As an example to affix a digital seal that difficulty can be removed or altered (digital watermarking) or to create "covert channel" that is difficult to reconstruct the detailed of the transceiver [13].
An important discipline of Information Hiding is Steganography. While the Cryptography studies how to protect the content of a message, the Steganography studies the methodologies for hiding the presence of the message. The origin of the word steganography comes from the Greek $\sigma\tau\epsilon\gamma\alpha\nu\varnothing\varsigma$ and $\gamma\rho\alpha\varphi\iota\alpha$ that literally means "hidden writing" [1]. Currently it is considered as the possibility to hide information in other data.

3.1 Steganography

Steganography has remote origins and it appears in various periods of history. The Greeks, for example, used wooden tablets covered with wax or a tattoo on the head of the slaves [9]. In the old China the messages were written on strips of fine silk, wrapped to form small balls, covered with wax and then heated. In Italy Girolamo Cardano, a mathematician, in 1550 invented a quick and easy method to send hidden messages. He used sheet on which were carved out of holes, then placed this grid on a blank sheet and, inside the holes, the characters of the hidden message were positioned. Then the grid was removed and the rest of the sheet was completed to obtain correct sentences. Also during the World War II [4]

steganography was used and, to conceal information, inks sympathetic or microdots were used.

An important contribution, in this area, was offered by John Trithemius (1642-1516), a German abbot. He proposed an innovative method in his most famous work *Steganographia* [6]. This method, apparently magic, was useful to communicate hidden information using a seemingly innocuous text.

The advent of the digital age has greatly stimulated studies on information hiding. In fact, to be protected information are increased (e.g. access codes for ATMs, mobile phones, Internet services) and digital containers (virtual areas where stored secrets) are appeared. The main digital containers on which to apply steganography techniques (see Section 7.2) are: text files, graphic files and audio files [8]. These supports have three characteristics, that make them particularly suitable for steganographic applications [5]. In fact: they have the noise (mainly produced by communication channels and by analog-digital conversion), they can be changed without modify how they work and a little change, that modifies its content, appears imperceptible to people. A steganographic system can be a lots of applications but its purposes are substantially two: hide the presence of hidden data and protect the information from its removal. It is characterized by three main parameters [10]:

- **capacity**, how much the information can be hidden in the container;
- **security**, how much the hidden data are invisible;
- **robustness**, how much the system resists, without losing the hidden message, to manipulations of the files or to attack.

Although steganography has very old origins, only at the first International Workshop on Information Hiding, held in Cambridge in 1996, were defined key terms [11]. They are:

- ***cover*** (*cover medium*), container where the secret message will be injected;
- ***embedded***, secret information that will be hidden inside the *cover medium*;
- ***stego-key***, key that reveal the *embedded*;
- ***stego-medium*** (*stego-data*), the result after applying a steganographic system to *cover* to conceal an *embedded* using a *stego-key*:

$$cover + embedded + stego\text{-}key = stego\text{-}medium$$

Depending on the *cover*, three different steganografic approaches can be used: *Substitution Techniques* (or *Injection Techniques*), *Selective Techniques* e *Costruction Techniques* (or *Generation Techniques*).

Substitution Techniques are the most used. They employ an already existing container. *Cover* is suitably modified, partially or fully, in order to store the secret information without appearing "different" to an observer.

Selective steganography, however, has only a theoretical application. It is based on the principle that it should search a container, with specific characteristics, suitable to contain the secret message. As an example, suppose we want to communicate the information hidden "help". We choose four text files with the characteristic of having the fifth character equal to: "h", in the first file, "e", in the second file, "l' ', in the third file and " p" in last one. The recipient, who knows the applied algorithm, receiving the documents sequentially will be able to easily decrypt the content. This method, although difficult to identify because of the unmodified *cover*, allows for a concealment capacity very small, which makes it difficult to apply in the practical field.

Finally, *Construction Techniques* are based on the principle that, starting from the message to hide, we create a customized *cover*, for the specific secret. In this way the user can build a container with characteristics that best fit the *embedded*.

Subsequently, we formalized the steganography process. Suppose we have two users (A and B) that need to communicate a secret (E) in invisible way to other persons. To perform it they use a container (C) and a steganographic function (f_k), which is activated by a *stego-key* (K). Therefore, using the $f_k(C, E)$ it generates the *stego-medium* and using the inverse $f_k^{-1}(C')$ it revals the secret. The Fig. 3.1 shows the block diagram of a stego-system [12].

Usually, Steganography and Cryptography get confused, so it is important to clarify both concepts [7].

The target of Cryptography is to conceal the meaning of secret message [14]. So the adversary can intercept the secret and try to modify it or to decrypt it. Therefore, a successfully attack to cryptography system is when the enemy decrypt the secret message. The target of Steganography, however, is to make invisible the hidden message and conceal its meaning. Then, an attack the steganography mainly focuses on two targets:

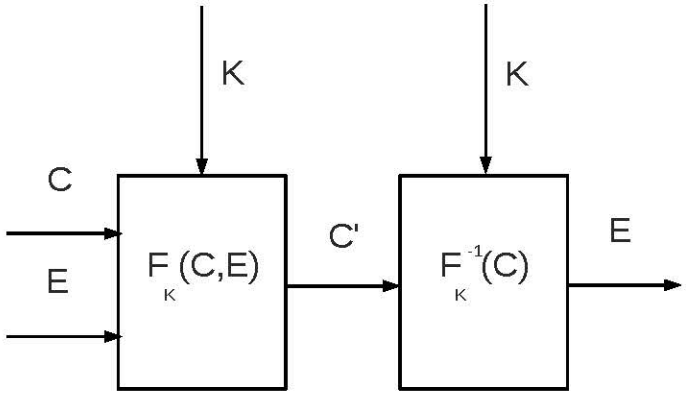

Figure 3.1: Stego-system.

- to find the presence of a hidden message;
- to identify the content of the hidden message itself.

The aim of steganography is to obtain a hidden communication between two subjects. Considering the two targets presented above, just discovers the presence of a secret communication can be considered as a successful attack.
It is important to add that, Cryptography and Steganography, can be combined together. Then, in order to guarantee data confidentiality, before proceeding to information embedding, applying an steganographic technique, all data may be encrypted, using a cryptographic algorithm.

3.2 Bauer classification

Bauer [2] classifies the Steganography in two categories: *Linguistic Steganography* and *Technical Steganography* (see Figure 3.2).
The *Linguistic Steganography* conceals the message in the carrier through not obvious methods while the *Technical Steganography* uses scientific methodologies. The first technique is also divided in: *Semagrams* and *Open Codes*.
In the *Semagrams*, the information is usually hidden through the use of symbols or signs. The *Visual Semagram* uses innocent-looking or everyday physical objects to communicate a message (for example: microdot [3] or positions of elements on a physical desk or virtual

desk). In the *Text Semagram* the message is hidden in the different way of visualization of data (for example a "thin" modification of the font or the dimension of the character, the addition of extra spaces, etc.).

The *Open Codes* technique, differently from the *Semagrams*, hides the information through "legitimate carrier message" in the way that the presence of information does not turn out obvious for an unsuspicious observer. Usually the "carrier message" is called *overt communication* while the hidden message is the *covert communication*. This technique can be further divided in: *Jargon Code* and *Covered Ciphers*.

Jargon Code, as suggests the name, is the use of a language understood from a limited group of people. The *Jargon Code* includes "warchalking" (the use of symbols to show the presence of wireless networks), underground terminologies or innocent conversation that hide special meanings that have sense only for the interlocutors. A subset of the *Jargon Codes* are the *Cue Codes*, where phrases properly composed carry a meaning different from that obvious one.

Covered, or *Concealment*, *Ciphers* explicitly hides a message on the carrier. The hidden information can be recovered by anyone who knows the secret of how it was concealed. The *Grille Cipher* needs the use of template. The template is applied on a carrier message, in a way that only the characters which compose the secret message are visible while the other are obfuscated. In the *Null Cipher* the message is hidden using a set of rules agreed among the users. For example, read every seven words or see the second character of each word.

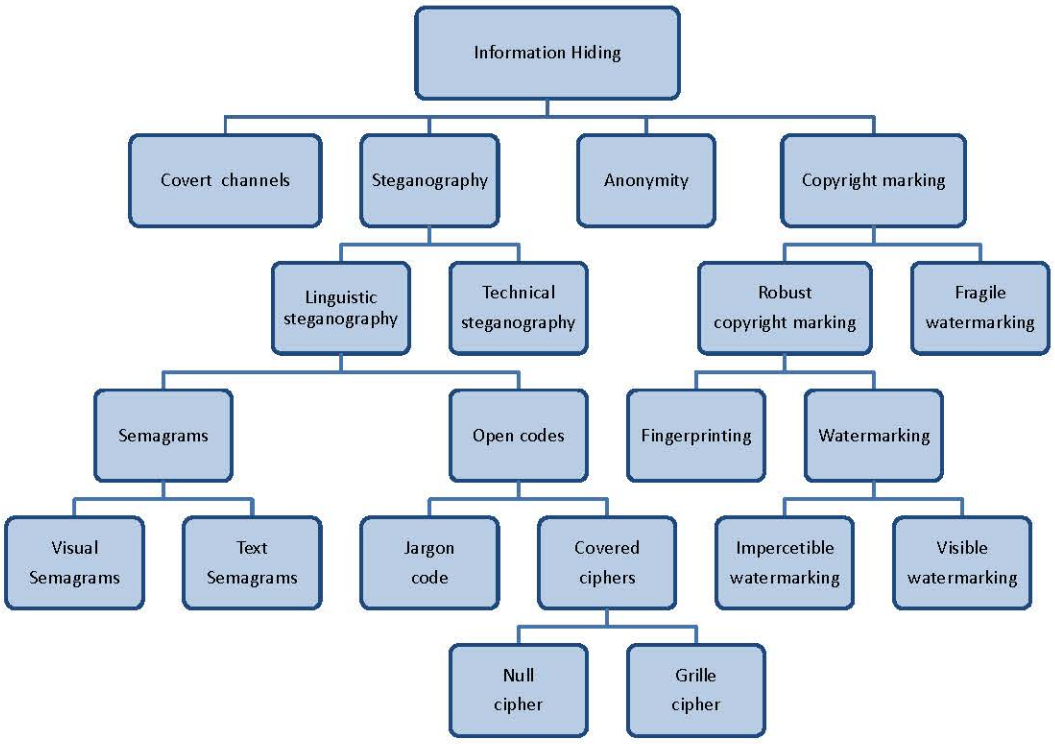

Figure 3.2: Bauer classification of Information Hiding.

References

[1] G. Schotti, "Schola Steganographica: In Classes Octo Distributa". *Whipple Collection (Cambridge Univ)*, 1680.

[2] F.L. Bauer, "Decrypted Secrets: Methods and Maxims of Cryptolog, 4rd ed", pages 9-24, 2007.

[3] J.E. Hoover, "The enemy's masterpiece of espionage". *Reader's Digest*, vol.48, pages 49-53, May 1946.

[4] D. Kahn, "The Code-Breakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet". *New York: Scribner*, 1996.

[5] N. Provos, P. Honeyman, P., "Hide and seek: an introduction to steganography". *Security & Privacy, IEEE*, vol.1(3), pages 32-44, May-June 2003.

[6] J. Trithemius, "Steganographia". *Zubrodt*, 1676.

[7] R. Yadav, "Study of Information Hiding Techniques and their Counterattacks: A Review Article". *International Journal of Computer Science & Communication Networks*, vol.1(2), pages 142-164, Oct-Nov 2011.

[8] H.O. Alanazi, A.A. Zaidan, H.A. Jalab and Z.K. Al-Ani, "New Classification Methods for Hiding Information into Two Parts: Multimedia Files and Non Multimedia Files". *Journal Of Computing*, vol.2(3), March 2010.

[9] D. Kahn, "Information Hiding", *Information Hiding - Lecture Notes in Computer Science*, vol.1174, pages 1-5, 1996.

[10] I.J. Cox, M.L. Miller, J.A. Bloom, J. Fridrich and T. Kalker, "Digital Watermarking and Steganography. Second Edition". *Morgan Kaufmann Publishers is an imprint of Elsevier*, page 15, 2008.

[11] R.J. Anderson, "Information Hiding: 1st Int. Workshop (Lecture Notes in Computer Science)". *Springer-Verlag*, vol.1174, 1996.

[12] B. Pfitzmann, "Information Hiding Terminology". *Information Hiding Workshop. Springer-Verlag*, June 1996.

[13] F.A.P. Petitcolas, R.J. Anderson and M.G. KUHN, "Information Hiding A Survey". *Proceeding of the IEEE*, vol.87(7), July 1999.

[14] A. Kerckhoffs, "La cryptographie militaire". *J. Sciences Militaires*, vol.9, pages 538, January 1883.

Chapter 4

Information Hiding by ZIP files

4.1 Introduction

As a consequence of the OOXML features of generating compressed files, it is possible to hide data "inside" the compressed file structure. Considering that the compressed file format is the ZIP format, it is possible to use some hiding methods common, and applicable, to the general ZIP format.

4.2 Comment in ZIP File

The applications that manage the majority of compression algorithms offer the possibility to add a comment to a compressed archive. The "text" inserted as a comment could be the secret message. Obviously, if applied, such methodology is easily discovered because the comment is visible just by opening the archive. Moreover, even though the secret message is embedded by. In this way the comment added will appear [3].

4.3 Information Hiding by Different Compression Algorithm of ZIP

Taking advantage of the characteristic that OOXML standard produces on compressed files, it is possible to hide information inside a ZIP structure without taking

Algorithm	Acronym	Option	Char
best method for each file (based on the file type)	-	*ez*	-
maximum (*PPMd*)	*PPDM*	*ep*	-
maximum (*LZMA*)	*LZMA*	*el*	-
maximum (*bzip2*)	*BZIPPED*	*eb*	-
maximum (*enhanced deflate*)	*EnhDefl*	*ee*	-
fast	*DeflateF*	*ef*	*0*
normal	*DeflateN*	*en*	*1*
maximum (*portable*)	*DeflateX*	*ex*	*2*
super fast	*DeflateS*	*es*	*3*
no compression	*Stored*	*e0*	*4*

Table 4.1: Compression options in the ZIP format and association characters-algorithms.

into account that the same file will be interpreted by MS-Office as a document produced by its own application. The ZIP format is a data compression and archive format. Data compression is carried out using the DeflateS format [1], which is set as default, with it being possible to set a different compression algorithm. For example, by using WinZip (ver. 14.5 with the command-line add-on ver. 3.2) it is possible to choose one of the compression algorithm indicated in Table 4.1.

Therefore, by inserting one of the options indicated in Table 4.1 in the command:

```
wzzip [options] zipfile [@listafile] [files...]
```

the desired algorithm compression will be applied. It is worth noting that, in a ZIP container, all the files contained can be compressed with a different algorithm. To reproduce a MS-Office file, it is possible to manually choose/modify the compression algorithm. In fact, each file contained in a MS-Office document can be compressed with a different compression algorithm. The default compression algorithm used by MS-Office is the DeflateS format but it is possible to choose some other algorithms. Not all the algorithms listed in Table 4.1 are correctly interpreted by MS-Office. In fact, after some tests, it has been possible to ascertain that only the last five algorithms are supported by MS-Office. Initially, the tests have been performed on a *.docx* file, which has been compressed by using the different compression algorithms. It has been determined that both MS-Office 2007 and MS-Office 2010 do not correctly handle compressed file with the following compression switches: *eb*, *ee*, *el*, *ep*, *ez*. In such

a case, it is shown an error message stating that the ZIP format is not supported. The proposed steganographic technique considers different compression algorithms as different parameters of source encoding [2]. The hidden data is encoded with an alphabet of five elements, the five different values that indicate the compression algorithm used. Then we will apply:

- at the first file, stored inside the ZIP container, the compression algorithm matching the first character of the hidden message;
- at the second file, stored inside the ZIP container, the compression algorithm matching the second character of the hidden message;
- ...
- at the last file, stored inside the ZIP container, the compression algorithm matching the last character of the hidden message.

In this way the compression algorithm used to ZIP the single file, corresponds to the value of the character to be hidden.
More generally, if the MS-Office file contains M files, the proposed technique allows to hide:

$$\log_2 5^M = M \cdot \log_2 5 \approx M \cdot 2.32$$

bits of information. M is at least 12, but usually is greater.

Example 4.3.0.1 *Consider the binary string* $(10101011011111110010001000001)_2$ *to be hidden in a Word document which has just been created and has no characters. This document is made up of 12 files, as listed in the first column of Table 4.2.*

The files are listed in alphabetical order in relation to their "absolute" name (comprehensive of the path). Thus, there is an univocal sequence on which it encodes or decodes. In order to hide the binary string, it has to be first converted into a number in base 5. The base 5 representation of the number $(10101011011111110010001000001)_2$ *is a string of 12 numbers:* $(332013432413)_5$*. It is assumed that the values indicated*

File	Algorithm	*Char*
$[Content\_Types].xml$	$DeflateS$	3
$\backslash docProps\backslash app.xml$	$DeflateS$	3
$\backslash docProps\backslash core.xml$	$DeflatX$	2
$\backslash word\backslash document.xml$	$DeflatF$	0
$\backslash word\backslash fontTable.xml$	$DeflatN$	1
$\backslash word\backslash settings.xml$	$DeflateS$	3
$\backslash word\backslash styles.xml$	$Stored$	4
$\backslash word\backslash stylesWithEffects.xml$	$DeflateS$	3
$\backslash word\backslash webSettings.xml$	$DeflatX$	2
$\backslash word\backslash theme\backslash theme1.xml$	$Stored$	4
$\backslash word\backslash \_rels\backslash document.xml.rels$	$DeflatN$	1
$\backslash \_rels\backslash .rels$	$DeflateS$	3

Table 4.2: Decoding table.

*in Table 4.1 (column "*Char*") can be associated to the various compression algorithms. In order to obtain the stego-text, every file will be simply compressed with the corresponding algorithm associated to the character to be hidden (see Table 4.2).*

Algorithms 1 and 2 show how to encode and decode information using the methodology of different compression algorithm of ZIP.

Algorithm 1 Sequence of steps to encode secret message inside ZIP structure

X is the MS-Office file that will contain the secret message
S is the secret message
C is the encoding table
F is the array of strings that lists the absolute name of the files stored in ZIP container

X = MS-Office file
F = array of strings which stores the absolute name of the files within the ZIP container (the elements of F are alphabetically ordered)
M = number of elements of F (the maximum number of characters that will be hidden)
input S
B = encode S in base 5
L = number of character of B
if $L > M$ **then**
  print "Too many characters to hide"
else
  for $i = 1$ to M **do**
    if $i \leq L$ **then**
      X = append the file named $F[i]$ using compression algorithm $C[B[i]]$
    end if
  end for
end if

Algorithm 2 Sequence of steps to decode secret message from ZIP structure

X is MS-Office file that contains the secret message
S is the secret message
C is the decoding table
F is the array strings that lists the absolute name of the files stored in ZIP container

S = null
X = MS-Office file
F = array of string that stores the absolute name of the files within the ZIP container (the elements of F are alphabetically ordered)
M = number of elements of F (the maximum number of characters that will be hidden)
for $i = 1$ to M **do**
  $B[i] = 0$
end for
for $i = 1$ to M **do**
  K = algorithm used to compress the file $F[i]$
  $B[i]$ = value associated to K in the C
end for
S = decode B from base 5
print S

References

[1] P. Deutsch, "DEFLATE Compressed Data Format Specification version 1.3". http://www.ietf.org/rfc/rfc1951.txt, May 1996.

[2] A. Castiglione, B. D'Alessio, A. De Santis and F. Palmieri, "New Steganographic Techniques for the OOXML File Format". *Availability, Reliability and Security for Business, Enterprise and Health Information Systems*, vol.6908, pages 344-358, 2011.

[3] S.L. Garfinkel, J.J. Migletz, "New XML-Based Files: Implications for Forensics" *Digital Forensics - IEEE Security & Privacy*, pages 38-44, March-April 2009.

Chapter 5

Information Hiding by XML

5.1 Introduction

The Extensible Markup Language (XML) is used for the representation of structured data and document on digital supports [3]. It is a language of markup and it is composed of instructions, defined as *tags* or *markers*. In XML a document is described, in form and content, starting from a sequence of elements where:

- every element is defined from a *tag* or a couple of *start-tag*, *end-tag*;
- every element can have one or more attributes (this attributes are used to define some properties).

The known methods to hide information inside XML files are various. They can be simple, for example using comment lines inside the code [4], or complex, like the techniques described from various authors ([2] and [5]). In the follow chapters all these methodologies are analysed, with a special attention to the possibility of being used in MS-Office documents.

5.2 Information Hiding by XML Comment

One of the simplest method to hide information inside XML files is to insert comment lines that contain the secret message. Standard XML uses the command "`<!-text ->`" to insert comments in the code. The inserted "text" will be the hidden

information. In the tests performed it has been verified that it is possible to insert comment with the string of "text" having around 10,000,000 characters. Even though the dimension of the hidden message is sufficiently large, it is advisable to distribute the secret message in more than one comment line. Such methodology can be easily found simply using a system that will search the comment lines in XML files. Only the presence of comment lines inside XML files contained in MS-Office documents, is an indication of the presence of hidden data because the OOXML standard does not use comments.

5.3 Representation of Empty Elements

In the XML documents it is possible, for some kinds of XML elements, to insert a *end-tag* immediately after a *start-tag*. This sequence of tags, that doesn't produce anything, is called "empty elements". They could be used to memorize hidden data. The methodology based on this propriety is described in the following. A XML document corresponds to the ordered sequence of XML elements. Then, it is possible to insert an "empty element", after every occurrence of XML elements (for some kinds), without any results. So, these XML elements could store one bit of value:

- 1, if there is an "empty element";
- 0, if there isn't an "empty element".

This methodology allows to hide small data because it depends on the number of XML elements in the file.

Example 5.3.0.1 *Consider, as an example, the XML element "*`w:t`*" used to insert a frame of text. Suppose that inside the XML document, that will store the secret message, there are five XML elements "*`w:t`*" (that appear in the following order of occurrence).*

```
<w:t xml:space="preserve">All the information contained in a plain-text document are
     visible to everybody. </w:t>
<w:t xml:space="preserve">...</w:t>
<w:t xml:space="preserve">Those information could be</w:t>
<w:t>exploited by third party for illegal purposes.</w:t>
<w:t>This is an example of text written with word 2010</w:t>
```

Let's apply the methodology illustrated in this chapter. The stego-key *could be:*

```
<w:t>test</w:t>              0
<w:t>test</w:t><w:t></w:t>  1
```

and therefore the stego-data *will be:*

```
<w:t xml:space="preserve">All the information contained in a plain-text document are
     visible to everybody. </w:t>
<w:t Xml:Space="Preserve">...</w:t>
<w:t Xml:Space="Preserve">Those information could be</w:t><w:t></w:t>
<w:t>exploited by third party for illegal purposes.</w:t><w:t></w:t>
<w:t>This is an example of text written with word 2010</w:t>
```

resulting in the embedded data to be: `00110`.

5.4 White Space in Tag

Another technique that can be used to hide data in XML files is to insert spaces inside a tag. It is important to notice that the space cannot be inserted anywhere in the XML files. In fact, placing a space in the wrong place would result with an error message "command interpretation". In MS-Office the space can be inserted, for example in the *end-tag* before the character ">". This methodology, with the same number of XML elements, allow to conceal up to the double of data that could be hidden using the technique described in Chapter 5.3. Actually, to each couple *start-tag/end-tag* will be assigned two values and not one as in Chapter 5.3.

Example 5.4.0.1 *Consider the couple* start-tag/end-tag *"<w:t> </w:t>", like the example in Chapter 5.3. We consider that inside the XML document, that we will use to store the secret message,* 5 *XML elements "w:t" are present (that appear in the following order of occurrence).*

```
<w:t xml:space="preserve">All the information contained in a plain-text document are
     visible to everybody. </w:t>
<w:t xml:space="preserve">...</w:t>
<w:t xml:space="preserve">Those information could be</w:t>
<w:t>exploited by third party for illegal purposes.</w:t>
<w:t>This is an example of text written with word 2010</w:t>
```

Let's apply the methodology illustrated in this chapter. The stego-key *could be:*

```
<w:t > or </w:t >     1
<w:t> or </w:t>       0
```

Therefore the stego-data *will be:*

```
<w:t xml:space="preserve" >All the information contained in a plain-text document are
     visible to everybody. </w:t>
<w:t xml:space="preserve">...</w:t >
<w:t xml:space="preserve">Those information could be</w:t >
<w:t>exploited by third party for illegal purposes.</w:t>
<w:t >This is an example of text written with word 2010</w:t >
```

resulting in the embedded data to be: `1001010011`.

5.5 Appearing Order of Elements

A XML document appears like an ordered sequence of XML elements. There are some elements with a specified property: exchanging of the appearing order of elements produces the same output in the XML code. Starting from this observation, Inoue et al. created a new methodology. They proposed to hide information inside XML documents attributing different values to the various "orders of apparition" of the elements. This technique can be successfully applied to MS-Office files but it is difficult to define a general rule. For this reason, only detailed analysis of all the XML elements used from MS-Office will give a precise indication of where to hide the information. An example of XML elements that satisfy the "exchanging of the appearing order of elements" propriety are: `<w:autoSpaceDE w:val="0"/>` and `<w:autoSpaceDN w:val="0"/>`. These two elements specify whether inter-character spacing shall automatically be adjusted between regions of Latin text and regions of East Asian text in the current paragraph. These regions shall be determined by the Unicode character values of the text content within the paragraph [1]. Based on the conducted

tests (see Chapter 10) it is possible to verify that only few `<w:autoSpaceDE...>` and `<w:autoSpaceDN...>` are found in the MS-Office files. Then, using this couple of elements only, is it possible to conceal only some information's bits. In order to increase the amount of data to be hidden it is necessary to apply the methodology using different XML elements, that satisfy the "exchanging of the appearing order of elements" property.

Example 5.5.0.1 *Consider* `w:autoSpaceDE` *and* `w:autoSpaceDN` *as couple of elements. Supposes to represent value* 1 *with the sequence:*

```
<w:autoSpaceDN w:val="0"/>
<w:autoSpaceDE w:val="0"/>
```

and value 0 *with the sequence:*

```
<w:autoSpaceDE w:val="0"/>
<w:autoSpaceDN w:val="0"/>
```

If in the document under investigation there are 3 *occurrences of the chosen couple of elements, the* stego-key *could be:*

```
<w:autoSpaceDE w:val="0"/>
<w:autoSpaceDN w:val="0"/>
                              value   0
<w:autoSpaceDN w:val="0"/>
<w:autoSpaceDE w:val="0"/>
                              value   1
```

Therefore the stego-data *will be:*

```
<w:autoSpaceDN w:val="0"/>
<w:autoSpaceDE w:val="0"/>
...                                   1
<w:autoSpaceDN w:val="0"/>
<w:autoSpaceDE w:val="0"/>
...                                   1
<w:autoSpaceDE w:val="0"/>
<w:autoSpaceDN w:val="0"/>
                                      0
```

resulting in the embedded data to be: `110`.

5.6 Appearing Order of Attributes

The sequence order of attributes of an element does not influence the result. Therefore, changing the order of the attributes, we obtain different XML elements that produce the same result. Starting from this property, Inoue et al. propose a new methodology in order to hide a secret data, in XML documents, by exchanging of the appearing order of attributes in the element. We have verified (see tests in Chapter 10) that this technique, generally, is applicable also to the MS-Office files. Only a specific analysis that examines all XML elements, with multiple-attributes used in MS-Office, can detect if it is applicable everywhere. An example of XML element with more attributes is `<w:rFonts>`. It specifies the font which shall be used to display the text and can be used to hide text in a multiple-attributes element.

If m is the number of attributes on which to apply this methodology, the secret message can be encoded using an alphabet of t characters, where $t = m!$ (number of simple permutations). As an example, if we use an XML element with 2 attributes ($m = 2$), the message will be codified using an alphabet of $t = 2!$ characters, that is a binary code. Using only an element with multiple attribute allows to store few bits. So, to increase the amount of hidden information, it is necessary to use different type of elements with multiple attributes.

Comparing this methodology with the one's illustrated in Chapter 5.4, both applied only to the XML elements with multiple-attributes, we verify that the first technique makes it possible to hide more information. In fact, if we have n XML elements everyone with m attributes, using this methodology n characters of an alphabet of $m!$ characters can be hidden, without additional overhead, while applying the other technique n bits introducing an overhead of n characters can be hidden. It is important to notice that the frequency of occurrences of XML elements with multiple attributes is less than all the other elements able to apply the technique cited in Chapter 5.4.

Example 5.6.0.1 *Consider the element* `<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs= "AdvEPSTIM"/>`*. It has three attributes, so it is possible to use an alphabet of six characters:*

```
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>   A
<w:rFonts w:ascii="AdvEPSTIM" w:cs="AdvEPSTIM" w:hAnsi="AdvEPSTIM"/>   B
<w:rFonts w:hAnsi="AdvEPSTIM" w:ascii="AdvEPSTIM" w:cs="AdvEPSTIM"/>   C
<w:rFonts w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM" w:ascii="AdvEPSTIM"/>   D
<w:rFonts w:cs="AdvEPSTIM" w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM"/>   E
<w:rFonts w:cs="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:ascii="AdvEPSTIM"/>   F
```

If in the document under investigation there are 8 *occurrences of the chosen elements, as follow:*

```
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>
```

therefore the stego-data *will be:*

```
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>   A
<w:rFonts w:ascii="AdvEPSTIM" w:cs="AdvEPSTIM" w:hAnsi="AdvEPSTIM"/>   B
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>   A
<w:rFonts w:hAnsi="AdvEPSTIM" w:ascii="AdvEPSTIM" w:cs="AdvEPSTIM"/>   C
<w:rFonts w:cs="AdvEPSTIM" w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM"/>   E
<w:rFonts w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM" w:ascii="AdvEPSTIM"/>   D
<w:rFonts w:cs="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:ascii="AdvEPSTIM"/>   F
<w:rFonts w:ascii="AdvEPSTIM" w:hAnsi="AdvEPSTIM" w:cs="AdvEPSTIM"/>   A
```

resulting in the embedded data to be: `ABACEDFA`*.*

References

[1] ECMA International, "Final draft standard ECMA-376 Office Open XML File Formats - Part 1". *ECMA International Publication*, December 2008.

[2] S. Inoue, K. Makino, I. Murase, O. Takizawa, T. Matsumoto and H. Nakagawa, "A Proposal on Information Hiding Methods using XML. The First NLP and XML Workshop". 2001.

[3] A. Zisman, "An overview of XML". *Computing Control Engineering Journal*, vol.11(4), pages 165-167, August 2000.

[4] S. Mohamma, "A New Method for Steganography in HTML Files". *Advances in Computer, Information, and Systems Sciences, and Engineering*, pages 247-252, 2006.

[5] N. Mir and S.A. Hussain, "Secure web-based communication". *Procedia Computer Science, World Conference on Information Technology*, vol.3, pages 556-562, 2011.

Chapter 6

Information Hiding by Characteristics and Formats

6.1 Introduction

The MS-Office suite offers a lot of features to process the documents (worksheet, slide or text). Such features, that can have effect on the entire document or only on a parts of it, make unique versions of the same file. Some of these features modify, in properties and in characteristics, the contents of the MS-Office document. We consider, for example, the format of the page, the font, the dimension of the character, the style, the inter-word space, the revision tracking and the property of the cells. In the following, some methodologies that use these proprieties to hide information are described.

6.2 Text Color on Color

This methodology, applicable to all MS-Office documents, consists in adding text with "font color" equal to the "page color" [6]. The visual effect is that the text cannot be distinguished from the background and therefore the text is hidden. Clearly, it is easy to discover the presence of an hidden message. What follows are two possible way of discovering the presence of an hidden message.

1. Activate all the options in the section "When correcting spelling and grammar in Word" ("Review" → "Language Preferences" → "Proofing") and change the language in use in the document ("Review"→"Set Proofing Language") in order to make appear red-colored the wrong words that are not recognized the active dictionary. In such a way also the lines outlined in lack zones will appear. Figure 6.1 shows this anomaly: the window on the left contains the text written in black while the right one contains the text written "color on color" where the red line emphasizes the word "Hidden", invisible in this case.
2. Shows "paragraph marks and other hidden formatting symbols" so that the zones of the document that contain the hidden characters will appear as an anomalous empty space between two not printable characters. Figure 6.2 shows this anomaly:
   (a) the first column shows where the secret message "HIDDEN DATA" (highlighted in yellow) has been inserted;
   (b) the second column shows the hidden text disappeared because the color of the characters is white, the same as the background;
   (c) the third column shows the non printable characters (space "·", tabulation "⟶" and end-paragraph). The yellow area between the special characters "·" (space) and the end-paragraph is an anomaly that indicates hidden characters.

6.3 Information Hiding by Style or Color of Character

Another idea that can be used to hide information in a MS-Office document is to take advantage of the properties of the characters (as an example the style or the color). This approach uses different values of the same properties of a character that produce undetectable results to the human eye ([7] and [8]). Apply a property to a word or to a

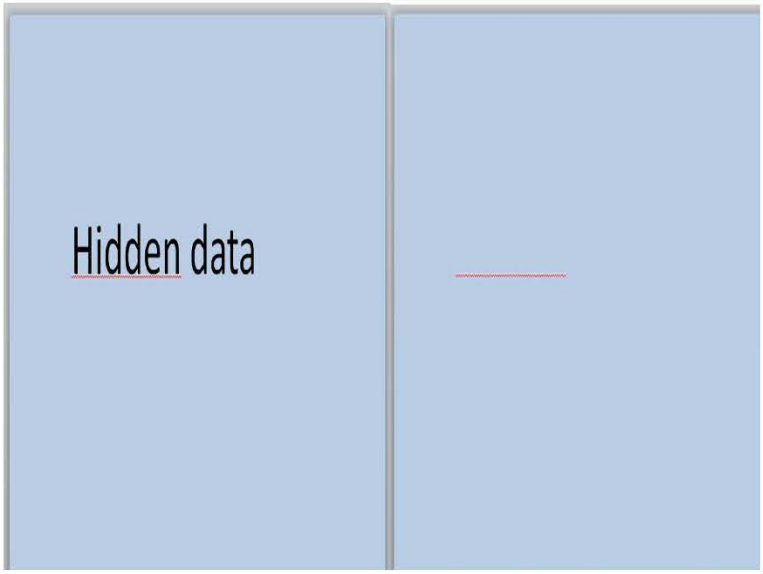

Figure 6.1: Text color on color.

This element specifies a reference to XML content in a format not defined by ISO/IEC 29500. HIDDEN DATA This part allows the native use of other commonly used inter-change formats,	This element specifies a reference to XML content in a format not defined by ISO/IEC 29500. This part allows the native use of other commonly used inter-change formats,	This·element·specifies·a reference · to · XML content·in·a·format·not defined · by · ISO/IEC 29500.· ¶ This · part · allows · the native · use · of · other commonly · used · inter-change·formats,·¶

Figure 6.2: Visualization not printable characters in text color on color.

single characters produces the same visual effect. As an example, it can be associated the value 0, when the property is applied to the entire word and value 1 when the property is applied to the single characters of a word. Another technique is the use of a similar color that cannot be distinguished by the human eye. As an example, it is possible to use different color to encode the secret message, in a similar way as the methodology "least significant bit" applied to graphical files (see Chapter 7.2). Usually the text is written in black. The value of the attribute which defines the black color is "000000" and the value of the nearest color is "000001": in both cases

the human eye will always see "black". In that way, the color of the characters who compose the text to hide will be changed. For example, the value 0 can be associated to the original color and value 1 to the nearest color. The same visual effect is also obtained with other nearest colors (i.e., "000002", "000003", etc.), so we can encode the secret message with an alphabet of more than two characters (i.e. 000000 is the color value associated to the character 0, 000001 is the color value associated to the character 1, 000002 is the color value associated to the character 2 and 000003 is the color value associated to the character 3). In this way the amount of information that can be hidden increases.

Khairullah [9] suggests the idea to apply this technique to a particular kinds of characters. In the MS-Office documents there are some special characters (e.g. the space or the carriage return) which are always invisible, independently from the its color. So it is possible to hide a secret message changing the color, of these invisible characters, without any risk of exposure.

Example 6.3.0.1 *Data concealment using the style of a character. If in a Word document the phrase "`In the middle of the night`" is in "Bold style", the file document.xml will contain the following:*

```
<w:r w:rsidRPr="006A4533">
  <w:rPr>
    <w:b/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">In the middle of the night</w:t>
</w:r>
```

The element `<w:b/>` indicates that the text that follows will be in "Bold style". The text is composed of 6 words resulting in 6 bits of information useful for the hiding process.

Supposing that the secret message is `110100`, the previous XML document will result as:

```
<w:r w:rsidRPr="006A4533">
  <w:rPr>
```

```
    <w:b />
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">I</w:t>
  <w:t xml:space="preserve">n </w:t>
  <w:t xml:space="preserve">t</w:t>
  <w:t xml:space="preserve">h</w:t>
  <w:t xml:space="preserve">e </w:t>
  <w:t xml:space="preserve">middle</w:t>
  <w:t xml:space="preserve">o</w:t>
  <w:t xml:space="preserve">f </w:t>
  <w:t xml:space="preserve">the night</w:t>
</w:r>
```

The same result can be produced using a different module for each word:

```
<w:r w:rsidRPr="006A4533">
  <w:rPr>
    <w:b />
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">I</w:t>
  <w:t xml:space="preserve">n </w:t>
</w:r>
<w:r w:rsidRPr="006A4533">
  <w:rPr>
    <w:b />
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">t</w:t>
  <w:t xml:space="preserve">h</w:t>
  <w:t xml:space="preserve">e </w:t>
</w:r>
<w:r w:rsidRPr="006A4533">
  <w:rPr>
    <w:b />
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">middle</w:t>
</w:r>
<w:r w:rsidRPr="006A4533">
  <w:rPr>
    <w:b />
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">o</w:t>
  <w:t xml:space="preserve">f </w:t>
</w:r>
<w:r w:rsidRPr="006A4533">
  <w:rPr>
    <w:b />
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">the </w:t>
</w:r>
<w:r w:rsidRPr="006A4533">
  <w:rPr>
    <w:b />
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">night</w:t>
</w:r>
```

If we want to conceal a secret message in a PowerPoint document, we will proceed in analogous way. It is important to notice that:

- *the text that appear in the slide n (number of the slide) is contained in the file \ppt\slides\sliden.xml;*
- *to set the bold style it will be assign the value* 1 *to attribute* `b` *in the element* `<a:Pr lang="it-IT" b="1" dirty="0" smtclean="0"/>`.

Example 6.3.0.1 *Data concealment using the color of a character. If in a Word document the phrase "In the middle of the night" is in "Black color", the file document.xml will contain the following:*

```
<w:r w:rsidRPr="00EF289F">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t>In the middle of the night</w:t>
</w:r>
```

The element `<w:color w:val="000000"/>` *indicates that the text that follows will be of black color. The text is composed of* 26 *characters (the spaces are counted) resulting in* 26 *bits of information. We suppose that the binary code, of the message to hide, is* `00100111001011011000010100` *and which we use:*

- *color "*000000*" to conceal the value* 0*;*
- *color "*000001*" to conceal the value* 1*.*

Therefore, the encoding associated to each characters will be:

I	*n*		*t*	*h*	*e*		*m*	*i*	*d*	*d*	*l*	*e*		*o*	*f*		*t*	*h*	*e*		*n*	*i*	*g*	*h*	*t*
0	0	1	0	0	1	1	1	0	0	1	0	1	1	0	1	1	0	0	0	0	1	0	1	0	0
1		2	3		4			5		6	7	8		9	10		11				12	13	14	15	

Grouping the consecutive characters who have the same color are created 15 *groups. In that way, the document under investigation will be divided in* 15 *consecutive modules*

of alternate "000000" and "000001" colors. Therefore in the file document.xml, the previous module will be replaced with:

```
<w:r w:rsidRPr="00B3524A">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">In</w:t>
</w:r>
<w:r w:rsidRPr="0042626E">
  <w:rPr>
    <w:color w:val="000001"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve"></w:t>
</w:r>
<w:r w:rsidRPr="00B3524A">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">th</w:t>
</w:r>
<w:r w:rsidRPr="0042626E">
  <w:rPr>
    <w:color w:val="000001"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">e m</w:t>
</w:r>
<w:r w:rsidRPr="0042626E">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">id</w:t>
</w:r>
<w:r w:rsidRPr="0042626E">
  <w:rPr>
    <w:color w:val="000001"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">d</w:t>
</w:r>
<w:r w:rsidRPr="00B3524A">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">l</w:t>
</w:r>
<w:r w:rsidRPr="0042626E">
  <w:rPr>
    <w:color w:val="000001"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">e</w:t>
</w:r>
<w:r w:rsidRPr="00B3524A">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
```

```
  </w:rPr>
  <w:t xml:space="preserve">o</w:t>
</w:r>
<w:r w:rsidRPr="0042626E">
  <w:rPr>
    <w:color w:val="000001"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">f</w:t>
</w:r>
<w:r w:rsidRPr="00B3524A">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">the</w:t>
</w:r>
<w:r w:rsidRPr="0042626E">
  <w:rPr>
    <w:color w:val="000001"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">n</w:t>
</w:r>
<w:r w:rsidRPr="00B3524A">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">i</w:t>
</w:r>
<w:r w:rsidRPr="0042626E">
  <w:rPr>
    <w:color w:val="000001"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">g</w:t>
</w:r>
<w:r w:rsidRPr="00B3524A">
  <w:rPr>
    <w:color w:val="000000"/>
    <w:lang w:val="en-US"/>
  </w:rPr>
  <w:t xml:space="preserve">ht</w:t>
</w:r>
```

If we want to conceal a secret message in a PowerPoint document, we will proceed in analogous way. It is important to notice that, to set the color, we need to change the value of the attribute `val`*:*

```
<a:solidFill>
  <a:srgbClr val="000000"/>
</a:solidFill>
```

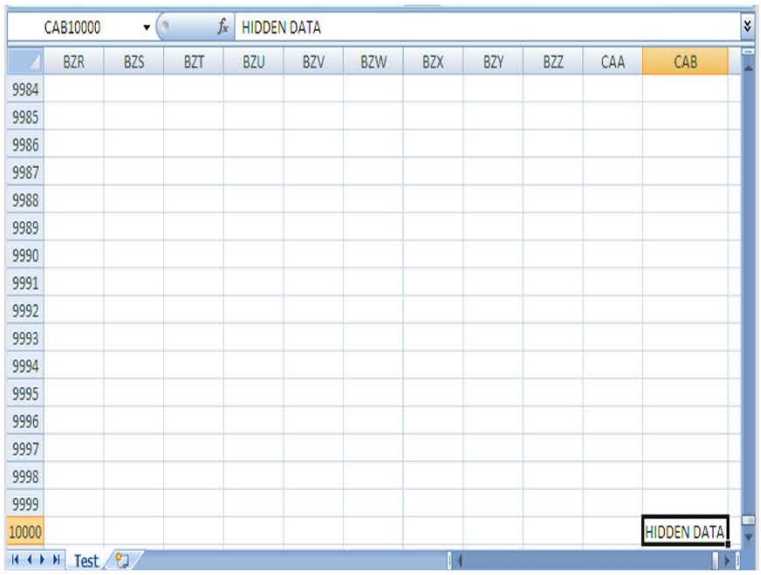

Figure 6.3: Hidden data in far cells.

6.4 Use of Far Cells

A user usually explores the worksheet only in the area where are contained his data. Therefore the information (texts or OLE object), contained in cells far away from this area, difficultly will be visualized. Therefore, the data contained in these far cells are "hidden" to the most users. As an example, a data inserted in the cell with coordinates column "CAB" and line "10000" (see Figure 6.3) will be placed far from the user's view and will be unnoticeable.

It is easy to find secret messages hidden using this technique. In the following are described two ways to discover secret messages embedded using this technique.
The sliding cursor of "horizontal and vertical sliding" bars is smaller than usually (see Figure 6.4). This indicates that the worksheet uses a lot of cells but only an expert and skillful user can notice this anomaly.
Another way to check for the presence of hidden data is to use the display mode "Page Layout". The "Page Layout" shows in white the used sheets and in gray the

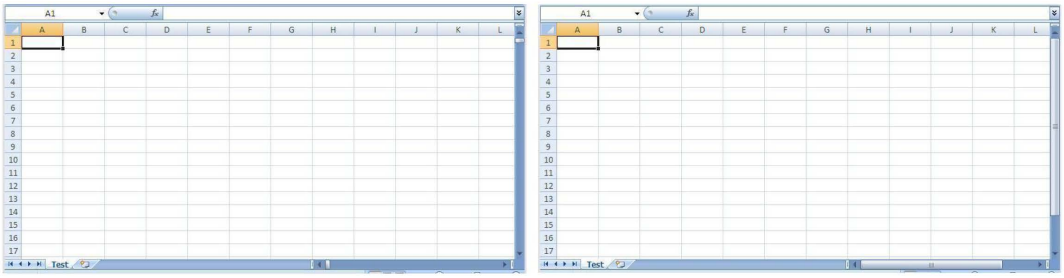

Figure 6.4: Worksheet with and without hidden data.

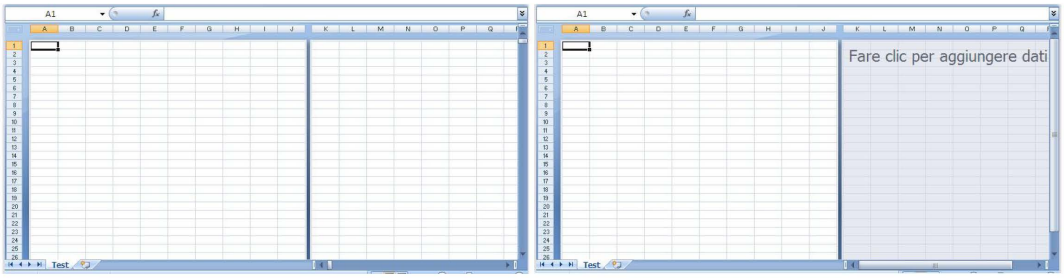

Figure 6.5: Layout of worksheet with and without hidden data.

available ones. The used sheets are all the ones that contain cells in use or that are situated to the left and over the last cell with data. The user, set up such modality using the menu "View", will placed in the lower right corner of its workspace (that is the last cell that contains own data). If the sheets over this cell, those on the right or below it, are all gray indicates that there aren't hidden information. The Figure 6.5 shows as worksheet appears in "Page Layout" with and without hidden data.

The proposed methodology can only be applied to Excel files. In fact it uses the cells, elements present only in worksheets.

6.5 Information Hiding by Change Tracking in Collaborative Document

The strength of the steganographic techniques is that the adversary knows alone the modified document, the one with the inserted secret message. If the adversary knew both documents, the original and the modified one, he could confront them and identify their differences. Such differences would be interpreted as suspicious anomalies. Liu et al. [1] describe a new methodology that changes the described approach. In a such technique, called "change tracking", it is indispensable to know all the version of the document because only analyzing the differences it will retrieve the secret message. It is common, in a collaborative writing MS-Office document, to track the modifications. We consider, for example, a student preparing a thesis and the professor who supervises the work. In such a case the student writes the document and send it to the approval of the professor. Then the professor suggests the corrections and annotations and the student will modify the files involved accordingly. The technique proposed from Liu et al. consists in modifying automatically the initial document, introducing errors and/or synonyms. Therefore, the file will be sent to a reviser who will correct it. It is the codifies/decodes phase of the secret message. In fact, the different modality of correction of the errors or the use of a synonym, in alternative to another word, can be used like code in order to hide information. The techniques usable to insert errors in the document are based on manipulation and transformation of the text. As an example, it can be possible to use common errors and typos, based on the words frequently mistyped (e.g. advise/advice) or techniques based on the substitution of a word with its synonym. To apply this methodology it is important to create a database of synonyms and errors with the occurrence probability of each entry (the occurrence probabilities of the entries, as an example, can be estimated by making use of the Google SOAP Search API [10]). Then it will use the occurrence

probability to encode/decode the secret message (e.g. Huffman code on occurrence probability) [3]. The proposed methodology is difficult to intercept but, to encode or decode the message, is necessary to share the same "database of degeneration". Therefore it is necessary that the database, or the algorithm of degeneration, is communicated using a secure channel among the users.
The implementation of such methodology is limited to Word and PowerPoint documents.

6.6 Information Hiding by Office Macro

A macro is a group of commands which make it possible to obtain a series of operations with a single command [4]. Thus, a macro is a simple recording of sequence of commands which are already available in a software. For this reason, there is no need of using a programming language, but actually MS-Office offers the possibility to create macro also writing VBA code (Visual Basic for Applications) [5]. The new format of MS-Office, as previously stated, in order to guarantee a greater level of security does not allow macro to be saved inside the file. When using macro in documents, it is necessary to enable this function as well as modify the extension of the name file, which will be: *.docm*, *.xlsm*, *pptm*, etc.. The structure of the files with macro (e.g. *example.docm*) and without (e.g. *example.docx*) is different. This is evident when carrying out a simple test: changing the extension of the file from *.docm* to *.docx* and displaying the document, the system gives an error message indicating that the format is not the one expected. However, MS-Office can open the file, recognizing it as a document with macro and processing it as a normal *.docm* file. Thus, it is possible to consider using MS-Office macro as a channel to transmit hidden information [2]. In fact, macro can be seen as a function:

$$F(x) : x \in X, \text{ where } X \text{ is the set of the input of macro.}$$

Therefore, it is possible to hide information:

- in the description of the function $F(x)$;
- in the value associated to the function $F(k)$, where $k \in K$ and $K \subseteq X$ is the set of stego-key that are highly unusual inputs.

In the first case, the information to be hidden will be stored in the macro. For example, it is possible to insert the data to be hidden as a comment to the code or to assign it as a value of a variable. In the second case, as consequence of specific input, macro has a behavior that generates an output that makes the hidden data visible. An example is a macro, in a Word document, that given a word as input, searches for it in the text and highlights it in yellow. There is also another routine in the code, that can only be executed if the searched word is the stego-key, than highlights several characters in the document in yellow. These characters, read in sequence, are the hidden information. In this case the macro will be recognized as reliable by a user as it carries out the task for which it has been realized. Inside the code of the macro, the hidden message does not have to be explicitly stored. So, we can store only the coordinates of the positions, in the document, where we can find the characters that compose the hidden information. In this way, only who has stego-key will know the secret.

This methodology does not place limits on the amount of information that can be hidden. Indeed if there is a macro in the MS-Office file, it is always possible to modify it, in order to hide information, otherwise we can create a new one.

References

[1] T.Y. Liu and W.H. Tsai, "A New Steganographic Method for Data Hiding in Microsoft Word Documents by a Change Tracking Technique". *IEEE Transactions on Information Forensics and Security*, vol.2(1), pages 24-30, 2007.

[2] A. Castiglione, B. D'Alessio, A. De Santis and F. Palmieri, "New Steganographic Techniques for the OOXML File Format". *Availability, Reliability and Security for Business, Enterprise and Health Information Systems*, vol.6908, pages 344-358, 2011.

[3] D.A. Huffman, D.A., "A Method for the Construction of Minimum-Redundancy Codes". *Proceedings of the IRE*, vol.40(9), pages 1098-1101, September 1952.

[4] MSDN Library, "Introduction to macros". http://msdn.microsoft.com/en-us/library/bb220916.aspx, visited March 2011.

[5] R. Stephens, "Visual Basic 2010 Programmers Reference". *Wrox*, March 2010.

[6] A.H. Al-Hamami, M.S. Al-Hakeem and M.A. Al-Hamami, "A Proposed Method to Hide Text Inside HTML Web Page File". November 2009.

[7] , F. Chen and B. Wang, "An algorithm of text information hiding based on font". *Computer Technology and Development.*, vol.16(1), pages 20-22, January 2006.

[8] P. Chen, S. W. Guo and H. L. Chen, "Color-based information hiding algorithm for text documents". *Science Technology and Engineering*, 7(14):pages 3544-3546, July 2007.

[9] M. Khairullah, "A Novel Text Steganography System Using Font Color of the Invisible Characters in Microsoft Word Documents". *Second International Conference on Computer and Electrical Engineering (ICCEE '09)*, 1:pages 482-484, Decembere 2009.

[10] Google, "What is the Google SOAP Search API". http://code.google.com/intl/it-IT/apis/soapsearch/api_faq.html, visited February 2012.

Chapter 7

Information Hiding by "OLE object"

7.1 Introduction

OLE "Object Linking and Embedding" is a framework developed by Microsoft used to make content that is created in one program available in another program. The peculiarity is that the object maintains the original format and then it may be modified and customized to satisfy the users requirement. In the follow chapter we will describe how it is possible to hide information, inside MS-Office files 2007/2010, using the OLE Object.

7.2 Use Steganography in the Attached and Related File

In the new OOXML format the OLE object inserted in the documents are copied, preserving the original format, in the folder *media*. Therefore, the cited files (text, graphic or audio) could be used to hide information by applying the well-known techniques of steganography ([2], [4], [3], [5], [17]). Such techniques are classifiable with respect of the medium types used to conceal the data.
The techniques used to codify secrets messages in "textual files" can be classified in [6]:

- *Open space methods.* The secret message is codified by inserting spaces inside the

text. There are different applications of this technique. For example, it is possible to change the space dimension between the lines, insert space characters at the end of the lines, change the dimension of the single letters that compose the words, etc..

- *Syntactic methods.* To hide information the text is changed using "syntactic variations" without introduce errors. For example, using the punctuation marks in different way, it is possible to obtain two different versions of the same phrase, without modifying the meaning. So we can associate value "0" to a version and value "1" to the other, obtaining a coding system to conceal data.
- *Semantic methods.* To hide information, the text is changed using "semantic variations" without introducing errors. For example, it is possible to replace some words in the text with the respective synonyms. So we can associate value "0" to a word and value "1" to its synonymous, obtaining a coding system to conceal data.

The human eye is unable to perceive the totality of chromatic details and other characteristics stored in files. The methodologies which encode secret messages in "graphic files" are based on this concept. The main used techniques of steganography on graphics files are the following which have been introduced by [1].

- *Least significant bit.* The least significant bit of the byte which describes the color is used to hide information.
- *Masking and filtering.* The image is "marked" as in the watermarking techniques.
- *Transformations.* Using the transformation functions is possible to pass from a discrete domain to another in which there are some redundant bits. Such bits will be used to hide data.
- *Pixel calculator.* The first step is to choose an area of the image in which the pixel color is homogeneous. Then, the system calculates a magic number chosen

among the values of the pixel in the zone under investigation. The final step is to design the secret message using the value of the magic number.

To understand the methodologies that can be used to hide information in "audio files" it is necessary to know two fundamental concepts: the "digital format" of the audio and the "transmission medium" of the audio. The first one can be of three types (*sample quantization*, *temporal sampling rate* and *perceptual sampling*) while the second one, that is the path that connects sender to receiver, can be of four types (*digital end-to-end environment*, *increased/decreased resampling environmental*, *analogue transmission and resampling*, *"over the air" environment*) [11]. Clarified these concepts, it is possible to proceed to illustrate four of the most popular techniques to hide information in audio files.

- *Low bit encoding.* Data is embedded by replacing the least significant bit of each sampling point by a coded binary string [10].
- *Phase coding.* The phase of the original audio signal is replaced with reference phase of the data to be hidden [9].
- *Spread spectrum.* The secret message is transmitted in a narrow-band signal. Over a much larger bandwidth the spectral density of this signal in the channel look like a noise [8].
- *Echo hiding.* The signal is imperceptible degraded. In this way are introduced some disturbances completely similar to environmental conditions [7].

7.3 Objects in Notes and Outside Slide

The users usually explores the document only in the area where their data are contained (see Chapter 6.4). In PowerPoint, as an example, the user usually utilizes only

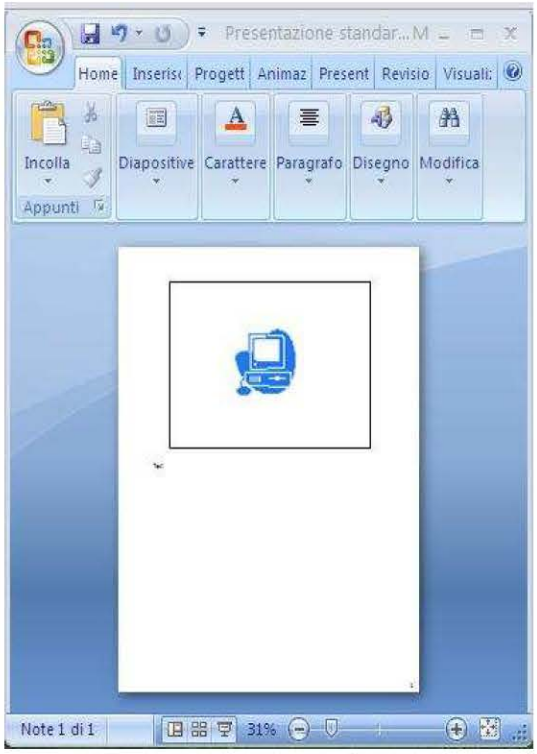

Figure 7.1: Visualization notes.

"Normal" view without insert comments in the notes area. Therefore, the information in notes will be "very probably hidden" because the notes usually remain unread. Another technique in order to hide information can be applied using the visualization mode "Notes Page" (see Figure 7.1). The user, after set up this mode from the menu "View", can notice that there are two areas:

- "white sheet", that contains the slide on the top and the notes on the bottom;
- "gray area", around the "white sheet", in which objects can be inserted.

As a consequence, in order to hide information it is possible to insert the secret message in the "gray area" and later to apply the methodologies shown in Chapter 6.2 and 6.4. As in Chapter 6.4, the sliding cursor of "horizontal and vertical sliding" bars is smaller than usual, but only an expert user can notice this anomaly.

Animation Timing Effect Operations	Number of Variations
Start	3
Delay	*Unlimited*
Speed	5(*default*); *Unlimited*
Repeat	8

Table 7.1: The number of animation timing effect variations.

7.4 Animation Timing Effects in PowerPoint

The animation effects are usually used in the multimedia presentation. In this way it is possible to give more emphasis to some arguments or objects present in the slides. The animations can be applied to several type of objects (text, images, diagrams, etc.) and the effects that can be set up are several (*appear*, *fly in*, *shape*, *pulse*, *wave*, *split*, etc.). In PowerPoint all the animation effects are grouped in 4 categories: *Entrance*, *Emphasis*, *Exit* and *Motion Paths* [13]. The categories *Entrance* and *Exit* group all the effects to animate the entrance and exit of the objects. The *Motion Paths* animation effects move the object inside the slide (e.g. *lines*, *arcs*, *loops*, etc.) while the *Emphasis* animates the object on the place (*pulse*, *lighten*, *teeter*, etc.). Independently from their category, all the animations have commons proprieties that characterized the execution times and the activation mode. Such properties, defined as *timing operation*, have 4 attributes: *Start rule*, *Delay time*, *Speed* and *Repeat time*. The range of values that can assume such attributes is illustrated in Table 7.1.

Jing et al. in their work [16] propose a new methodology of steganography that uses the values associated to the attributes of the timing operation to hide information. This technique requires that sender and receiver share the same method of encode/decode: the *Animation Timing Codebook* (ATC). The ATC is a table in which to every value of animation timing effect is associated the encode character.

Example 7.4.0.1 *Table 7.2 is a ATC example that uses the effects "Delay" and*

Delay	Speed	Symbol
0,1 *seconds*	0,5 *seconds* (*VeryFast*)	0
0,2 *seconds*	0,5 *seconds* (*VeryFast*)	1
0,3 *seconds*	0,5 *seconds* (*VeryFast*)	2
0,4 *seconds*	0,5 *seconds* (*VeryFast*)	3
0,5 *seconds*	0,5 *seconds* (*VeryFast*)	4
0,6 *seconds*	0,5 *seconds* (*VeryFast*)	5
0,7 *seconds*	0,5 *seconds* (*VeryFast*)	6
0,8 *seconds*	0,5 *seconds* (*VeryFast*)	7
0,9 *seconds*	0,5 *seconds* (*VeryFast*)	8
1 *seconds*	0,5 *seconds* (*VeryFast*)	9
0,1 *seconds*	1*second* (*Fast*)	*A*
0,2 *seconds*	1*second* (*Fast*)	*B*
0,3 *seconds*	1*second* (*Fast*)	*C*
0,4 *seconds*	1*second* (*Fast*)	*D*
0,5 *seconds*	1*second* (*Fast*)	*E*
0,6 *seconds*	1*second* (*Fast*)	*F*

Table 7.2: ATC for hexadecimal alphabet.

"Speed" to encode an hexadecimal alphabet.

Therefore, after having shared the ATC it is to modify, in the slides, the values of the animation timing effects. This value will be that associates to the character to hide in the ATC. As an example, there is a slide with three "Text Box" with animation effects and it needs to hide the value: `A2F`*. Using the ATC (see Table 7.2), it will be necessary to set up the values as follow:*

- *"Delay"=0,1 and "Speed"=1 are the parameters of the animation timing effect to set to the first "Text Box" (it encode the character "A");*
- *"Delay"=0,3 and "Speed"=0,5 are the parameters of the animation timing effect to set to the second "Text Box" (it encode the character "2");*
- *"Delay"=0,6 and "Speed"=1 are the parameters of the animation timing effect to set to the third "Text Box" (it encode the character "F").*

It is to note that "Speed" and "Delay" can assume unlimited value but some of them can be perceived from the human eye like "unnatural": its anomalous. So, it is suggested to chose the values in the interval $0,1$ *and* 1 *second to make "natural" effects.*

7.5 Combination of Objects in PowerPoint

The object inserted in a PowerPoint slide can be grouped and becames a kind of virtual object [14]. In such a way the transformations applied to the virtual object will be propagated to the single objects that compose it. As an example, there is a slide with ten "Text Box". If it needs to change colour background to all the "Text Box", it can group all and then change colour only to the virtual object. Normally it would be necessary to select the first "Text Box" and to change the background colour, therefore to repeat this operations for the others nine cases. Zhang et al. [15] propose to codify the hide information using different combinations of objects in a PowerPoint. In this way it can encode different values if an object is included or not included in a group of objects. Also this technique requires that sender and receiver share the table of encode/decode and the rule to order the objects.

Example 7.5.0.1 *It uses a slide with four "Text Box". To hide information using the different combinations in which can be grouped the objects in the slide. The first step is to make a table of encode/decode. In the table is associate an unique sequence of bits to everyone combination of the objects (see Table 7.3). The second step, necessary to identify the different objects* $O_1 \dots O_n$ *in the slide, a rule to order this objects is to define (as an example can be ordered according to the value of coordinate (x,y)). To encode the secret message* `101` *it groups the object* O_1, O_2, O_4*, as indicate in Table 7.3.*

Binary Sequence	Objects included in combinations
000	O_2, O_3, O_4
001	O_1, O_4
010	O_1, O_3
011	O_1, O_3, O_4
100	O_1, O_2
101	O_1, O_2, O_4
110	O_1, O_2, O_3
111	O_1, O_2, O_3, O_4

Table 7.3: Table of encode/decode.

7.6 Information Hiding by Zero Dimension Image

The methodology proposed in this Section uses an OLE object (of type "image"), inserted into a MS-Office document, in order to contain the information to be hidden [2]. This object, which is totally compatible with the OOXML standard, will:

- be located in the upper-left position and placed in any of the pages that make up the document;
- have both the height and width equal to 0;
- be placed "behind the text".

These properties will make it possible to hide the image during the display or modification of the document. It is worth noting that the file associated to OLE object, even if declared as "image", can in reality be any type of file (text, audio, etc.) with a appropriate extension (*.jpg*, *.bmp*, etc.). Therefore, this methodology can be used in order to hide data of a different nature, and is not only limited to images. The identification of the OLE object and the decoding of the hidden text make it more difficult to associate files of reduced dimensions and encrypt the message to be hidden.

A simple and fast method to hide information using this methodology is the following:

- rename the file which contains the hidden message with an extension compatible with an image type;
- insert the image introduced in the previous step into the Word, Excel or PowerPoint document;
- modify the layout of the text related to the image, setting the "Behind text style";
- move the image to the upper-left position;
- from the menu "Dimension and position" set both the height and width of the image to 0.

The folder where to copy the OLE object, associated to the file, varies according to the type of MS-Office document worked on, with it being: *word\media* for Word files, *xl\media* for Excel files, and *ppt\media* for PowerPoint files.
Another way of applying such methodology is to work directly on the XML files. In this case, it is necessary, besides copying the file containing the message to hide in the proper directory (of the ZIP container), to insert in the XML files the elements to:

- relate to the image;
- declare the presence of the image;
- set the position of the image on the upper-left;
- set the image placed behind the text;
- set the dimensions of the image equal to zero.

In order to set the dimensions of the image to zero, the XML `extent` attribute will have to be worked on (see pp. 3173-3176 of the [12]). This element, in fact, defines the dimension of the bounding box that contains the image. Therefore, reducing the

height and width of the bounding box to zero, will obtain the desired effect. Two examples of the `extent` element, respectively for Word and Excel files, are shown:

```
<wp:extent cx="0" cy="0" />
<a:ext cx="0" cy="0" />
```

where attributes `cx` and `cy` are respectively, the width and height of the bounding box. In the Excel files, among the elements used to describe the image inserted in the spreadsheet, there are:

```
<xdr:from>
   <xdr:col>0</xdr:col>
   <xdr:colOff>9525</xdr:colOff>
   <xdr:row>0</xdr:row>
   <xdr:rowOff>28575</xdr:rowOff>
 </xdr:from>
<xdr:to>
   <xdr:col>0</xdr:col>
   <xdr:colOff>161925</xdr:colOff>
   <xdr:row>0</xdr:row>
   <xdr:rowOff>28575</xdr:rowOff>
 </xdr:to>
```

These elements identify the box of cells that contains the image (see pp. 3516-3517, 3523-3524 and 3532-3533 of the [12]). The coordinates (row, column) are relative to the two cells situated respectively in the upper-left and lower-right. Therefore, in order to reduce the dimensions of the image to zero, it is sufficient to reduce the box of cells that contains it (`<xdr:col>0` and `<xdr:row>0`) to zero. Thus, there is no need to place the image in the upper-left position due to it already being in a not selectable position: the cell with the coordinates (0,0).

In order to set the image in the upper-left position of the page, for Word files, it will be necessary to operate on the `position` element (see pp. 3480-3483 of the [12]). This element indicates the position of the image in respect to a part of the document (page, column, paragraph). Therefore, placing the image at a distance 0 of the "page" will obtain the desired effect. An example of how the block of elements on which the modification operates, is the following:

```
<wp:positionH relativeFrom="column">
<wp:posOffset>1685925</wp:posOffset>
<wp:positionV relativeFrom="page">
<wp:posOffset>>967105</wp:posOffset>
```

The attribute `relativeFrom` indicates the part of the document in relation to which the position will be calculated while `posOffset` is the position. Therefore, upon placing the image on the left, the following elements will be modified as:

```
<wp:positionH relativeFrom="page">
<wp:posOffset>0</wp:posOffset>
<wp:positionV relativeFrom="page">
<wp:posOffset>>0</wp:posOffset>
```

In order to place the image in the upper-left position, the `<a:off x="0" y="0"/>` element cannot be used due to the position indicated by the x and y coordinates referring to the paragraph and not to the page.

There is a problem for PowerPoint files, where the image, also if reduced to dimension zero and placed in the upper-left position, could still be selected by using the "Select Area" function. Moreover, it is impossible to insert an image outside a slide. In fact, the image would be interpreted as an anomaly by the *Document Inspector*. This methodology, therefore, is not really suitable for PowerPoint files.
Algorithms 3 and 4 show how to encode and decode information using the methodology of Zero Dimension Image.

7.7 Information Hiding by Overlapping Two Images

One of the limits of the *Document Inspector*, is the impossibility to find and to remove objects covered from others. Taking advantage of such vulnerability a new methodology to hide information in a MS-Office document overlapping two images it proposes. The image in background, that contain the secret message, will be totally covered to the foreground image, so it will be concealment. A simple and fast method to hide information using this methodology is the following:

- insert two images in the document;
- resize the image that contains the secret message, it will have a dimension less or equal than to the other image;

Algorithm 3 Sequence of steps to encode secret message by using the zero dimension image methodology

S is the file that contains the secret message
REL is the file that contains the relationships of the OLE-object inserted
MED is the folder that contains the OLE-objects
CEN is the file that contains the elements to declare as an image

if file extension == ".docx" **then**
  REL = word_rels\document.xml.rels
  MED = word\media
  CEN = word\document.xml
else
  if file extension == ".xlsx" **then**
    REL = xl\drawings_rels\drawings.xml.rel
    MED = xl\media
    CEN = xl\drawings\drawings.xml
  else
    if file extension == ".pptx" **then**
      REL = ppt\slides_rels\slide1.xml.rel,
      MED = ppt\media
      CEN = ppt\slides\slide1.xml
    else
      goto end
    end if
  end if
end if
copy S into the folder MED
in the REL file add the relationships element
in the CEN file add the elements to declare as an image
in the CEN file set the image behind text
in the CEN file set the image dimension equal zero
in the CEN file set the image in the left corner

Algorithm 4 Sequence of steps to decode secret message by using the zero dimension image methodology

F is the absolute name of the file that contains the secret message in ZIP container
S is the file that contains the secret message
REL is the file that contains the relationships of the OLE-object inserted
MED is the folder that contains the OLE-objects
CEN is the file that contains the elements to declare as image

if file extension == ".docx" **then**
  REL = word_rels\document.xml.rels
  MED = word\media
  CEN = word\document.xml
else
  if file extension == ".xlsx" **then**
    REL = xl\drawings_rels\drawings.xml.rel
    MED = xl\media
    CEN = xl\drawings\drawings.xml
  else
    if file extension == ".pptx" **then**
      REL = ppt\slides_rels\slide1.xml.rel,
      MED = ppt\media
      CEN = ppt\slides\slide1.xml
    else
      goto end
    end if
  end if
end if
in the CEN file searches for an image with dimension equal zero
if result search == TRUE **then**
  in the CEN file search the "rId" that refers to image with dimension equal to zero
  J = value of the "rId" found
  in the REL file search the "Target" value assigned for "rId" equal to J
  F = value of the "Target" found
  S = file referenced by F
else
  print "There is no secret message in file"
end if

- make in background the image that contain the secret message;
- overlapping the images;
- group the two images, so the user see only one.

As in Chapter 7.6, the file associated to the OLE object image, even if declared as "image", can be any type of file (text, audio, etc.) with an appropriate extension (.jpg, .bmp, etc.). Therefore, this methodology can be hide data of a different nature, and is not only limited to images. The image can be resized in two ways: change the value in the menu "Size and Position" of the image or modify the XML elements inside the files *word\document.xml* (for Word), *ppt\slides\slide1.xml* (PowerPoint) and *xl\drawings\drawings.xml* (for Excel). In this last case, it will set up the values x and y.

```
<!--dimension of the group-->
- <a:xfrm>
   <a:off x="971600" y="1275064"/>
   <a:ext cx="1800000" cy="1800000"/>
   <a:chOff x="971600" y="1275064"/>
   <a:chExt cx="1800000" cy="1800000"/>
 </a:xfrm>
...
<!--dimension image background-->
- <a:xfrm>
   <a:off x="971600" y="1275064"/>
   <a:ext cx="1800000" cy="1800000"/>
 </a:xfrm>
...
<!--dimension image foreground-->
- <a:xfrm>
   <a:off x="971600" y="1275064"/>
   <a:ext cx="1800000" cy="1800000"/>
 </a:xfrm>
```

Only in Excel files, will it be necessary to resize the box of cells that contains the grouped images. Thus, the values of row and column will be changed in the following elements:

```
- <xdr:from>
    <xdr:col>1</xdr:col>
    <xdr:colOff>9525</xdr:colOff>
    <xdr:row>1</xdr:row>
    <xdr:rowOff>28575</xdr:rowOff>
  </xdr:from>
- <xdr:to>
    <xdr:col>10</xdr:col>
    <xdr:colOff>304800</xdr:colOff>
```

```
    <xdr:row>20</xdr:row>
    <xdr:rowOff>114300</xdr:rowOff>
  </xdr:to>
```

The values x and y and the row and column of the background image will have always values less or equal then to the other image. Such methodology can be easily found by an expert and much careful user. If it selects the image and click with the right button of the mouse, the menu "Group" is able. This means that the element is a part group of objects.

References

[1] S.M. Thampi, "Information hiding techniques: a tutorial review". *ISTE-STTP on Network Security & cryptography*, 2004.

[2] , G.C. Kessler, "An Overview of Steganography for the Computer Forensics Examiner". *Federal Bureau of Investigations, viewed 28*, 2004.

[3] N.F. Johnson and S.C. Katzenbeisser, "A Survey of Steganographic Techniques". *Information Hiding Techniques for Steganography and Digital Watermarking*, pages 43-78, 2000.

[4] S.K. Bandyopadhyay, D. Bhattacharyya, P. Das, D. Ganguly and Swarnendu Mukherjee, "A tutorial review on Steganography". *International Conference on Contemporary Computing (IC3-2008), Noida, India*, pages 105-114, 7-9 August 2008.

[5] E.T. Lin and E.J. Delp, "A Review of Data Hiding in Digital Images". *PICS'99*, pages 274-278, 1999.

[6] W. Bender, D. Gruhl, N. Morimoto and A. Lu, "Techniques for data hiding". *IBM Systems Journal*, vol.35, pages 313-336, 1996.

[7] D. Gruhl, A. Lu and Walter Bender, "Echo hiding". *Information Hiding*, vol.1174, pages 295-315, 1996.

[8] I.J. Cox, J. Kilian, F.T. Leighton and T. Shamoon, "Secure spread spectrum watermarking for multimedia". *IEEE Transactions on Image Processing*, vol.6(12), pages 1673-1687, 1997.

[9] Y. Yardimci, A. Enis Çetin and R. Ansari, "Data hiding in speech using phase coding". *EUROSPEECH*, 1997.

[10] N. Cvejic and T. Seppanen, "Increasing robustness of LSB audio steganography using a novel embedding method". *International Conference on Information Technology: Coding and Computing (ITCC2004)*, vol.2, pages 533-537, April 2004.

[11] B. Dunbar, "A Detailed look at Steganographic Techniques and their use in an Open-Systems Environmen". *SANS Institute InfoSec Reading Room*, January 2002.

[12] ECMA International, "Final draft standard ECMA-376 Office Open XML File Formats - Part 1". *ECMA International Publication*, December 2008.

[13] Microsoft Corporation, "Animate text or objects. Applies to: Microsoft PowerPoint 2010". http://office.microsoft.com/en-us/powerpoint-help/animate-text-or-objects-HA010336726.aspx, visited February 2012.

[14] Microsoft Corporation, "Group or ungroup shapes, pictures, or other objects in PowerPoint 2007". http://office.microsoft.com/en-us/powerpoint-help/group-or-ungroup-shapes-pictures-or-other-objects-in-powerpoint-2007-HA010201377.aspx, visited February 2012.

[15] H. Zhang, L. Huang, Y. Ye and P. Meng, "A new steganography method via combination in PowerPoint files". *International Conference on Computer Application and System Modeling (ICCASM2010)*, vol.2, pages 62-66, October 2010.

[16] M.Q. Jing, W-C. Yang and L.H. Chen, "A new steganography method via various animation timing effects in PowerPoint files". *International Conference on Machine Learning and Cybernetics 2009*, vol.5, pages 2840-2845, July 2009.

[17] J.F. Neil and C.S. Katzenbeisser, "A survey of steganographic techniques". *Infor-*

mation Hiding Techniques for Steganography and Digital Watermarking, pages 43-78, 2000.

[18] Microsoft Corporation, "MS-OLEDS: Object Linking and Embedding (OLE) Data Structures". *Information Hiding Techniques for Steganography and Digital Watermarking*, December 2011.

Chapter 8

Information Hiding by OOXML Format

8.1 Introduction

The following chapter describes the methodologies of information hiding based on the characteristic that MS-Office documents are compliant to the OOXML standard [6].

8.2 Information Hiding by Unknown Parts

Such methodology, proposed by Park et al. [5], is based on the idea of adding, inside ZIP container, parts that do not have relationships in the document: it adds new files in OOXML structure that does not appear inside of the relationships files. The presence of these files is interpreted by the MS-Office suite as "problems with the contents". Therefore the document could be visualized only after an automatic recovery operation that will remove the "unknown parts".

It is possible to exceed this limit, and therefore to render the document MS-Office compatible, inserting in the file [*Content_Types*].*xml* the following element: `<Override PartName="/app.txt" ContentType="app.txt"`. Where `app.txt` is the file inserted in the container.

8.3 Information Hiding by Unknown Relationships

The MS-Office suite does not interpret unknown relationships in the XML code. Taking advantage of such a property, Park et al. [5] propose to insert the secret message like unknown relationships.

Those external relationships between parts of the document and file, in MS-Office, are inserted in the file *document.xml.rels* placed in the folder *word_ rels*.

As an example, the following relationship hide the text "HIDDENDATA":

```
<Relationship Id="rId15" Type="http://schemas.openxmlformats.org/officeDocument/2006/
 relationships/HIDDENDATA" Target="media/photo.gif"/>
```

8.4 Information Hiding by Unknown Parts and Unknown Relationships

Park et al. [5] to improve stronger and to hide more information in the methodologies analysed in Chapter 8.2 and 8.3 propose using the two techniques together. Thus, it will be:

- add a file in container ZIP (as an example in folder *media*);
- insert an unknown relationships to the added file like described Chapter 8.3.

So, it will not appear as a "problem of content" described in Chapter 8.2.

8.5 Information Hiding by Revision Identifier Value

Another proposed method of hiding information in MS-Office documents, which is only applicable to Word files, is to use the value of several attributes that are in XML. It is the revision identifier *rsid*, a sequence of 8 characters which specifies a

unique identifier used to track the editing session. An editing session is defined as the period of editing which takes place between any two subsequent save actions. The *rsid*, as an attribute of an XML element, gives information on the part of code contained in the same element. The types of revision identifier, usable in the OOXML standard, are listed in the specifications of the ECMA-376. These attributes, defined as the *ST_LongHexNumber* simple type, are strings of 8 hexadecimal characters:

$$(x_0x_1x_2x_3x_4x_5x_6x_7) : x_i \in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$$

All the revision identifier attributes, present with the same value in a document, indicate that the code in the element has been modified during the same editing session.

An example element which contains 3 *rsid* attributes is:

```
<w:p w:rsidR="000E634E" w:rsidRDefault="008C3D74" w:rsidP="00463DF8">
```

It is worth noting that there are three sequences of 8 characters, that represent the unique identifier associated to the attributes: *rsidR*, *rsidRDefault* and *rsidP* (see pp. 243-244 of the [3]).

The methodology proposed in this chapter consists of replacing the values of the *rsid* attributes with the data to be hidden, codified in hexadecimal [2]. Thus, if T is the number of occurrences of these attributes in the MS-Office files, the maximum number of bits that can be hidden will be:

$$\log_2 16^{T \cdot 8} = 32 \cdot T$$

due to every attribute being composed of 8 hexadecimal characters. If the information to be hidden exceeds the maximum number of bits that can be contained in the MS-Office document, it is possible to add to the XML file further elements with *rsid* attributes. Furthermore, one more trick is required to avoid the detection of hidden data by a stego-analysys inspection. MS-Office records in the file `setting.xml` all the *rsid*

values that has been used in the various versions of the file `document.xml`. To perform such an activity, MS-Office uses the XML element `<w:rsid w:val="002A31DF">`. Consequently, when, to hide information, it is used the methodology presented in this chapter, after having modified the *rsid* values in the file `document.xml`, it is necessary to insert the same values even in the file `setting.xml`. In fact, the presence of *rsid* values in the `document.xml` which are not present in the `setting.xml` it is a strange situation that could raise suspicion.

Among the various functionalities available in MS-Office, there is the possibility to track the changes of a document. By using such feature, MS-Office keeps track of all the modifications performed in a document (deleted, inserted or modified text), of the date when they have been made and of the user who has carried out such modifications. Those information, even though can be partially reconstructed by the analysis of the *rsids*, are traced by using two XML elements. Such elements, delimited by a pair of *start-tag* and *end-tag*, are different if used to track a deletion (with the tag `<w:del ...> </w:del>`) or an insertion (with the tag `<w:ins...> </w:ins>`).

This element has the following three attributes: identification code (*id*), author who modified the document (*author*) as well as time and date in which the change (*date*) occurred (this is an optional attribute). Consequently, all the modifications performed by the same author within the same editing session will be placed in the XML file between the *start-tag* and *end-tag* of the "change-tracking" element.

For example, if the user `PCCLIENT` would have deleted the text "`one`" at `09:23:00 GMT` of `October 11, 2010`, the code excerpt will be like:

```
<w:del w:id="0" w:author="PCCLIENT" w:date="2010-10-11T09:23:00Z">
  <w:r w:rsidRPr="00111111" w:rsidDel="00333333">
     <w:rPr>
        <w:lang w:val="en-US"/>
     </w:rPr>
     <w:delText xml:space="preserve">one</w:delText>
  </w:r>
</w:del>
```

00	46	3D	F8	00	46	3D	F8	00	46	3D	F8	00	74	04	7B	00	46	3D	F8
00	46	3D	F8	00	46	3D	F8	00	8C	3D	74	00	74	04	7B	00	46	3D	F8
00	8C	3D	74	00	46	3D	F8	00	46	3D	F8	00	0E	63	4E	00	46	3D	F8
00	46	3D	F8	00	0E	63	4E	00	46	3D	F8	00	9B	2A	88				

Table 8.1: Sequence of *rsid* values.

That being stated, the methodology presented in this Chapter will continue to work even though the change tracking is activated in MS-Office. Enabling the change tracking means that personal information is inserted into the document. Therefore, the *Document Inspector* signals the presence of the change tracking as an anomaly and proceeds to eliminate this information from the document.

Example 8.5.0.1 *It can be considered that the document under scrutiny has* 19 *occurrences of the rsid characters:*

```
<w:p w:rsidR="00463DF8" w:rsidRDefault="00463DF8" w:rsidP="00463DF8">
<w:r w:rsidRPr="0074047B">
<w:p w:rsidR="00463DF8" w:rsidRDefault="00463DF8" w:rsidP="00463DF8">
<w:r w:rsidRPr="008C3D74">
<w:r w:rsidRPr="0074047B">
<w:p w:rsidR="00463DF8" w:rsidRPr="008C3D74" w:rsidRDefault="00463DF8" w:rsidP="00463DF8">
<w:p w:rsidR="000E634E" w:rsidRPr="00463DF8" w:rsidRDefault="00463DF8">
<w:sectPr w:rsidR="000E634E" w:rsidRPr="00463DF8" w:rsidSect="009B2A88">
```

Thus, it has 152 *(*19×8*) characters to store information (see Table 8.1).*

Assuming that the message "this message is hidden in a word document" *(*41 *characters) is to be hidden, using a standard ASCII code. The first step is to replace every character of the message with the* 2 *characters that are the relative representation of the ASCII code (see Table 8.2).*

A sequence of 82 *characters is obtained, with a further* 70 *symbols "*0*" attached. Thus, a string of* 152 *symbols is obtained (see Table 8.3). Finally it will be enough to replace, in an XML file, the string of symbols in Table 8.3 to the values of the* rsid *attributes in order to complete the steganography process.*

t	*h*	*i*	*s*		*m*	*e*	*s*	*s*	*a*	*g*	*e*		*i*	*s*		*h*	*i*	*d*	*d*
74	68	69	73	20	6D	65	73	73	61	67	65	20	69	73	20	68	69	64	64
e	*n*		*i*	*n*		*a*		*w*	*o*	*r*	*d*		*d*	*o*	*c*	*u*	*m*	*e*	*n*
65	6E	20	69	6E	20	61	20	77	6F	72	64	20	64	6F	63	75	6D	65	6E
t																			
74																			

Table 8.2: Coded message.

74	68	69	73	20	6D	65	73	73	61	67	65	20	69	73	20	68	69	64	64
65	6E	20	69	6E	20	61	20	77	6F	72	64	20	64	6F	63	75	6D	65	6E
74	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00				

Table 8.3: Sequence of *rsid* values with hidden data.

```
<w:p w:rsidR="74686973" w:rsidRDefault="206D6573" w:rsidP="73616765">
<w:r w:rsidRPr="20697320">
<w:p w:rsidR="68696464" w:rsidRDefault="656E2069" w:rsidP="6E206120">
<w:r w:rsidRPr="776F7264">
<w:r w:rsidRPr="20646F63">
<w:p w:rsidR="756D656E" w:rsidRPr="74000000" w:rsidRDefault="00000000" w:rsidP="00000000">
<w:p w:rsidR="00000000" w:rsidRPr="00000000" w:rsidRDefault="00000000">
<w:sectPr w:rsidR="00000000" w:rsidRPr="00000000" w:rsidSect="00000000">
```

Obviously the message to be hidden would be preferably encrypted before embedding.

Algorithms 5 and 6 show how to encode and decode a secret message using the methodology analyzed in this Chapter.

8.6 Information Hiding by CustomXML feature

The feature *CustomXML* is the instrument of Microsoft that allows to embed data inside the files MS-Office [1]. It is possible using *Custom XML data properties* defined in standard ECMA-376 like the properties *"allow the ability to store arbitrary XML in a package, along with schema information used by that XML"*. An example of MS-Office document that contains Custom XML is shown in Fig. 8.1. As it can be observed, in the root folder of the container ZIP it is present the folder *customXml* and inside it are stored the files *test.xml* and *test.xml.rels*. The last one is contained

Algorithm 5 Sequence of steps to encode secret message using "rsid" values

X is the XML document which contains the sequences of rsid
S is the message to hide
H is the values of rsid
M is the maximum number of characters to be hidden in X

S = null
SA = null
go to the start of the XML document X
repeat
  A = null
  search in X next occurrence of "rsid"
  A = value of "rsid"
  H = append A
until eof(X)
L = number of characters in H
$M = \lfloor \frac{L}{2} \rfloor$
input S
LS=number of characters of S
if LS ¿ M **then**
  print "Too many characters to hide"
else
  $I = 1$
  repeat
    F = first character of hexadecimal representation of $S(I)$
    S = second character of hexadecimal representation of $S(I)$
    SA = append F
    SA = append S
    $I = I + 1$
  until $I \leq LS$
  go to the start of XML document X
  $I = 1$
  repeat
    search in X next occurrence of "rsid"
    if $I > LS$ **then**
      A = "00000000"
    else
      for $I = 1$ to 8 **do**
        $A = A||A[I]$
      end for
    end if
    "rsid" = A
    $I = I + 1$
  until eof(X)
end if

Algorithm 6 Sequence of steps to decode secret message from "rsid" values

X is the XML document which contains the sequences of rsid
S is the message to hide
H is the values of rsid
M is the maximum number of characters to be hidden in X

S = null
go to the start of XML document X
repeat
  A = null
  search in X next occurrence of "rsid"
  A = value of "rsid"
  H = append A
until eof(X)
L = number of characters in H
$I = 1$
repeat
  $K = H[I]||H[I+1]$
  B = representation of hexadecimal value K
  S = append B
  $I = I + 2$
until $I - 1 \leq L$

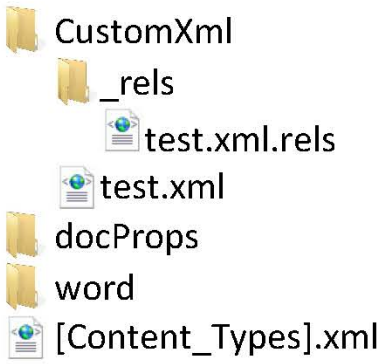

Figure 8.1: Document contains CustomXML

in the subfolder *_rels*. This structure, opportunely customized, can be used in order to contain a secret message.

Taking advantage of this property, in the thesis *"Data hiding and detection in Office Open XML (OOXML) documents"*, Raffay [4] proposes a new technique in order to hide information inside OOXML documents. The methodology is simple. In the first instance it is necessary to create the structure of a folder and subfolder illustrated in Fig. 8.1. Then, the file *test.xml* that contains the secret message (the file name can be also different but must have the extension xml) can be inserted. The text file is in XML format to looks legitimate *Custom XML data*. An example of file *test.xml*, that contains secret message "This is an example of secret message" is the following one:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:document xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships
 /customXml" xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/main">
<w:body>
 <w:p w:rsidR="00F72053" w:rsidRDefault="00A34A8F">
  <w:r>
   <w:t>This is an example of secret message</w:t>
  </w:r>
 </w:p>
</w:body>
```

So, the file containing the secret message are recognized like a content valid from MS-Office. It is now necessary to create just the one relationship. This is possible inserting, in folder *\customXml\rels* the file *test.xml.rels*. Such file will contain the

relationship that defines the file *test.xml* of type *custom XML Data*. An example of *test.xml.rels* is the following one:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
 <Relationship Id="rId100" Type="http://schemas.openxmlformats.org/officeDocument/2006/
  relationships/customXmlData" Target="/customXML/test.xml"/>
</Relationships>
```

At the end of these operations, it will only copy the folder *customXml* in the root of the MS-Office ZIP container. In such a way the secret message is hidden and the *Document Inspector* is not able to detect the presence of such contents. The only problem is that such methodology is vulnerable to the save operation. In fact the folder *customXml*, and consequently the hidden data, is removed as a result of the save action.

References

[1] Microsoft, "Custom XML Parts Overview". http://msdn.microsoft.com/en-us/library/bb608618.aspx, April 2011.

[2] A. Castiglione, B. D'Alessio, A. De Santis and F. Palmieri, "New Steganographic Techniques for the OOXML File Format". *Availability, Reliability and Security for Business, Enterprise and Health Information Systems*, pages 344-358, 2011.

[3] ECMA International, "Final draft standard ECMA-376 Office Open XML File Formats - Part 1". *ECMA International Publication*, December 2008.

[4] M. A. Raffay, "Data Hiding and Detection in Office Open XML (OOXML) Documents". http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.2099&rep=rep1&type=pdf, March 2011.

[5] B. Park, J. Park and Sangjin Lee, "Data concealment and detection in Microsoft Office 2007 files". *Digital Investigation*, vol.5(3-4), pages 104-114, 2009.

[6] Zhangjie Fu, Xingming Sun, Yuling Liu, Bo Li, "Forensic investigation of OOXML format documents". *Digital Investigation*, vol.8(1), pages 48-55, July 2011

Chapter 9

Methodologies Compared and Classified

9.1 Introduction

In this chapter all the described techniques are analyzed to classified them according to the Bauer tree, to verify if the *Document Inspector* can detect and remove the secret message, to analyze the resulting behavior of save actions and to calculate the introduced overhead. A common consideration is that the methodologies discussed in this Thesis can all be applied simultaneously to the same document. The amount of information that can therefore be hidden in the file will be greater than when using a single technique. Finally, in order to guarantee ulterior data confidentiality, before proceeding to the phase of embedding, all the data to be hidden should be encrypted using a symmetrical key algorithm.

9.2 Classification of Methodologies

In the preview chapter, the steganography methodologies applicable to MS-Office 2007/2010 documents are analysed. Starting from the idea that to manage Office Open XML documents is like to process files ZIP, XML and several other file formats, we have classified these techniques in the following five groups:

- Information Hiding by ZIP files;

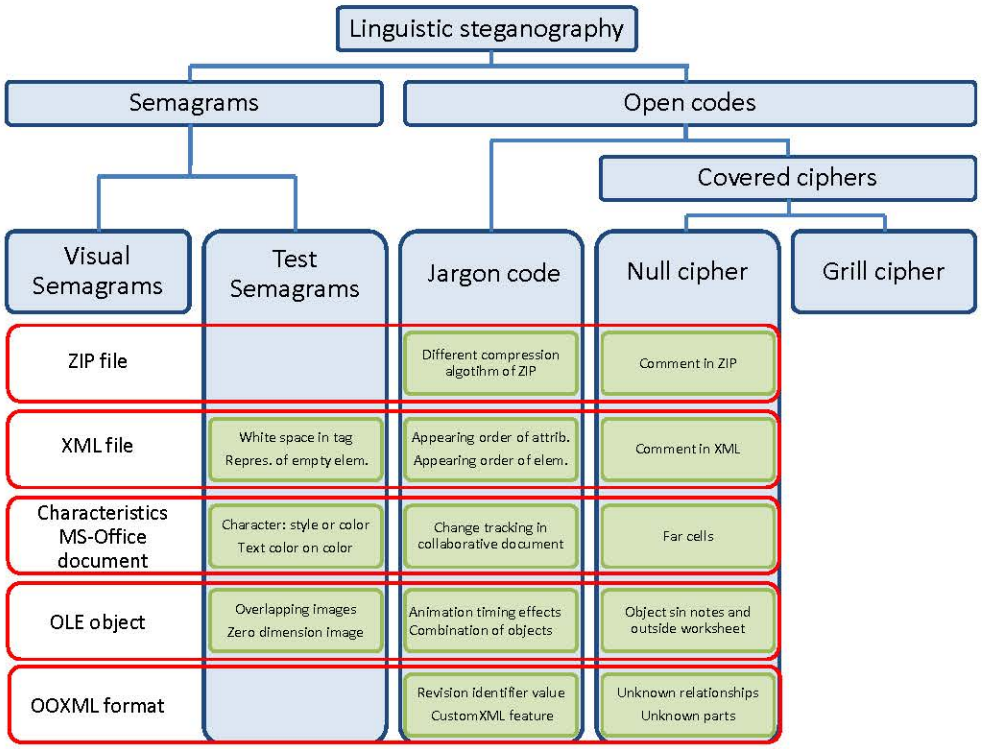

Figure 9.1: Classification of Information Hiding for MS-Office documents.

- Information Hiding by XML;
- Information Hiding by Characteristics and Formats of the MS-Office documents;
- Information Hiding by OLE object;
- Information Hiding by OOXML format.

Starting from these five groups and from the classification of the steganography methodologies proposed from Bauer, we will placed all the techniques in the right category. Figure 9.1 shows where, in our opinion, the different techniques can be collocate. The methodology of "use of macro" is not present because according to how they are implemented, it can fall into the categories *Test Semagram*, *Null Cipher* and *Jargon Code*.

The methodologies analyzed in this thesis have been classified as steganography techniques. In fact, the methodologies are not *Anonymity* techniques, because is not operated in way to hide the sender or the receiver. For this reason, if we suppose to track the path of the document and having availability of the several versions, it would be possible identify the user that inserted the hidden data.

They cannot be considered *Covert Channels* as they do not satisfy the definition of Lampson [2]. *Covert Channels* have been defined, by Lampson, in the context of multilevel secure systems, as communication paths that were neither designed nor intended to transfer information at all. While these channels are typically used by untrustworthy programs to leak information to their owner performing to service for another program.

At last, the methodologies are not classified as *Watermarking* because they do not satisfy the second and third attribute presented by Cox et al. [1]. They claimed that watermarking is distinguished from other techniques in three important ways. First, watermarks are imperceptible. Unlike bar codes, they cannot be detracted from the aesthetics of an image. Second, watermarks are inseparable from the "works" in which they are embedded. Unlike header fields, they are not removed when the "works" containing a watermark are displayed or converted to other file formats. Finally, watermarks undergo the same transformations as the Works. This means that it is sometimes possible to learn something about those transformations by looking at the resulting watermarks. It is these three attributes that make watermarking invaluable for certain applications. In fact, the hidden information are not always "*inseparable from the work*" (for example many do not resist to save operations or can be removed without damage the file) and not always "*undergo the same transformations*" of the "*container file*" (usually transformations produce the loss of the hidden data).

9.3 Resistance

The *Document Inspector*, as indicated in Chapter 2, is the tool supplied by Microsoft, which is used to search for and remove any eventual information hidden in MS-Office files. Thus, an Information Hiding methodology must pass the controls of this tool to be considered good. All the methodologies presented in this thesis resist the analysis of the *Document Inspector*. Only in two cases, Chapter 6.5 and 6.6, the *Document Inspector* give an users alert. These cannot be considered an anomalies because are find change tracking and macro, not data concealment, so also these methodologies resist to the analysis of the *Document Inspector*. In addition to controlling and removing hidden information with the *Document Inspector*, MS-Office also carries out a type of optimization and normalization in the ZIP container every time the file is saved. These operations consist of removing everything that it is not recognized as valid for the application (e.g. files attached without a link) as well as reorganizing the elements that make up the XML code according to its own outline. These particular aspects render the techniques presented in Chapters 4.2, 4.3, 5.2, 5.3, 5.4, 5.5, 5.6, 6.3, 8.2, 8.3 and 8.5 vulnerable. In fact, as a result of a save action, MS-Office compresses all the files inside the ZIP container using the default algorithm (DeflateS), removes ZIP "comments" and assigns new values to the *rsid* attributes. In addition, it merges together consecutive elements and parts, of XML code, that have same style proprieties and removes "comment", "empty element", "white space in tag", "unknown parts" and "unknown relationships" from XML file. Therefore, in order to avoid that the hidden information be removed as a result of an "involuntary" save action (e.g. automatic saving), it is worthwhile marking the document as the "final version". The user is therefore dissuaded from making any modifications unless specifically authorized. The methodologies analyzed in Chapter 6.3, when uses the property "color of character" are not vulnerable to save action. In fact, it is possible to modify or to

insert XML elements that will not be removed after the save action. These operations are possible only in part of code without secret message.

9.4 Overhead Data

It is impossible to make any general consideration, about the overhead introduced by the hiding methods presented in this thesis. Then, we analyzed the single methodologies. The methodologies shown in Chapter 5.5, 5.6 did not introduce an overhead because nothing is added (characters, elements, files, etc.). On the other hand, the methodology presented in Chapter 8.5 has a null overhead, in the event in which the text to be hidden is less than the maximum number of bits that can be contained in the document, with it being a function of the parts inserted in the XML files, in the other cases. In Chapter 4.2 the introduced overhead is very small and it is only equal to the length of the hidden message. The methodologies analyzed in Chapter 6.5, 6.6, 7.2 and 8.6 introduce an overhead function of the changes applied (to document and OLE object) or elements inserted (macro or CustomXML). It depends on the types and quantities of changes applied to the document, to macros introduced, the techniques of steganography applied as well as CustomXML files inserted. In the event discussed in Chapter 4.3, the overhead is a function of the compression ratio applied for the different algorithms. Therefore, the dimension of the file can either increase, remain unchanged or diminish. The overhead introduced by the solution proposed in Chapter 7.3, 7.6 and 7.7 is a function of two values. These values are the dimension of the attached file, that contains the hidden data, plus the dimension of the elements added in the XML files and required in order to insert the OLE object with the needed characteristics. Also using the technique analyzed in Chapter 8.2 the overhead is a function of two values: the dimension of the unknown part plus the element added in XML code to declare a trusted unknown part (`Override PartName="/app.txt"`

- 51 characters). The methodologies analyzed in Chapter 6.2, 6.3, 6.4, 7.4 and 7.5 introduce an overhead that is equal to the dimension of the elements added in the XML files and needed to apply the specific property or functionality. The overhead introduced by the technique proposed in Chapter 8.3 is the number of characters which compose the added unknown relationship (60 characters plus the numbers of characters of hidden text and plus the length of attribute "Target"). While applying the methodology of Chapter 8.4, the overhead introduces is sum of the one introduced in Chapter 8.2 plus the one introduced in Chapter 8.3. The overhead introduced in the methodologies analyzed in Chapter 5.2, 5.3, and 5.4 is function of numbers of characters that compose the hidden text plus the numbers of characters that are introduced in the XML code. If L is the numbers of characters that compose the hidden text:

- in Chapter 5.2 the overhead is L plus 7 characters (the XML comment element "<!--->"), for every line of comment in which the hiding text will be decomposed;
- in Chapter 5.3, if it suppose to use the element <w:t>, the overhead is 11 characters (length of "<w:t></w:t>") for every "empty element" insert.
- in Chapter 5.4 the overhead is 1 characters for every space introduced.

In the last two cases, the overhead is function of the number of the occurrence of the characters (0 and 1). If 1 occurs n times and 0 occurs k the overhead introduced is n or $n \cdot 11$, when it associates the empty space to bit 1 and k or $k \cdot 11$, otherwise. If it supposes that $n > k$ ($n \cdot 11 > k \cdot 11$) it can be used an algorithm that associates the "white space", or "empty element", to the bit that occurs the minor times in the secret message, so it have the lower overhead. The Table 9.1 summarize the different methodologies, of information hiding, regarding the following parameters:

- *Property*: the property of the MS-Office document taken advantage of from the methodology;
- *D.I.*: it indicates if the methodology is detected by the *Document Inspector*;
- *Save*: it indicates if the inserted data remain in the document after a save action;
- *Max over*: it indicates the maximum overhead (maximum number of characters) inserted in the document.

Methodology	Property	D.I.	Save	Max over
Comment in ZIP file	ZIP file	YES	NO	***L***
Different compression algorithm of ZIP	ZIP file	YES	NO	***X***
Comment in XML	XML file	YES	NO	***L+7*R***
Representation of empty elements	XML file	YES	NO	***11*L***
White space in tag	XML file	YES	NO	***L***
Appearing order of elements	XML file	YES	NO	***0***
Appearing order of attributes	XML file	YES	NO	***0***
Text color on color	MS-Office	YES	YES	***E***
Style of a character	MS-Office	YES	NO	***E***
Color of a character	MS-Office	YES	YES	***E***
Far cells	MS-Office	YES	YES	***E***
Change tracking in collaborative document	MS-Office	YES	YES	***X***
Office macro	MS-Office	YES	YES	***X***
Steganography in the attached and related file	OLE object	YES	YES	***X***
Objects in notes and outside worksheet	OLE object	YES	YES	***F+C+E***
Animation Timing effects	OLE object	YES	YES	***E***
Combination of objects	OLE object	YES	YES	***E***
Zero dimension image	OLE object	YES	YES	***F+C+E***
Overlapping two images	OLE object	YES	YES	***F+C+E***
Unknown parts	OOXML	YES	NO	***F***
Unknown relationships	OOXML	YES	YES	*C*
Unknown parts and unknown relationships	OOXML	YES	YES	***F+C***
Revision identifier value	OOXML	YES	NO	***0*** or ***X***
CustomXML feature	OOXML	YES	YES	***X***

X: Variable and not estimable with a general function

L: Number of characters who compose the text to hide

R: Number of inserted lines of comment, in which the text to hide is decomposed

F: Dimension of the file added

E: Dimension of the elements added in XML file

C: Number of characters who compose the introduced relation

Table 9.1: Comparison methodologies.

References

[1] I.J. Cox, M.L. Miller, J.A. Bloom, J. Fridrich and T. Kalker, "Digital Watermarking and Steganography. Second Edition". *Morgan Kaufmann Publishers is an imprint of Elsevier*, page 15, 2008.

[2] B.W. Lampson, "A note on the confinement problem". *Commun. ACM*, vol.16(10):pages 613-615, October 1973.

Chapter 10

Experiments

10.1 Introduction

In order to verify the real amount of information that can be hidden by using the proposed techniques, several tests have been carried out on a set of 53,841 files composed of Word, Excel and PowerPoint documents (see Fig. 10.1). The files have been downloaded from the Internet by different domains, as shown in Fig. 10.2. The amount of files analyzed in this chapter, classified by type and domain, is shown in Tab. 10.1. All the researches, downloads and analyses are performed using scripts that we have created for the specific purpose.

10.2 Files Collection

MS-Office documents need for the experiments (files with extension *.docx* , *.xlsx* e *.pptx*) are downloaded from Internet using Google.

The Google search engine accepts as input a search requests. Usually, people use HTML forms displayed in a web browser to make the search requests, but other applications can also send search requests by making appropriate HTTP requests. The search request is a URL composed of: the host name (www.google.it) or ip address, search interface port (usually 80) and a path describing the search query.

Domain	docx	pptx	xlsx	Total
ae	884	669	442	**2,995**
at	849	685	484	**2,018**
au	932	807	749	**2,488**
ca	888	939	746	**2,573**
ch	881	813	524	**2,218**
com	886	900	810	**2,596**
de	907	862	666	**2,435**
edu	969	973	913	**2,855**
es	926	879	404	**2,209**
fr	883	852	406	**2,141**
in	752	724	351	**2,827**
int	797	466	445	**2,708**
ir	903	749	394	**2,046**
it	915	845	453	**2,213**
jp	842	851	823	**2,516**
kr	844	846	676	**2,366**
net	925	745	842	**2,512**
nl	914	892	553	**2,359**
org	902	851	909	**2,662**
ru	937	841	861	**2,639**
se	935	794	665	**2,394**
th	888	757	900	**2,545**
uk	877	940	709	**2,526**
Total	**20,436**	**28,680**	**24,725**	**53.841**

Table 10.1: Files per domain.

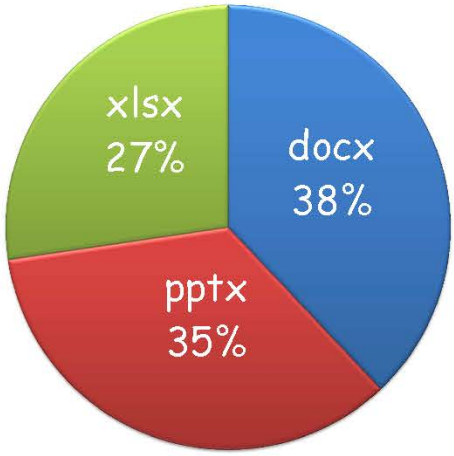

Figure 10.1: Distribution of the test files used for the experiments.

In our experiment we have used the HTTP requests and we have set the parameters to restrict the research to specific filetype and domain. The script to research the Word's files is:

```
#!/bin/bash
domain=("ae" "at" "au" "ca" "ch" "com" "de" "edu" "es" "fr" "in" "int" "ir" "it" "jp" "kr"
        "net" "nl" "org" "ru" "se" "th" "uk")
maxido=${#domain[@]}
maxlink=900
ido=0

# Part_1: Delete all files and folders named "link"
rm -r -f log_crealinks
while (( "ido" <= $maxido-1 )); do
   dirdownload=query_
   ap=${domain[$ido]}
   dirdownload=\.\/$dirdownload$ap
   rm -f -r $dirdownload
   link=link_
   link=\.\/$link$ap
   rm -f $link
   let ido=ido+1
done

# Part_2: Download Google research pages and create file "link"
ido=0
while (( "ido" <= $maxido-1 )); do
   dirdownload=query_
   ap=${domain[$ido]}
   dirdownload=\.\/$dirdownload$ap
   page0=http://www.google.com/
   pagea=search\?q=+site:.
   pageb=+filetype:docx\&num=100\&hl=it\&lr=\&as_qdr=all\&start=
   pagec=\&sa=N\&filter=0
   num=0
   link=link_
   link=\.\/$link$ap
```

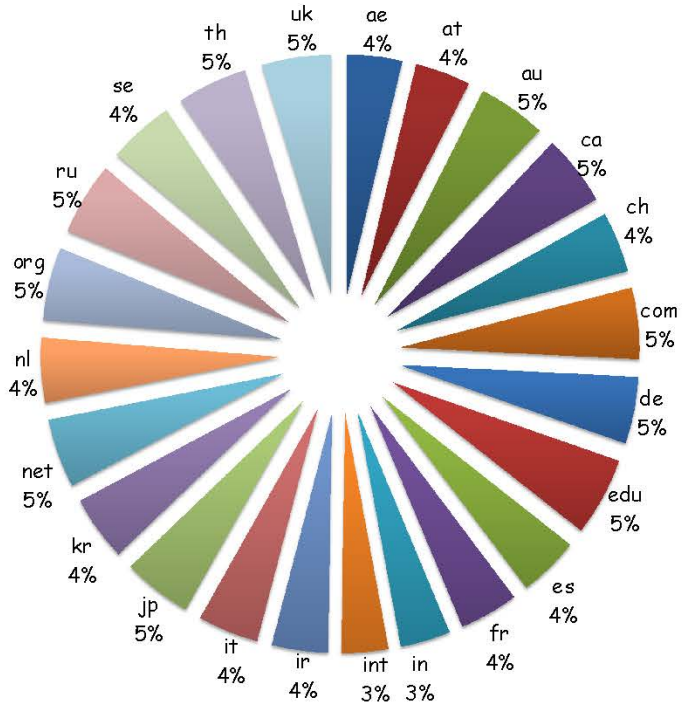

Figure 10.2: Test files distribution based on the domain of download.

```
   mkdir  $dirdownload
   nome='/appo'
   while (( "num" <= $maxlink )); do
      page=$page0$pagea$ap$pageb$num$pagec Initializes the ta
      wget -U  'Mozilla/5.0 (X11; U; Linux i686; it; rv:1.9.1.15) Gecko/20101027 '
           $page -O$dirdownload$nome  --append-output=log_crealinks
      sleep 1m
      nuovonome=$dirdownload\/$num\.txt
      iconv --from-code=ISO-8859-1 --to-code=ISO-8859-1 $dirdownload$nome > $nuovonome
      sed 's/<a href=\"/\n/g' $nuovonome  | grep '\.docx\"' | sed 's/\.docx\".*/\.docx/g' |
           sed 's/<a href=\"//g'>> $link
      rm -f $dirdownload$nome
      let num=num+100
   done
   let ido=ido+1
done
ido=0

# Part_3: Delete all temporary files and folders
while (( "ido" <= $maxido-1 )); do
   dirdownload=query_
   ap=${domain[$ido]}
   dirdownload=\.\/$dirdownload$ap
   rm -f -r $dirdownload
   let ido=ido+1
done
```

At the start, the script initializes the target variables where:

- *domain*, array that store top-level domain (TLD) name where we research the documents;
- *maxido*, number of elements of *domain*;
- *maxlink*, max number of files that we will download for every domain (in this case 900).

Then, the script is divided into three parts:

1. `Part_1`, delete every temporary directory (*query_ae* ... *query_uk*) and every files that will stored link (*link_ae* ... *link_uk*);
2. `Part_2`, performs a parameterized Google search, restricted to 100 results for every time, using the follow URL:

   ```
   http://www.google.com/search\?q=+site:XX+filetype:docx\&num=100\&hl=it\&lr=\&as_qdr=all
   \&start=YY\&sa=N\&filter=0\\
   ```

   where XX is the top-level domain name and YY is a code that indicate the number of page result. Using `wget command`, the search results are stored in a temporary file (e.g. *query_ae/appo*). From this file, using the regular expression:

   ```
   sed 's/<a href=\"/\n/g' $nuovonome  | grep '\.docx\"' | sed 's/\.docx\".*/\.docx/g' |
        sed 's/<a href=\"//g'>> $link
   ```

   only the link will be extract, that looks like:

   ```
   http://www.adaep.ae/AR/MediaCenter/Publication/FAQ_General_AR.docx
   ```

   and stored in corrisponding file (e.g. *link_ae*).
3. `Part_3`, delete every temporary directory (*query_ae* ... *query_uk*).

The text files *link_ae* ... *link_uk* contain URL which connects to download the searched files. The download will be done using the script:

```
#!/bin/bash
domain=("ae" "at" "au" "ca" "ch" "com" "de" "edu" "es" "fr" "in" "int" "ir" "it" "jp" "kr"
        "net" "nl" "org" "ru" "se" "th" "uk")
maxido=${#domain[@]}
ido=0

# Part_1: Delete all files and folders di appoggio
rm -f -r log_scarica_file_docx
while (( "ido" <= $maxido-1 )); do
```

```
   dirdownload=docx_
   ap=${domain[$ido]}
   dirdownload=\.\/$dirdownload$ap
   rm -f -r $dirdownload
   rm -f $dirdownload\/assoc_docx
   let ido=ido+1
done

# Part_2: download files
ido=0
while (( "ido" <= $maxido-1 )); do
   dirdownload=docx_
   ap=${domain[$ido]}
   dirdownload=\.\/$dirdownload$ap
   link=link_
   link=\.\/$link$ap
   nome=\/docx\_$ap\_
   mkdir $dirdownload
   prog=1
   for linkcat in `cat $link 2>/dev/null`
   do
     wget -U 'Mozilla/5.0 (X11; U; Linux i686; it; rv:1.9.1.15) Gecko/20101027 ' $linkcat
          -O$dirdownload$nome$prog\.docx --dns-timeout=60 --connect-timeout=60
          --read-timeout=1200 --append-output=log_scarica_file_docx
     echo $linkcat','$nome$prog\.docx >> $dirdownload\/assoc_docx
     let prog=prog+1
   done
   let ido=ido+1
done
```

The target variables initialization and the `Part_1` of the download script have the same behavior of the search script. In the `Part_2`, for every domain XX:

- creates a new directory, named "*_XX*";
- downloads documents finds at the URL stored in the "*link_XX*" files;
- the documents are stored in the directory "*docx_XX*" and named "*docx_XX_*" with a progressive number and extension "*.docx*";
- the associated file name assigned to the document and to the URL where it was downloaded from and are stored in the file "*assoc_docx*".

The same scripts, customized, are used to search and download files PowerPoint ed Excel. We only substitute the occurence of *.docx* with *.pptx* or *.xlsx*. We have downloaded about 41 GB of documents.

10.3 How and What is under analysis

For the experiments we have not been considered all the presented methodologies because in some cases it is not possible, starting from the properties of MS-Office files, to establish the amount of information that can be hidden. In fact, when using the *Unknown Parts* or *Unknown Parts and Unknown Relationships* or *Zero Dimension Image* methodologies, the amount of secret information is a function of the size of the added "image" as well as the type of steganography that is applied to it. Furthermore, the amount of information that can be concealed with the *Office Macro* methodology depends on the type and quantity of the added instructions in the macro. When it applying the *Change Tracking in Collaborative Document* the length of hidden message depends on the chosen database of degeneration. And so on.

As a consequence, the experiments have been conducted only on the:

1. *different compression algorithm of ZIP*;
2. *representation of empty elements*;
3. *white space in tag*;
4. *appearing order of elements*;
5. *appearing order of attributes in element*;
6. *value of revision identifier.*

Focus of this study is to identify the steganographic methodologies that can be applied to the MS-Office documents and not the implementation of the same ones. Therefore, although they are various *XML elements* that can be used in order to apply the techniques described in this work, for the this experimentation only some have been chosen. Follow are indicated *XML elements* chosen in order to estimate the amount of information that can be hidden:

- *representation of empty elements*, are searched the *XML elements* `<w:t>` and `</w:t>` in the document.xml (Word), `</row>` in the files sheet1.xml ... sheetn.xml (Excel), while are not individuate any *XML elements* to apply such technique for the PowerPoint files;

- *white space in tag*, are searched the *XML elements* `<w:t>` and `</w:t>` in the document.xml (Word), `<a:t>` and `</a:t>` in the files slide1.xml ... sliden.xml (PowerPoint) and `<v>` and `</v>` in the files sheet1.xml ... sheetn.xml (Excel);

- *appearing order of elements*, are searched the *XML elements* `<w:autoSpaceDE w:val="0"/>` and `<w:autoSpaceDN w:val="0"/>` in the file document.xml (Word), `<a:lumMod val="20000"/>` and `<a:lumOff val="80000"/>` in the files slide1.xml ... sliden.xml (PowerPoint), while are not individuate any *XML elements* to apply such technique for the Excel files;

- *appearing order of attributes in element*, are searched the *XML elements* `rFonts w:ascii` in the document.xml (Word), `<a:rPr lang="en-US" dirty="0" smtClean="0">` in the slide1.xml ... sliden.xml (PowerPoint) and `<c r=" " s=" " t=" ">` in the files sheet1.xml ... sheetn.xml (Excel).

We have analysed all the downloaded files using the following script:

```
#!/bin/bash
domain=("ae" "at" "au" "ca" "ch" "com" "de" "edu" "es" "fr" "in" "int" "ir" "it" "jp" "kr"
        "net" "nl" "org" "ru" "se" "th" "uk")
maxido=${#domain[@]}
ido=0
rm -f ./statistics_docx.csv
while (( "ido" <= $maxido-1 )); do
   dirdownload=docx_
   ap=${domain[$ido]}
   dirdownload=\.\/$dirdownload$ap
   link=link_
   link=./$link$ap
```

```
   rm -f -r ./tmp
   rm -f EDN
   rm -f Erfonts
   rm -f Ewts
   rm -f ErsidRD
   rm -f ErsidR
   rm -f ErsidRPr
   rm -f ErsidRP
   rm -f Ersid
   rm -f elenfile
   rm -f line
   rm -f media
   for file in `ls $dirdownload 2>/dev/null`
   do
# Variable assignment
      file=$dirdownload/$file
      filecont=./tmp/word/document.xml
# Container information
# Count and list files present in ZIP container and store their names in elenfile
      unzip -l $file | cut -c29- > elenfile
      sed -i '1,3d' elenfile
      sed -i '$d' elenfile
      sed -i '$d' elenfile
      lineselenfile=$(wc -l elenfile | cut -f1 -d' ')
      # Uncompress the file stores in direcotry ./tmp
      unzip -o $file -d ./tmp
      size=$(du -bs ./tmp| cut -f1 -d'.')
      sizec=$(du -b $file| cut -f1 -d'.')
      cat elenfile | grep -oE 'word/media/*' > media
      linesmedia=$(wc -l media | cut -f1 -d' ')
      if [ "$linesmedia" -ne "0" ]; then
            sizemedia=$(du -bs ./tmp/word/media| cut -f1 -d'.')
      else
            sizemedia=0
      fi
# Counts the occurence of 'DN', 'rfonts', 'wts', 'rsid' (rsidRD+rsidR+rsidRPr+rsidRP)
      sizesfilecont=$(du -b $filecont| cut -f1 -d'.')
      cat $filecont | grep -oE '<w:autoSpaceDN.....' > EDN
      cat $filecont | grep -oE '<w:rFonts w:ascii.....' > Erfonts
      cat $filecont | grep -oE '/w:t' > Ewts
      cat $filecont | grep -oE 'rsidRDefault=".....' > ErsidRD
      cat $filecont | grep -oE 'w:rsidR=".....' > ErsidR
      cat $filecont | grep -oE 'w:rsidRPr=".....' > ErsidRPr
      cat $filecont | grep -oE 'w:rsidP=".....' > ErsidRP
      cp ErsidRD Ersid
      cat ErsidR >> Ersid
      cat ErsidRPr >> Ersid
      cat ErsidRP >> Ersid
      linesEDN=$(wc -l EDN | cut -f1 -d' ')
      linesErfonts=$(wc -l Erfonts | cut -f1 -d' ')
      linesEwts=$(wc -l Ewts | cut -f1 -d' ')
      linesErsid=$(wc -l Ersid | cut -f1 -d' ')
# Append to 'statistics.csv' a new row with the fields: domain, filename, tot files,
  dim. extracted files, dim. compressed file,tot files OLE, dim. files OLE, dim.
  document.xml, occurrences 'DN', occurrences 'rfonfs', occurrences '\w:t', occurrences 'rsid'
      line=$ap','$file','$lineselenfile','$size','$sizec','$linesmedia','$sizemedia',
           '$sizesfilecont','$linesEDN','$linesErfonts','$linesEwts','$linesErsid
      echo $line > line
      cat line >> statistics_docx.csv
      rm -f -r ./tmp
      rm -f EDN
      rm -f Erfonts
      rm -f Ewts
      rm -f ErsidRD
```

```
        rm -f ErsidR
        rm -f ErsidRPr
        rm -f ErsidRP
        rm -f Ersid
        rm -f elenfile
        rm -f line
        rm -f media
    done
    let ido=ido+1
done
```

The script produces a comma separated value (*.csv*) file that contain, for every downloaded file, information on:

- main domain;
- file name;
- file dimension;
- number of files stored in ZIP container;
- dimension of extracte files;
- number of OLE files;
- dimension of OLE files;
- dimension of document.xml;
- number of "DN" occurrence;
- number of "rfonfs" occurrence;
- number of "w:t" occurrence;
- number of "rsid" occurrence.

The same scripts, customized, are used to search and download files PowerPoint ed Excel. The scripts have produced three files *statistics_docx.csv*, *statistics_pptx.csv* and *statistics_xlsx.csv*. Then we merge all these data in a summary file named *summary.xlsx*. All the files (*statistics_docx.csv*, *statistics_pptx.csv*, *statistics_xlsx.csv* and *summary.xlsx*) are not listed in this Thesis because they are too long.

	Word			PowerPoint			Excel		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
Number of files	20,436	20,436	20,436	18,680	18,680	18,680	14,725	14,725	14,725
Dimension of ZIP file	287,134	4,035	16,607,100	2,320,961	24,983	59,011,334	120,874	6,333	27,721,494
Dimension of extracted files	777,234	12,560	287,817,186	4,009,739	79,100	287,802,760	943,787	15,048	342,234,161
% compression	38,11%	0,91%	99,71%	58,75%	0,95%	99,65%	20,27%	2,18%	99,51%
Dimension of XML file	207,256	0	85,930,158	311,197	0	25,545,191	365,378	666	170,994,712
Number OLE obj	3	0	1,806	21	0	501	1	0	1,428
Dimension OLE obj	441,867	0	287,058,370	2,787,268	0	287,359,019	44,238	0	13,776,231
Number of *XML elements* in *appearing order of element*	16	0	10,779	26	0	3,444	0	0	0
Number of *XML elements* in *appearing order of attributes in element*	393	0	259,898	336	0	24,975	9,393	0	4,541,401
Number of *XML elements* used in *white space in tag*	906	0	544,696	456	0	31,731	6,764	0	4,541,401
Number of *XML elements* used in *empty element*	906	0	544,696	456	0	31,731	800	0	1,048,576
Number of *rsid*attributes	1,497	0	586,594	0	0	0	0	0	0
Number of files in ZIP	20	8	1,839	146	18	2,387	21	9	2,686

Table 10.2: Statistics on the data set.

10.4 Experimental Results

We have analized all the collection data, stored in *summary.xlsx*, and evaluated the various methodologies. The result of such operations and the properties of the considered documents are shown in Table 10.2.

As shown in the previous chapters, the amount of bits that can be hidden by using the various methodologies is obtained by multiplying the number of occurrences of *XML elements* for the amount of bits that every element can be contain (see Tab. 10.3).

	Word			PowerPoint			Excel		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
Data Hiding using Appearing Order of Element	2	0	1,347	3	0	431	0	0	0
Data Hiding using Appearing Order of Attributes in Element	127	0	83,978	109	0	8,070	3,035	0	1,467,419
Data Hiding using White Space in Tag	226	0	136,174	114	0	7,933	1,691	0	1,135,350
Data Hiding using Representation of Empty Element	113	0	68,087	0	0	0	100	0	131,072
Data Hiding by the Revision Identifier Value	5,990	0	2,346,376	0	0	0	0	0	0
Data Hiding by Different Compression Algorithm of ZIP	6	2	533	42	5	692	6	3	779
Applying all the methods	6,465	2	2,636,496	268	5	17,125	4,832	3	2,734,620

Table 10.3: Amount of bytes that can be hidden using the different methodologies.

At the end, the percentage of data concealment (%) is calculated by dividing the amount of information hidden by the dimension of the ZIP container (MS-Office file). The results are shown in Tab. 10.4.

The experiments shows that the methodology *different compression algorithm of ZIP* and *appearing order of element* bring a next contribution to the zero (usually a few bytes) while *value of revision identifier* is the most advantageous (but may be applied only to the Word). Thus, the largest amount of hidden data, for every type of MS-Office files, can be performed by applying all the techniques together.

	Word			PowerPoint			Excel		
	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max
Data Hiding using Appearing Order of Element	0,00%	0,00%	0,39%	0,00%	0,00%	0,21%	0,00%	0,00%	0,00%
Data Hiding using Appearing Order of Attributes in Element	0,16%	0,00%	4,56%	0,02%	0,00%	0,77%	1,38%	0,00%	12,84%
Data Hiding using White Space in Tag	0,25%	0,00%	6,48%	0,02%	0,00%	2,04%	0,63%	0,00%	7,21%
Data Hiding using Representation of Empty Element	0,12%	0,00%	3,24%	0,00%	0,00%	0,00%	0,06%	0,00%	2,46%
Data Hiding by the Revision Identifier Value	6,74%	0,00%	355,21%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
Data Hiding by Different Compression Algorithm of ZIP	0,01%	0,00%	0,08%	0,01%	0,00%	0,05%	0,02%	0,00%	0,08%
Applying all the methods	2,25%	0,06%	15,88%	0,01%	0,02%	0,03%	4,00%	0,04%	9,86%

Table 10.4: Percentage of data that can be hidden using the different methodologies.

Chapter 11

Conclusions and Further Research

The MS-Office 2007/2010 files, although conform to the OOXML standard, are not immune to the possibility to conceal data. In this work, we have presented different steganographic techniques that can be applied to these files. We have shown that it is possible to take advantage of some features of the new format adopted by Microsoft in order to store information that they must be held in secure and inaccessible way. We have verified that the amount of data that can be hidden is variable, with it being a few Bytes (useful for storing a PIN) or a few MByte (useful for storing larger and more complex messages like images, audio, etc.).

Unfortunately, such a peculiarity could be taken advantage also for illicit purposes. It is well-know that steganographic techniques are used also by the criminals (terrorists, criminal organizations, pedophiles, etc.). With these methodologies, suspects can communicate in secure way, eluding the controls of the police forces. Therefore, to be aware that is possible to hide information also in the new format of the MS-Office documents can help the investigators in the search of the digital evidence.

The current ICT development is driving market and users to the Cloud and Ubiquitous Computing paradigm. Today, there are so widespread applications that allow people to work on digital documents everywhere and everytime, allowing them to interact with digital documents by means of various digital equipment (e.g., smartphones, tablets, SmartTV). In fact, Microsoft and Google are moving their business

on the Cloud by introducing their upcoming webapps (i.e., Office 365 and Google Drive). This makes it more and more difficult for a digital investigator to conduct investigations when dealing with electronic documents. For this reason he has to broaden its point of view and expand its analysis also on the Web. Digital investigators, besides to be always updated on the latest technologies in the Digital Forensics Science, should also deeply focus on the research and development advances coming from different side of the ICT world, sometimes also from those ones that appear not so critical, such as "a simple electronic document". Future works will be oriented in the search of methodologies to characterize, to reveal and to remove the presence of concealed data in these kinds of files.

Author

Bonaventura D'Alessio received a degree in Computer Science from the University of Salerno, Italy, in 1994 and a Master's degree in Intelligence and Security from the University of Malta in 2004.

From December 1997 to October 2000 he has been technical civil servant of the italian Minister of Justice.

Since 2000 he is an Officer in Permanent Service, with a Technical Role in computer science, of the italian Carabinieri Force. He currently holds the rank of Major.

From November 2000 to August 2001 he attended the Formation Course for Officers of the Technical-Logistic Role at the Carabinieri Officers College in Rome.

From September 2001 to March 2006 he was assistant chief, of the high-tech unit involved in systems and networks security management, at the Carabinieri General Headquarter. From April 2006 to May 2010 he was the chief of the unit involved in the management of S.I.T.A. project (Information System for Environmental Conservation) of Carabinieri Environmental Care. He currently is the chief of the unit involved in Telecommunication, Information Technology, Weapons and Special Equipments, at the Carabinieri Specialist Mobile Unit Command.

His research interests include Communication Networks, Steganography, Data Security, Privacy and Security, Digital Forensics and Investigation.