Hiding Information into OOXML Documents: New Steganographic Perspectives*

Aniello Castiglione†
Dipartimento di Informatica
University of Salerno
Salerno, Italy
castiglione@acm.org

Bonaventura D'Alessio
Dipartimento di Informatica
University of Salerno
Salerno, Italy
bdalessio@dia.unisa.it

Alfredo De Santis
Dipartimento di Informatica
University of Salerno
Salerno, Italy
ads@dia.unisa.it

Francesco Palmieri
Dipartimento di Ingegneria dell'Informazione
Second University of Naples
Aversa (CE), Italy
francesco.palmieri@unina.it

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. The "Microsoft Compound Document File Format" (MCDFF) has often been used to host secret information. The new format created by Microsoft, first used with MS-Office 2007, makes use of a new standard, the "Office Open XML Formats" (OOXML). The benefits include that the new format introduces the OOXML format, which lowers the risk of information leakage, as well as the use of MS-Office files as containers for steganography.

In this work the authors, starting from the classification of information hiding adapted from Bauer, analyze four new methods for embedding data into the OOXML file format. These methods can be extremely useful when using MS-Office documents for steganographic purposes. The authors, analyzing a scenario composed of about 50.000 MS-Office files, highlight how the proposed methods are really helpful in real applications. An evaluation of the limits of the proposed methods is carried out by comparing them against the tool introduced by Microsoft to sanitize MS-Office files. The methods presented can be combined in order to extend the amount of data to be hidden in a single cover file.

Keywords: Steganography, OOXML format, stegosystem, document steganography, microsoft office document, information hiding.

1 Introduction

The MS-Office suite is probably the most widely used word-processing tool when preparing and writing documents, spreadsheets and presentations [2]. Therefore, the possibility to hide information inside them is a challenge that probably interests many different parties. Starting with 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. In fact, it has gone from

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 2, number: 4, pp. 59-83

*This paper is an extended version of the work originally presented at the 6th International Conference on Availability, Reliability and Security (ARES'11), Vienna, Austria, August 2011 [1]

†Corresponding author: Aniello Castiglione, ✉ Dipartimento di Informatica - Università degli Studi di Salerno, Via Ponte don Melillo, I-84084 Fisciano (SA), Italy. ☎: +39089969594, 🖷: +39089969821, 🖃: castiglione@ieee.org, castiglione@acm.org

using the old binary format to the new OOXML [3], which uses XML files. In addition, in order to guarantee a significantly high level of "privacy and security", it has also presented the feature *Document Inspector*, which makes it possible to quickly identify and remove any sensitive, hidden and personal information. It is therefore evident that the old methodologies of Information Hiding that exploit the characteristics of the binary files of MS-Office are no longer applicable to the new XML structures. However, the steganography techniques that take advantage of the functions offered by the Microsoft suite ([4], [5], [6], [7]) are still valid and therefore do not depend on the version used. The new format offers new perspectives, as proposed by Garfinkel et al. [8] and by Park et al. [9]. 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. This study proposes and analyzes four new steganography techniques for MS-Office files, with only the first not taking advantage of characteristics that do not conform to the OOXML standard.

The remaining of this paper is structured as follows. Section 2 introduces the OOXML standard and the features of the *Document Inspector*. Section 3 introduces the terminology and classification that will help to better understand the following of the paper. Section 4 discusses the methodology that takes advantage of the possibility to use different compression algorithms in generating MS-Office files. Section 5 illustrates how the macro of MS-Office can be used to hide information. In Section 6 a methodology that uses images not visualized by MS-Office, but present in the file, is analyzed in order to contain hidden information. Section 7 highlights how it is possible to hide data in the values of the attribute that specifies a unique identifier used to track the editing session (by using the revision identifiers). In Section 8 the methodologies are compared in order to verify the overhead introduced by them as well as to analyzed the behavior of "save" operation. Finally, in Section 9 the results of test are presented.

2 The OOXML Format

Starting with the 2007 version, Microsoft has adopted the OOXML format based on XML. In fact, Microsoft started the transition from the old binary file format to a new one that uses XML files. 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 and markers. Therefore, in XML a document is described, in form and content, by a sequence of elements. Every 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 MS-Office file just by using standard transformation methods. This is possible because XML text is clearly written and therefore visible and modifiable with any text editor. Moreover, OLE attachments are referenced in the source file and therefore can be visualized with any compatible viewer.

Distinguishing documents produced in this new format is easy due to the file extensions being characterized by an "*x*" at the end, with the Word, Excel and PowerPoint files respectively being *.docx*, *.xlsx*, and*.pptx*. An additional feature is that macros 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*, and*.pptm*. The new structure of an OOXML file, which is based on the ECMA-376 standard [10], uses a a ZIP file container, which contains a set of files, mostly XML properly organized into folders describing the content, the properties and the relationships. It is highly likely that the ZIP standard was chosen because it is one of the most widely adopted on the Internet. In addition, it have the characteristic of flexibility and modularity which allows for any eventual expansions in future functionalities [11].

There are three types of files stored in the "container", that can be common to all the applications of MS-Office or specific for each one:

- XML files, which describe application data, metadata, and even 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;
- relationship parts that specify the relationships between the parts; this design provides the structure for an MS-Office file.

For example, analyzing a simple Word document, the structure [12] of the folders and files will be like that shown in Fig. 1.

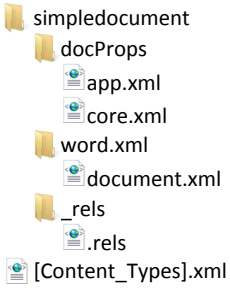

Figure 1: Structure of a simple Word document.

Therefore, beginning from version 2007, the MS-Office documents are 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, properly customized. Another key concept related to the OOXML format is the modularity, which allows for either the easy addition of new elements or 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 markup that defines the contents of the document) in order to describe the insertion methods within the page;
- adding, in several files of the relationships, some XML lines which declare the use of an image.

The OOXML format gives new opportunities to the community, as stated by Microsoft [3]. In fact with the new standard:

- it is possible to show just the text of the document. If the file is a Word document, for example, only the file *document.xml* will be analyzed without necessarily opening all the files which contain the remaining information of the document;
- the files are compressed, and consequently are short and easy to manage;
- it is simpler to scan for viruses or malicious contents thanks to its textual format instead of the old binary one;
- the new format does not allow to have macro inside it, thus guaranteeing a satisfactory level of security;

- if some of the 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. The new suite, for example, sends 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 on the Internet. In [13] Microsoft analyzes, describing some of their characteristics, all the new features introduced in the new version (Office 14), highlighting the updated parts with respect to the old one.

The management flexibility offered by the new OOXML format has obvious implications when dealing with security. On one hand, the clear-text offers the seeming impossibility to hide information. While, on the other, it offers the possibility to malicious parties to read its content and eventually freely 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. Castiglione et al. [14] introduced a steganography system which can be applied to all versions before MS-Office 2007. Furthermore, authors analyzed the information leakage [15] issue raised by MS-Office 2007 documents.

The feature called *Document Inspector* has been introduced by Microsoft to enhance the security and privacy of the Office documents. It is accessible from the menu "File" -> "Info" -> "Check for Issues" (see Fig. 2). The Document Inspector makes it possible to find and remove, quickly, personal, sensitive and hidden information (see Fig. 3). More details on the *Document Inspector* can be found in [16].

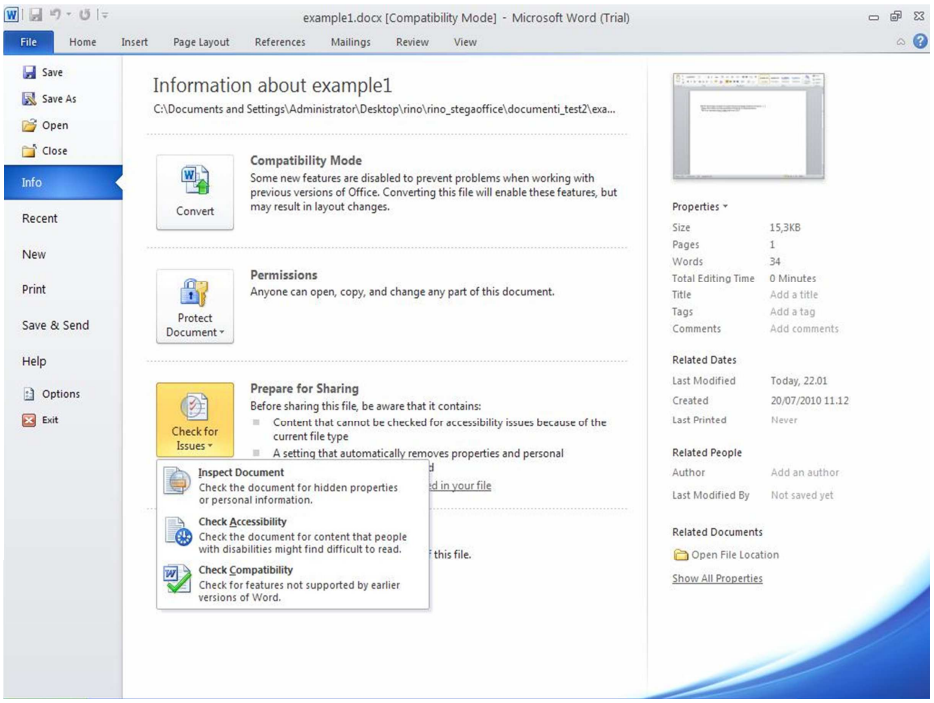

Figure 2: How to find the Document Inspector feature inside Microsoft Word.

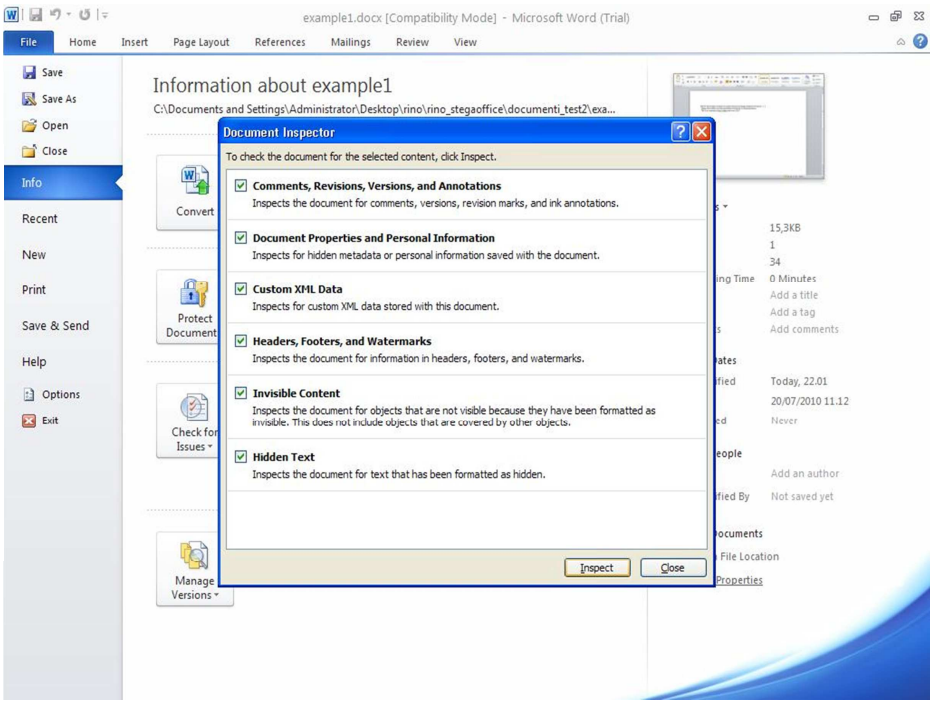

Figure 3: What can be found and removed with the Document Inspector.

3 Information Hiding in MS-Office Files

The Information Hiding is a field of the Information Security. This term refers to several techniques used to hide information in different types of "digital containers" (transmission channels, documents, audio, video, programs, images, etc). The reasons for hiding information can be different, for example to put a digital seal to a digital content that can not be removed or altered (digital watermarking) or to create "covert channel" where it is difficult to derive details about the transceiver. An important discipline of the Information Hiding is the Steganography. While Cryptography studies how to protect the content of a message, Steganography deals with the methodologies that can be used to hide the presence of the hidden message. The origin of the word Steganography is from the Greek $\sigma\tau\varepsilon\gamma\alpha\nu\varsigma$ and $\gamma\rho\alpha\varphi\iota\alpha$ and that literally it means "hidden writing". Currently it is interpreted like the possibility to hide information in other information. Generally, a model that hide data in other data can be described as follows. The data to be hidden is the message that need to be transmitted in a secure way. Usually this information is hidden into an "innocuous message" called *cover text*, *cover image* or *cover audio*, depending on if produces a *stego-text* or a *stego-object*. A *stego-key* is used to control the process to hide data as well as to limit the identification and the recovery of information to authorized users. The authorized users are those that know the right process for obtaining the value of the key. Attacks to the Steganography usually focus on finding the presence of an hidden message or on identifying the content of an hidden message.

The aim of the Steganography is to establish an hidden communication channel between two subjects. Petitcolas, in [17], defined a stegosystem like "one where an opponent who understands the system, but does not know the key, can obtain no evidence (or even grounds for suspicion) that a communication has taken place".

Considering the classification of Steganography proposed by Bauer [18] and depicted in Figure 4, in the following, the proposed methodologies are placed in the right category.

Bauer classifies Steganography in two categories: *Linguistic steganography* and *Technical steganogra-*

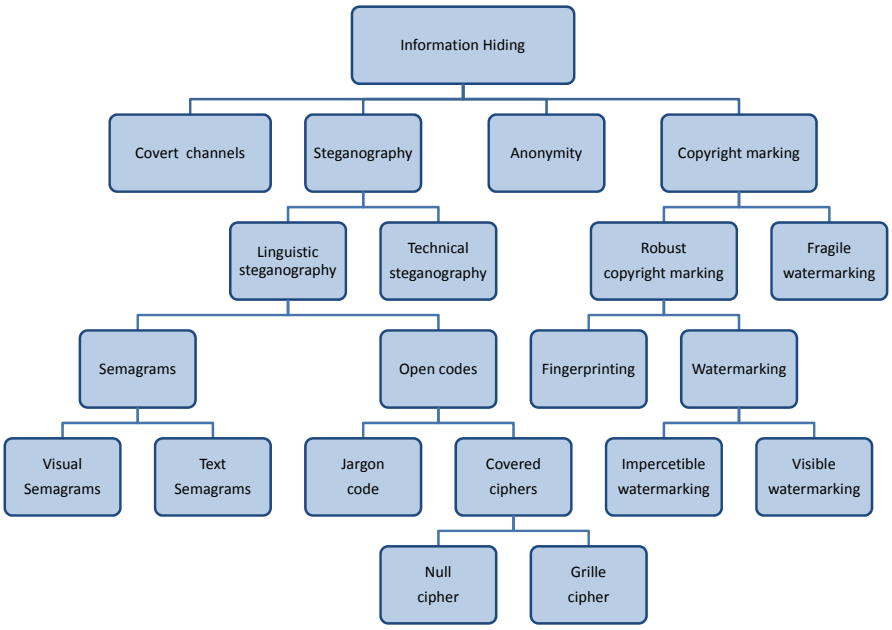

Figure 4: Classification of Information Hiding adapted from Bauer.

phy. The *Linguistic steganography* conceals the message in the carrier by means of not obvious methods while *Technical steganography* uses scientific methodologies in order to hide the information. The first technique is further divided into *Semagrams* and *Open Codes*. With the *Semagrams*, the information is usually hidden by using symbols and signs. The *Visual Semagram* uses innocent-looking or everyday physical objects to communicate a message (for example, a small paint or disposition of elements on a desk). In the *Text Semagram* the message is concealed exploiting the different ways of visualization of the 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 differs from the *Semagrams* because it hides the information through "legitimate carrier message" in the way that the presence of any information does not introduce any suspicion. Usually the "carrier message" is called *overt communication* while the hidden message is the *covert communication*. This category is classified in:

- *Jargon code*, as suggests the name, is the use of a language understood by a limited group of people. The *Jargon code* includes the "warchalking", that is the use of symbols to indicate the presence of a wireless network. This terminology is mostly used in the wild and is composed of innocent conversation that hide special meanings, having a means only to those interlocutors who share some common interests. A subset of the *Jargon codes* are the *cue codes ciphers* just hides a message on the carrier. The hidden information can be recovered by anyone who knows the secret used for the concealement. The *Grille cipher* needs the use of template. The template is applied on the carrier message, in a way that only the characters that compose the secret message are visible while the other are obfuscated. In the *Null cipher* the message is hidden using a set of rules established among the users. For example, to read every seven words or to consider the second character of each word.

Figure 5 shows the collocation of the different techniques proposed in this paper, according to the Bauer classification. The methodology of "Data Hiding by Office Macro" is not present because it can be

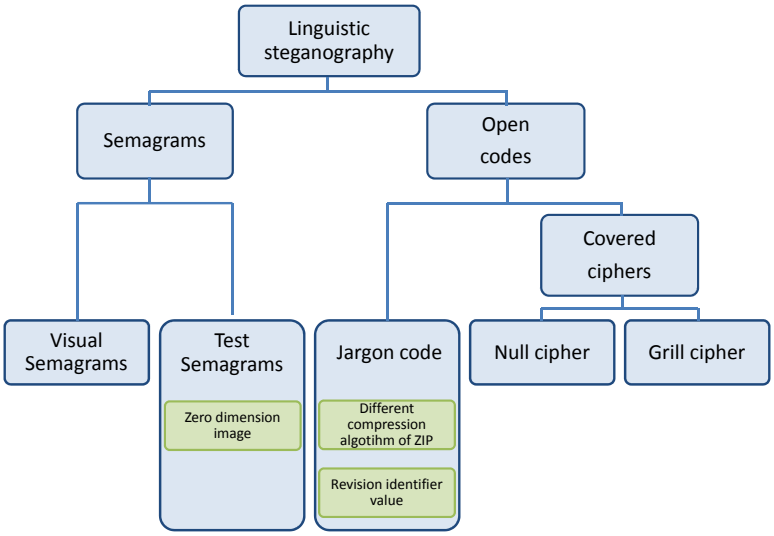

Figure 5: Classification of Information Hiding for MS-Office documents.

included in at least three categories, the *Test Semagram*, the *Null cipher* and the *Jargon code*.
The methodologies analyzed in this article have been classified as "Steganography" techniques. In fact, the methodologies are not *Anonymity* techniques, because they do not hide the sender and the receiver identity. For this reason, having the different version of the cover document, it is possible to identify who inserted the hidden data in the document.
The proposed methodologies cannot be considered *Covert channels* because they do not satisfy the definition of Lampson. 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. At last, the methodologies are not classified as *Watermarking* because they do not satisfy the second and third attribute presented in [19]. Watermarking is distinguished from other techniques in three important aspects. 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 into other file formats. Finally, watermarks undergo the same transformations of the "works". This means that it is possible, sometimes, to learn something about those transformations by looking at the resulting watermarked objects. These three attributes make watermarking invaluable for certain applications. Indeed, the hidden information is not always "*inseparable from the work*", for example, many do not resist to save operations or can be removed without damaging the file, and not always "*undergo the same transformations*" of the container file (usually transformations produce the loss of the hidden data).

4 Data Hiding by Different Compression Algorithm of ZIP

Taking advantage of the characteristic that OOXML standard produces compressed files, it is possible to hide information inside a ZIP structure without taking 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 DeflatS format [20], 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 1.

Table 1: Compression options in the ZIP format.

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

Therefore, by inserting in the command `wzzip [options] zipfile [@listafile] [files...]` one of the options indicated in Table 1, the desired algorithm compression will be applied. It is worth noting that, in a container ZIP, all the files contained can be compressed with a different algorithm. In the MS-Office files, that are ZIP containers, it is possible to set various compression algorithms.

Not all the algorithms listed in Table 1 are correctly interpreted by MS-Office. In fact, after some tests, it has been possible to ascertain that only the 5 algorithms present in Table 2 are supported by MS-Office. Initially, the tests has been performed on a .docx file, which has been compressed by using the different compression algorithms. It as been determined that both MS-Office 2007 and MS-Office 2010 do not correctly handle file compressed 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. MS-Office uses by default the compression algorithm named *DeflateS*.

Table 2: Association characters-algorithms.

Algorithm	**Option**	***Char***
DeflatF	*ef*	0
DeflatN	*en*	1
DeflatX	*ex*	2
DeflateS	*es*	3
Stored	*e0*	4

The proposed steganographic technique considers different compression algorithms as different parameters of source encoding. More precisely:

1. hidden data is codified with an alphabet of 5 elements, the 5 different values that indicate the compression algorithm used;

2. the codes obtained through the previous point are hidden in ZIP files associating a character to every file present in the container;

3. the compression algorithm applied to the single file corresponds to the value of the character to be hidden.

Consider the binary string $(10101011011111111001000100001)_2$ 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 3.
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 codifies or decodes. In order to hide the binary

Table 3: Decoding table.

File	Algorithm	*Char*
$[Content_Types].xml$	DeflatS	3
$\backslash docProps\backslash app.xml$	DeflatS	3
$\backslash docProps\backslash core.xml$	DeflatX	2
$\backslash word\backslash document.xml$	DeflatF	0
$\backslash word\backslash fontTable.xml$	DeflatN	1
$\backslash word\backslash settings.xml$	DeflatS	3
$\backslash word\backslash styles.xml$	Stored	4
$\backslash word\backslash stylesWithEffects.xml$	DeflatS	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
$\backslash \_rels\backslash .rels$	DeflatS	3

string, it has to be first converted into a number in base 5. The base 5 representation of the number $(10101011011111111001000100001)_2$ is a string of 12 numbers: $(332013432413)_5$. It is assumed that the values indicated in Table 2 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 3). □

If the MS-Office file contains M files, the proposed technique allows to hide

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

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

Algorithms 1 and 2 (see Appendix A) show how to encode and decode information using the methodology of different compression algorithm of ZIP.

5 Data 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 [21]. Thus, a macro is a simple recording of sequence of commands which are already available in a software. For this reason, there would seem no need for a programming language. However, macro has acquired a programming language that, in the event of MS-Office, is Visual Basic. 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. 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 inside of 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 assigned to a variable.

In the second case, as consequence of specific input, macro has a behavior that generates an output that renders 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, the characters of the hidden message will not be explicitly present but only the coordinates of the corresponding position in the document;
- only who has stego-key will know the secret.

This methodology does not place limits on the amount of information that can be hidden. In fact, a macro does not pre-exist but is created or modified according to the data to be hidden.

6 Data 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. 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 in the ECMA-376 specifications [10]). 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 ECMA specifications [10]). The coordinates (line, 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 ECMA specifications [10]). 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 not possible 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 (see Appendix A) show how to encode and decode information using the methodology of Zero Dimension Image.

7 Data Hiding by the 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 ECMA-376 specifications [10]).

The methodology proposed in this section consists of replacing the values of the *rsid* attributes with the data to be hidden, codified in hexadecimal. 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 section, 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 file `document.xml` which are not present in the file `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 3 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>
```

That being stated, the methodology presented in this Section 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.

As an example, 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 (19x8) characters to store information (see Table 4).

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 5).

Table 4: Sequence of *rsid* values.

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 5: Coded message.

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																			

A sequence of 82 characters is obtained, with a further 70 symbols "0" attached. Thus, a string of 152 symbols is obtained (see Table 6).

Finally it will be enough to replace, in an XML file, the string of symbols in Table 6 to the values of the *rsid* attributes in order to complete the steganography process.

```
<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 (see Section 8). □

Algorithms 5 and 6 (see Appendix A) show how to encode and decode a secret message using the methodology analyzed in this Section.

8 Methodologies Compared

The *Document Inspector*, as indicated in Section 2, is the tool supplied by Microsoft, which is used to search for and remove any eventual information hidden in MS-Office files. Thus, for an Information Hiding methodology to be considered good, it must pass the controls of this tool. All four methodologies presented in this paper resist 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 of the ZIP container every time the file is saved. These operations consist of eliminating 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 Sections 4 and 7 vulnerable. In fact, as a result of a save action, MS-Office compresses all the present files in the ZIP container using the default algorithm (DeflateS) and assigns new values to the *rsid* attributes. Therefore, in order to avoid that the hidden information

Table 6: Sequence of *rsid* values with hidden data.

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				

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. It is impossible to make any general considerations about the overhead introduced by the hiding methods introduced in this paper. However, there is a need to examine the single methodologies. In the event discussed in Section 4, 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. On the other hand, the methodology presented in Section 7 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. The overhead introduced by the solution proposed in Section 6 is a function of two values. These values are the dimension of the attached file image, that contains the hidden data, plus the dimension of the elements added in the XML files and required in order to insert the image with the characteristics described in Section 6. Finally, in the case discussed in Section 5, the overhead introduced is a function of the dimension of the macro applied.

The four methodologies discussed in this paper 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, it should be encrypted using a symmetrical key algorithm.

9 Experiments

In order to verify the real amount of information that can be hidden by using the proposed techniques, several tests have been done on a set of 53.841 files composed of Word, Excel and PowerPoint documents (see Fig. 6). The files have been downloaded from the Internet by different domains, as shown in Fig. 7.

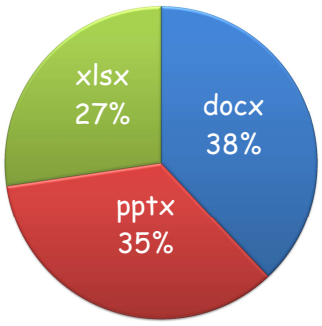

Figure 6: Distribution of the test files used for the experiments.

The amount of files analyzed in this section, classified by type and domain, is shown in Tab. 7.

For the experiments the authors 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 used to hide information. In fact, when using the *Zero Dimension Image* methodology, the amount of secret information is function of the size of the added "image" and of 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.

As a consequence, the experiments have been conducted only on the *Different Compression Algorithm of ZIP* and on the *Revision Identifier Value* methodologies. In order to establish the amount of information

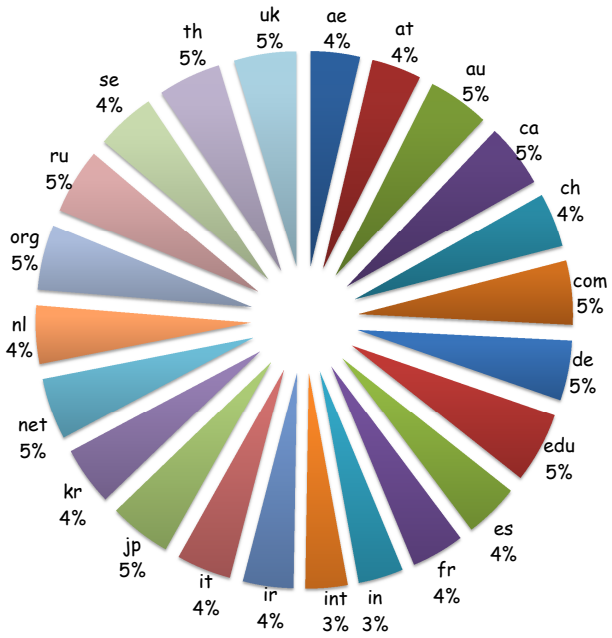

Figure 7: Test files distribution based on the domain of download.

Table 7: Files per domain.

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

that can be hidden, the number of *rsid attributes* within the XML files as well as the number of files present in the ZIP container, have been counted. The result of such operations and the properties of the considered documents are shown in Table 8.

As shown in Section 7, the number of bits that can be hidden by using the *Revision Identifier Value* methodology is 32 times the number of *rsid attributes*. In addition, as shown in Section 4, the amount of bits that can be concealed by using *Different Compression Algorithm of ZIP* methodology is 2.32

Table 8: Statistics on the data set.

	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 extracted files	777.234	12.560	287.817.186	4.009.739	79.100	287.802.760	943.787	15.048	342.234.161
Dimension ZIP file	287.134	4.035	16.607.100	2.320.961	24.983	59.011.334	120.874	6.333	27.721.494
% of compression	38,11%	0,91%	99,71%	58,75%	0,95%	99,65%	20,27%	2,18%	99,51%
Dimension XML file	207.256	0	85.930.158	311.197	0	25.545.191	365.378	666	170.994.712
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

times the number of files present in a ZIP container. 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. 9.

Table 9: Amount of information (in bytes) and the percentage that can be hidden using the different metodologies.

		Word files		PowerPoint files		Excel files	
		Revision Identifier Value	***Different Compression Algorithm of ZIP***	***Revision Identifier Value***	***Different Compression Algorithm of ZIP***	***Revision Identifier Value***	***Different Compression Algorithm of ZIP***
Amount Hidden Bytes	***Avg***	5.990	6	0	42	0	6
	Min	0	2	0	5	0	3
	Max	2.346.376	533	0	692	0	779
Percentage Information Hiding	***Avg***	6,74%	0,01%	0,00%	0,01%	0,00%	0,02%
	Min	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%
	Max	355,21%	0,08%	0,00%	0,05%	0,00%	0,08%

The experiments shown that the methodology *Different Compression Algorithm of ZIP* gives a contribution next to zero (usually only a few bytes are available) while the *Revision Identifier Value* one is the most advantageous but can be applied only to Word files. In conclusion, to maximize the amount of information that can be hidden within each kind of MS-Office files, all the proposed techniques should be applied together.

10 Conclusions

Four new methods for hiding data in MS-Office documents have been presented in this paper. The common feature is that they resist the *Document Inspector* analysis, which could not detect any hidden information. The first two techniques, which use different compression algorithms as well as revision identifier values, exploit particular features of the OOXML standard. These techniques have a null overhead, if the information to be hidden does not need to add any other modules. However, they do not resist save actions, in which case the hidden data is removed from the file. Whereas, the other two methodologies, which use either a zero dimension image or macro, are based on the characteristics of the MS-Office suite and are, therefore, not constrained to the OOXML format. Unlike the previous two, they resist save actions but have an overhead that depends on the sequence elements size which are inserted into the files.

The proposed techniques can be used in several scenarios where there is need of a secure communication channel to exchange information in a private manner. Due to the embedding facilities of the proposed techniques, the quantity of information that can be hidden is very short and for that reason the main usage of the presented methodologies is to exchange very short information such as passwords, pass-phrases, or short text. Furthermore, the proposed techniques can be combined in order to extend the overall quantity of data that can be hidden in a single Office file. It is important to remember that the secret information to

be embedded have to be protected in some way, for example by using some cryptographic means in order to avoid eavesdropping by a malicious party.

References

[1] A. Castiglione, B. D'Alessio, A. D. Santis, and F. Palmieri, "New steganographic techniques for the ooxml file format," in *Proc. of MURPBES 2011, Vienna, Austria, LNCS*, vol. 6908. Springer-Verlag, August 2011, pp. 344–358.

[2] Microsoft Press Release, "Microsoft office 2010 now available for consumers worldwide," http://www.microsoft.com/presspass/press/2010/jun10/06-152010officelaunchpr.mspx, visited March 2011.

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

[4] Z. Hao-ran, H. Liu-sheng, Y. Yun, and M. Peng, "A new steganography method via combination in powerpoint files," in *Proc. of 2010 International Conference on Computer Application and System Modeling (ICCASM'10), Taiyuan, China*, vol. 2. IEEE, October 2010, pp. V2–62–V2–66.

[5] M.-Q. Jing, W.-C. Yang, and L.-H. Chen, "A new steganography method via various animation timing effects in powerpoint files," in *Proc. of the 2009 International Conference on Machine Learning and Cybernetics, Baoding, China*. IEEE, July 2009, pp. 2840–2845.

[6] I.-C. Lin and P.-K. Hsu, "A data hiding scheme on word documents using multiple-base notation system," in *Proc. of the 6th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP'10), Darmstadt, Germany*. IEEE, October 2010, pp. 31–33.

[7] 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, no. 1, pp. 24–30, 2007.

[8] S. L. Garfinkel and J. J. Migletz, "New xml-based files implications for forensics," *IEEE Security & Privacy*, vol. 7, no. 2, pp. 38–44, 2009.

[9] B. Park, J. Park, and S. Lee, "Data concealment and detection in microsoft office 2007 files," *Digital Investigation*, vol. 5, no. 3-4, pp. 104–114, 2009.

[10] ECMA International, "Office Open XML File Formats - Part 1," Standard ECMA-376, 2nd edition, December 2008, http://www.ecma-international.org/publications/standards/Ecma-376.htm.

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

[12] M. C. Erika 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.

[13] Microsoft Corporation, "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.

[14] 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, no. 5, pp. 750–764, 2007.

[15] 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, no. 1, pp. 50–62, 2011.

[16] 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.

[17] F. Petitcolas, R. Anderson, and M. Kuhn, "Information hiding-a survey," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1062–1078, July 1999.

[18] F. L. Bauer, *Decrypted Secrets: Methods and Maxims of Cryptolog, 4rd ed.* Springer, 2007.

[19] 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, 2008.

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

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

Aniello Castiglione joined the Dipartimento di Informatica ed Applicazioni "R. M. Capocelli" of Università di Salerno in February 2006. He received a degree in Computer Science and his Ph.D. in Computer Science from the same university. He is a reviewer for several international journals (Elsevier, Hindawi, IEEE, Springer) and he has been a member of international conference committees. He is a Member of various associations, including: IEEE (Institute of Electrical and Electronics Engineers), of ACM (Association for Computing Machinery), of IEEE Computer Society, of IEEE Communications Society, of GRIN (Gruppo di Informatica) and of IISFA (International Information System Forensics Association, Italian Chapter). He is a Fellow of FSF (Free Software Foundation) as well as FSFE (Free Software Foundation Europe). For many years, he has been involved in forensic investigations, collaborating with several Law Enforcement agencies as a consultant. His research interests include Data Security, Communication Networks, Digital Forensics, Computer Forensics, Security and Privacy, Security Standards and Cryptography.

Bonaventura D'Alessio received a degree in Computer Science from the University of Salerno 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 Minister of Justice. Since 2000 he is an Officer in Permanent Service, with a Technical Role in computer science, of the Carabinieri Force. He have currently 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 unit involved in the management of systems and networks security, at the Carabinieri General Headquarters. From April 2006 to May 2010 he was the chief of the unit involved in the management of S.I.T.A. (Information System for Environmental Conservation) of Carabinieri Environmental Care. He is currently a Ph.D. student in Computer Science from the University of Salerno. His research interests include Communication Networks, Data Security, Digital Forensics, Security and Privacy, Steganography, Digital Forensics.

Alfredo De Santis received a degree in Computer Science (cum laude) from the Università di Salerno in 1983. Since 1984, he has been with the Dipartimento di Informatica ed Applicazioni of the Università di Salerno. Since 1990 he is a Professor of Computer Science. From November 1991 to October 1995 and from November 1998 to October 2001 he was the Chairman of the Dipartimento di Informatica ed Applicazioni, Università di Salerno. From November 1996 to October 2003 he was the Chairman of the PhD Program in Computer Science at the Università di Salerno. From September 1987 to February 1990 he was a Visiting Scientist at IBM T. J. Watson Research Center, Yorktown Heights, New York. He spent August 1994 at the International Computer Science Institute (ICSI), Berkeley CA, USA, as a Visiting Scientist. From November 2009 he is in the Board of Directors of Consortium GARR (the Italian Academic & Research Network). His research interests

include Algorithms, Data Security, Cryptography, Information Forensics, Communication Networks, Information Theory, and Data Compression.

Francesco Palmieri is an assistant professor at the Engineering Faculty of the Second University of Napoli, Italy. His major research interests concern high performance and evolutionary networking protocols and architectures, routing algorithms and network security. Since 1989, he has worked for several international companies on nation-wide networking-related projects and, from 1997 to 2010 he has been the Director of the telecommunication and networking division of the Federico II University, in Napoli, Italy. He has been closely involved with the development of the Internet in Italy as a senior member of the Technical-Scientific Advisory Committee and of the CSIRT of the Italian NREN GARR. He has published more that 70 papers in leading technical journals/conferences and currently serves as Editor-in-Chief of an international journal and is part of the editorial board of several other ones.

A Algorithms

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

X is 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 $C[B[i]+1]$
    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

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 pointing to the relationships file of the OLE-object inserted
MED is the folder pointing to 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
copy *S* into the folder *MED*
in the *REL* file add the relationships element
in the *CEN* file add the elements to declare as 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 pointing to the relationships file of the OLE-object inserted
MED is the folder pointing to 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

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 $J = J$ to 8 **do**
        $A = A \mid\mid A[J]$
      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
  $K = H[I] \mid\mid H[I+1]$
  B = representation of hexadecimal value K
  S = append B
  $I = I + 2$
until $I - 1 \leq L$