New Steganographic Techniques for the OOXML File Format

Aniello Castiglione, Bonaventura D'alessio, Alfredo Santis, Francesco Palmieri

New Steganographic Techniques for the OOXML File Format

Aniello Castiglione[1⋆], Bonaventura D'Alessio[1], Alfredo De Santis[1], and Francesco Palmieri[2]

[1] Dipartimento di Informatica "*R. M. Capocelli*"
Università degli Studi di Salerno, I-84084 Fisciano (SA), Italy
castiglione@acm.org, bdalessio@dia.unisa.it, ads@dia.unisa.it
[2] Dipartimento di Ingegneria dell'Informazione
Seconda Università degli Studi di Napoli, I-81031 Aversa (NA), 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 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.
This work presents some new methods of embedding information into the OOXML file format which can be extremely useful when using MS-Office documents in steganography. The authors highlight how the new methods introduced in this paper can also be used in many other scenarios and not only in MS-Office documents. 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 without a doubt the most widely used word-processing tool when preparing and writing documents, spreadsheets and presentations [14]. 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,

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

among other things, the level of security and thus making it more difficult to hide information inside them. In fact, it has gone from using the old binary format to the new OOXML [5], which uses XML files. In addition to guaranteeing a significantly high level of "privacy and security", it has also introduced the feature *Document Inspector*, which makes it possible to quickly identify and remove any sensitive, hidden and personal information, ("hiding date" 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([7], [8], [10], [11]), are still valid, and therefore independent from the version used. The new format offers new perspectives, as proposed by Garfinkel et al. [6] as well as Park et al. [16]. 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 discusses the methodology that takes advantage of the possibility to use different compression algorithms in generating MS-Office files. Section 4 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 (revision identifier). In Section 5 a methodology, that uses images not visualized by MS-Office, but are present in the file, is analyzed, in order to contain hidden information. Section 6 illustrates how the macro of MS-Office can be used to hide information. In Section 7 the methodologies are compared, verifying the overhead introduced as well as the resulting behavior of save actions.

2 The OOXML Format

Starting with the 2007 version, Microsoft has adopted the OOXML format based on XML (XML-based file format). In fact, Microsoft has begun the transition from the old logic, that saw the generation of a binary file format, to a new one that uses XML files. 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. 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 by only using standard transformation methods. This is possible because XML text is clearly written and therefore visible and modifiable

with any version of a text editor. Moreover, OLE attachments are present in the source file format 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 file Word, Excel and PowerPoint respectively being *.docx*, *.xlsx*, *.pptx*. An additional feature is that a macro is 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, which is based on the ECMA-376 standard [3], uses a container, a ZIP file, inside of which there are a series of files, mostly XML, and are opportunely organized into folders, that describe both the content as well as 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 [17]. 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 (Word, Excel, PowerPoint):

- XML files, that 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 [4] of the folders and files in a ZIP container will be like that shown in Fig. 1.

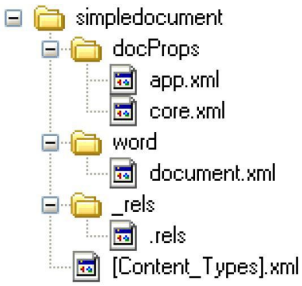

Fig. 1. Structure of a simple Word document

Therefore, beginning from version 2007, 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.);
– conform to the ECMA-376 standard, opportunely customized.

Another key concept related to the OOXML format is the modularity, either inside the files or between the same files, 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 relationship, some XML lines which declare the use of an image.

The OOXML format gives new opportunities to the community, as indicated by Microsoft [5]. 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 about the document;
– the files are compressed, and consequently are shorter and easy to manage;
– it is simpler to scan for viruses or malicious contents thanks to its textual form instead of the old binary format;
– 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 to the Internet. In [12] Microsoft analyzes, describing some of their characteristics, all the new features introduced in the new version, highlighting the updated parts in respect to the old version.

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. [1] introduced a steganography system which can be applied to all versions before MS-Office 2007. Furthermore, authors analyzed the information leakage [9] issue raised by MS-Office 2007 documents.

In order to guarantee a higher level of security and privacy, Microsoft (starting from MS-Office 2007 for Windows) have introduced the feature called *Document Inspector* that makes it possible to find and remove, quickly, personal, sensitive and hidden information. More details on the *Document Inspector* can be found in [13].

3 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 [2], 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 character-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:

- hidden data is codified with an alphabet of 5 elements, the 5 different values that indicate the compression algorithm used;
- the codes obtained through the previous point are hidden in ZIP files associating a character to every file present in the container;
- the compression algorithm applied to the single file corresponds to the value of the character to be hidden.

Example 1. 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 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

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

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

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

4 Data Hiding by the Revision Identifier Value

The second 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 [3]).

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.

Example 2 (Coding with rsid). 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).

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	

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 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				

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	

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:rsidRsidSect="00000000">
```

Obviously the message to be hidden would be preferably encrypted before embedding (see Section 7). □

5 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 [3]). 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 [3]). 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 [3]). 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.

6 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 [15]. 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.

7 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 3 and 4 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 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 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. On the other hand, the methodology presented in Section 4 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 5 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 5. Finally, in the case discussed in Section 6, 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.

8 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.

References

1. Castiglione, A., De Santis, A., Soriente, C.: Taking advantages of a disadvantage: Digital forensics and steganography using document metadata. Journal of Systems

and Software 80(5), 750–764 (2007)
2. Deutsch, P.: DEFLATE Compressed Data Format Specification version 1.3. http://www.ietf.org/rfc/rfc1951.txt (May 1996)
3. ECMA International: Final draft standard ECMA-376 Office Open XML File Formats - Part 1. In: ECMA International Publication (Dec 2008)
4. Erika Ehrli, M.C.: Building server-side document generation solutions using the open xml object model. http://msdn.microsoft.com/en-us/library/bb735940%28office.12%29.aspx (Aug 2007)
5. Frank Rice, M.C.: Microsoft MSDN. Introducing the Office (2007) Open XML File Formats. http://msdn.microsoft.com/it-it/library/aa338205.aspx (May 2006)
6. Garfinkel, S.L., Migletz, J.J.: New xml-based files implications for forensics. IEEE Security & Privacy 7(2), 38–44 (2009)
7. Hao-ran, Z., Liu-sheng, H., Yun, Y., Peng, M.: A new steganography method via combination in powerpoint files. In: Computer Application and System Modeling (ICCASM), 2010 International Conference on. vol. 2, pp. V2–62 –V2–66 (october 2010)
8. Jing, M.Q., Yang, W.C., Chen, L.H.: A new steganography method via various animation timing effects in powerpoint files. In: Machine Learning and Cybernetics, 2009 International Conference on. vol. 5, pp. 2840–2845 (july 2009)
9. Kiyomoto, S., Martin, K.M.: Model for a common notion of privacy leakage on public database. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications 2(1), 50–62 (2011)
10. Lin, I.C., Hsu, P.K.: A data hiding scheme on word documents using multiple-base notation system. In: Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 2010 Sixth International Conference on. pp. 31–33 (october 2010)
11. Liu, T.Y., Tsai, W.H.: A new steganographic method for data hiding in microsoft word documents by a change tracking technique. IEEE Transactions on Information Forensics and Security 2(1), 24–30 (2007)
12. 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)
13. 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)
14. 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)
15. MSDN Library: Introduction to macros. http://msdn.microsoft.com/en-us/library/bb220916.aspx (visited March 2011)
16. Park, B., Park, J., Lee, S.: Data concealment and detection in microsoft office 2007 files. Digital Investigation 5(3-4), 104–114 (2009)
17. Wikipedia: ZIP (file format). http://en.Wikipedia.org/wiki/ZIP_(file_format) (visited March 2011)