



BIULETYN WAT  
VOL. LXI, Nr 4, 2012

## Stego.docx — hidden communication system using docx files

KAMIL KACZYŃSKI

Military University of Technology, Institute of Mathematics and Cryptology,  
00-908 Warsaw, S. Kaliskiego 2 Str.,  
kkaczynski@wat.edu.pl

*This paper is dedicated to Prof. Jerzy Gawinecki  
on the occasion of his 60th birthday.*

**Abstract.** Stego.docx is a proposition of a simple steganographic system using some properties typical for the docx files. The proposed stegosystem meets the requirements for the first and the fourth protection level (with and without randomization). The paper contains specification of the Office Open XML and the WordProcessingML standards and classification of steganographic systems. Proposed stegosystem was implemented using Microsoft Visual Studio 2010 and .NET Framework 4.0. The analysis of the output data proved correctness of the Stego.docx stegosystem.

**Keywords:** steganography, steganalysis, docx, Office, Word, Microsoft

### 1. Introduction

Steganography is the art of creating hidden messages in such a way, that no one except the author and the intended recipient is able to find it. The term “steganography” is derived from the Greek words “stegos” (hidden) and “graphos” (writing). The main advantage of steganography over cryptography is the fact, that sending a message does not pay much attention. Messages protected with cryptography — even the strongest — will always be suspicious. Cryptography protects only the content of the messages, while steganography hides not only the message but also the identity of communicating parties.

Steganographic problem was clearly described in [1]:

“Two accomplices in a crime have been arrested and are about to be locked in widely separated cells. Their only means of communication after they are locked up will be by way of messages conveyed for them by trustees — who are known to be agents of the warden. The warden is willing to allow the prisoners to exchange messages in the hope that he can deceive at least one of them into accepting as a genuine communication from the other either a fraudulent message created by the warden himself or else a modification by him of a genuine message. However, since he has every reason to suspect that the prisoners want to coordinate an escape plan, the warden will only permit the exchanges to occur if the information contained in the messages is completely open to him — and presumably innocuous. The prisoners, on the other hand, are willing to accept these conditions, i.e., to accept some risk of deception in order to be able to communicate at all, since they need to coordinate their plans. To do this they will have to deceive the warden by finding a way of communicating secretly in the exchanges, i.e., of establishing a »subliminal channel« between them in full view of the warden, even though the messages themselves contain no secret (to the warden) information. Since they anticipate that the warden will try to deceive them by introducing fraudulent messages, they will only exchange messages if they are permitted to authenticate them.”

## 2. Classification of steganographic systems

The definitions of following terms can be found in [2]:

Steganographic field (area)  $SF$  is steganographic channel space, steganographic objects, embedding and detecting methods.

$SF = (SC, C, M, K, \hat{C}, E, D)$ , where steganographic objects are:

- host signal  $c$  (container), that belongs to the set of all containers  $C$ ;
- message signal  $m$  (message), that belongs to the set of all message signals  $M$ ;
- key  $k$ , that belongs to the set of all keys  $K$ ;
- modified container  $\hat{c}$  (stego), that belongs to set of all modified containers  $\hat{C}$ .

The embedding (coding) method is a method for embedding a message signal in a host signal. The modified signal (stego) should satisfy all steganographic requirements, like: the size of modified signal should be the same as the size of the host signal. What is more, stego should not interfere the host signal in any visible way.

The decoding method is a method used for decoding the modified signal in order to get the message signal.

The steganographic channel space (SC) is a frequency or/and amplitude container part, that is available for steganographic modification and message signal transmission.

The modified container  $\hat{c}$  should preserve the host signal size, and should not introduce any visible distortions to it.

The steganographic system SS is a set of encoding and decoding methods executed on steganographic objects, including the restrictions of the steganographic channel space.

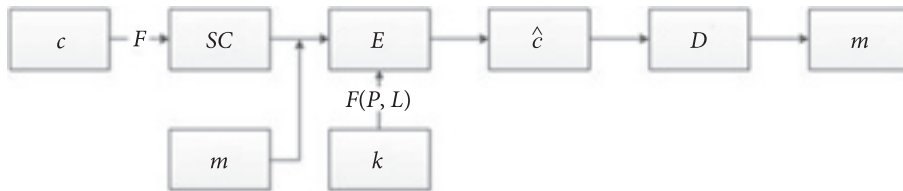


Fig. 1. Data flow in steganographic system [2]

On the basis of [2] there are four protection levels provided by the steganographic system.

The first level of protection is determined only by the choice of the embedding algorithm. It may be the least significant bits modification algorithm, or the algorithms modifying the frequency or spatial-temporal characteristics of the container.

Steganographic systems of the second, third and fourth protection level use a key (password) to make a steganographic modification. The table presented below contains the detailed classification of the steganographic systems.

TABLE 1

The classification of steganographic systems [2]

Protection level for steganographic channel	Existence of steganographic algorithm	Existence of key	Key influence on the message signal bits distribution per container	Key influence on the message signal bits selection and distribution per container
1	+	–	–	–
2	+	+	–	–
3	+	+	+	–
4	+	+	+	+

### 3. Office Open XML

ECMA-376 (ISO/IEC 29500, Office Open XML) is an open standard for electronic documents, specifying XML vocabularies for describing spreadsheets, charts, presentations and word processing documents.

The main purpose of creating the standard was to enable the implementation of the Office Open XML formats by the widest set of tools and platforms, supporting interoperability across office productivity applications and line-of-business systems and to support document archival and preservation. The standard is fully compatible with existing Microsoft Office file formats.

In December 2006, ECMA International ratified the specification of Office Open XML as the ECMA-376 standard. 3 April, 2008 ISO decided in voting, that it would be an ISO/IEC 29500 standard.

Office Open XML standard defines how the word processing documents, spreadsheets and presentations should be represented. For this purposes, the following markup languages can be used:

- WordprocessingML — for word processing documents,
- SpreadsheetML — for spreadsheets,
- PresentationML — for presentations,
- DrawingML — for specifying the location and appearance of drawing elements,
- Math — for specifying the location and appearance of mathematic equations,
- Bibliography — for use in citation and bibliography.

The Office Open XML specification can be used without paying any license fees. In addition, Microsoft provided “Covenant Not to Sue” resigning to sue for its patent license violations.

Following companies and organizations have taken part in creating the standard: Apple, Barclays Capital, BP, The British Library, Essilor, Intel, Microsoft, NextPage, Novell, Statoil, Toshiba and Library of Congress.

A document created with Open Office XML is represented by a series of related files stored in ZIP archive. The WordprocessingML document contains files storing text, images referenced in the created text and a part describing characteristic, style and fonts of the created document. The SpreadsheetML document contains separate files for each of the spreadsheets and images associated with them. In the PresentationML document, each slide of the presentation is represented by a separate file.

#### 3.1. WordprocessingML

The main part of the WordprocessingML archive is a file which stores the information about processed text. This file contains body section, consisting of one

or more paragraphs. A paragraph is composed of one or more runs, which includes one or more parts of the text with the same properties (e.g. font). Each paragraph and run can be assigned a set of characteristics describing its content.

The WordprocessingML document is organized into sections (one or many). The appearance of the document is determined by the properties of each section. Each section can have its own header and footer. The archive also contains a file responsible for defining the styles used (specifying the display format of the text). Each style can have its own properties, which are assigned to individual paragraphs and runs. Using the styles allows us to limit the number of repetitive text formatting definitions, and can also reduce the users' effort to create the document. With styles, the appearance of the whole document can be easily changed by simply changing only one definition. Series of paragraphs can be combined into numbered or bulleted lists. There is also an additional file that stores information about the definition and the style of the selected list. The WordprocessingML can also include other elements: tables, automatic texts (e.g. page numbering, date) and hyperlinks.

The data contained in WordprocessingML document is stored in many small files grouping some functionality, e.g. content of all the footers of the document is stored in one file, although each section can have up to three different headers and footers (to handle different headers and footers for even and odd pages and for first page).

Document archive should contain the file defining the relationships between files in the archive and the file that specifies specifying the type of the content. Below is a content of a sample file specifying the content types ([Content\_Types.xml]):

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<Types xmlns="http://schemas.openxmlformats.org/
package/2006/content-types">
<Default ContentType="application/vnd.openxmlformats-
package.relationships+xml" Extension="rels"/>
<Default ContentType="application/xml" Extension="xml"/>
<Override ContentType="application/vnd.openxmlformats-
officedocument.wordprocessingml.document.main+xml"
PartName= "/word/document.xml"/>
<Override ContentType="application/vnd.openxmlformats-
officedocument.wordprocessingml.styles+xml" PartName= "/
word/styles.xml"/>
<Override ContentType="application/vnd.openxmlformats-
officedocument.extended-properties+xml" PartName= "/
docProps/app.xml"/>
```

```

<Override ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.settings+xml" PartName="/word/settings.xml"/>
<Override ContentType="application/vnd.openxmlformats-officedocument.theme+xml" PartName="/word/theme/theme1.xml"/>
<Override ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.fontTable+xml" PartName="/word/fontTable.xml"/>
<Override ContentType="application/vnd.openxmlformats-officedocument.wordprocessingml.webSettings+xml" PartName="/word/webSettings.xml"/>
<Override ContentType="application/vnd.openxmlformats-package.core-properties+xml" PartName="/docProps/core.xml"/>
</Types>

```

Content of file defining relationships within the document:

```

<?xml version="1.0" encoding="UTF-8" standalone="true"?>
-<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
<Relationship Target="docProps/app.xml" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties" Id="rId3"/>
<Relationship Target="docProps/core.xml" Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties" Id="rId2"/>
<Relationship Target="word/document.xml" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/office Document" Id="rId1"/>
</Relationships>

```

Below is a content of a sample WordprocessingML archive:

```

/[Content_Types].xml
/_rels/.rels
/docProps/app.xml
/docProps/core.xml
/word/document.xml
/word/_rels/document.xml.rels

```

```
/word/fontTable.xml  
/word/settings.xml  
/word/styles.xml  
/word/theme/theme1.xml
```

Crucial files for the developed steganographic system are document.xml and settings.xml containing information about the edited paragraphs.

#### 4. Usage of WordprocessingML for hiding data

Steganographic system proposed in this paper uses .docx files as a container. Files are created using Microsoft Office Word 2007, later versions or other compatible software. Files should have full compatibility with the ECMA 376 standard.

It takes the author a lot of work to create an advanced and voluminous text documents. Most of them are created for a certain period of time — author repeatedly updates the content of the document or corrects previously created text. The WordprocessingML specification provides the possibility of assigning each document editing session a unique ID number (Single Session Revision ID, rsid), which should satisfy following assumptions:

- each subsequent edition session should have a unique ID number (rsid), which is greater than numbers previously used in a file,
- rsid should be a random number generated on the basis of current system time (to minimize the chance that two different sessions will be assigned the same rsid),
- any changes made to the document in the current edition session should be marked with the same rsid,
- two identical rsid values in two different documents with the same number of the first edition session (rsidRoot) means the same edition session.

The sample below shows edition sessions in a sample .docx file:

```
<w:rsids>  
  <w:rsidRoot w:val="005806B6"/>  
  <w:rsid w:val="00142417"/>  
  <w:rsid w:val="004272D3"/>  
  <w:rsid w:val="005806B6"/>  
  <w:rsid w:val="007D498F"/>  
  <w:rsid w:val="008972F3"/>  
  <w:rsid w:val="00C81557"/>  
</w:rsids>
```

The example above shows that the document was edited in six sessions. Rsid is represented as eight hexadecimal digits. The word processing software uses rsid for informational purposes only. Changing this value does not cause any changes for the word processing software or for the integrity of the document. The ECMA 376 standard does not specify how the rsid values can be used by the word processing software.

These properties of the edition session ID allows using it as a carrier for proposed stegosystem.

## 5. Specification of steganographic system

Stego.docx is a stegosystem using two protection levels consistent with the classification described in [2]. Data carrier for all protection levels is the set of all edition session identifiers of given text document. Hidden message can be any string of binary data. Each of protection levels has its own embedding and decoding methods. The amount of data that can be hidden in a single edition session ID is calculated according to the formula:

$$B = \frac{rsid_{max} - rsid_{min}}{l}, \quad (1)$$

where:  $rsid_{max}$  — the greatest edition session ID value;  
 $rsid_{min}$  — the smallest edition session ID value;  
 $l$  — the total number of edition session ID within document.

The amount of data hidden in a single rsid is calculated as the average difference between edition session IDs rounded up. This feature allows modified document to preserve similar properties of rsids.

The total amount of data that can be hidden in a file depends on the chosen protection level and for the first protection level is calculated according to the formula:

$$D_1 = B * (l - 1). \quad (2)$$

The first protection level embedding method uses all, except the first, rsids of the document. Each ID can carry B bits (calculated according to formula (1)), hence the total amount of data which can be hidden in the document is a product of the number of bits per which can be hidden in single rsid and the number of edition session minus one.

The total amount of data that can be hidden in a file in the 4<sup>th</sup> protection level is calculated according to the formula:



$$D_4 = B * (hwt(k) - k_1), \quad (3)$$

where:  $k$  — binary key string;  
 $k_1$  — first bit of a key string;  
 $hwt$  — Hamming weight of a key string.

The amount of data that can be hidden in the 4<sup>th</sup> protection level document is strictly dependent on the generated key string. The number of rsids which can hide data is dependent on the Hamming weight of the key string — data is hidden only in those rsids, for which a subsequent bit of binary key string is equal to 1. Like in the 1<sup>st</sup> protection level, there is no possibility of hiding data in the first edition session ID, so it is necessary to subtract from the Hamming weight the value of the first bit

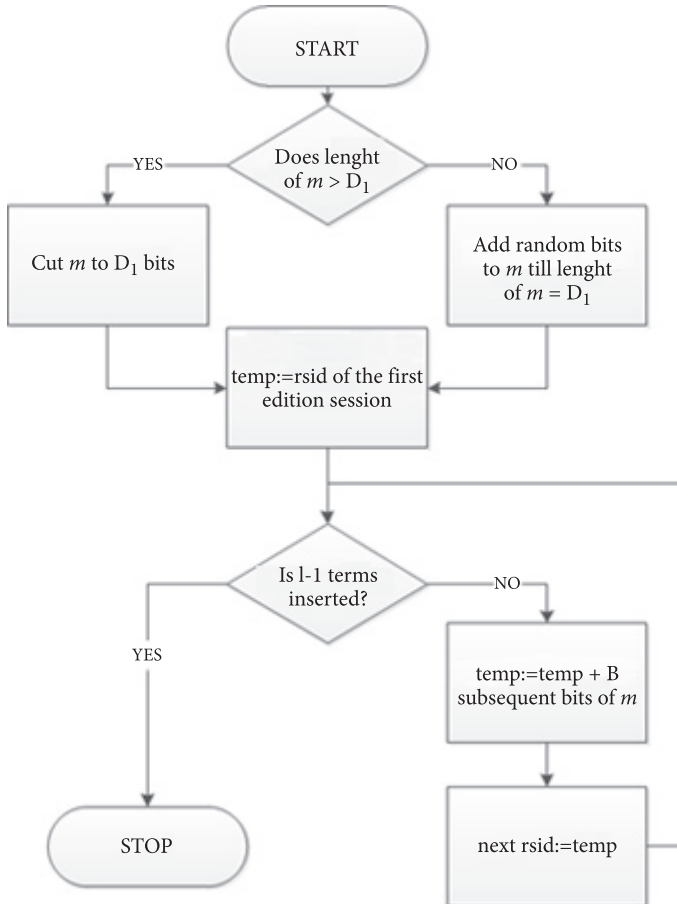


Fig. 2. The embedding method for the 1<sup>st</sup> protection level

of the key. The total number of bits possible to hide in the document is a product of the number of bits per which can be hidden in single rsid and Hamming weight of the binary key string minus the value of the keys first bit.

The embedding method for the 1<sup>st</sup> protection level is presented on the schema Fig. 2.

The decoding method for the 1<sup>st</sup> protection level is presented on the schema Fig. 3.

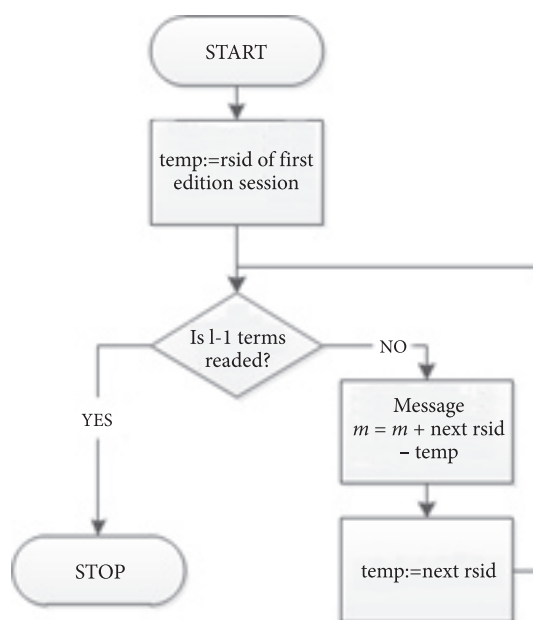


Fig. 3. The decoding method for the 1<sup>st</sup> protection level

The 4<sup>th</sup> protection level requires more sophisticated methods for decoding and embedding data. Both algorithms are key dependent. The hidden message undergoes the Knuth-Fischer-Yates permutation described in [3]. The schemas below present the embedding and decoding methods for the 4<sup>th</sup> protection level.

Properties of algorithms described above show that embedding methods do not change the modified file size comparing to the original. Embedding methods change not only the values in file settings.xml but also in document.xml file.

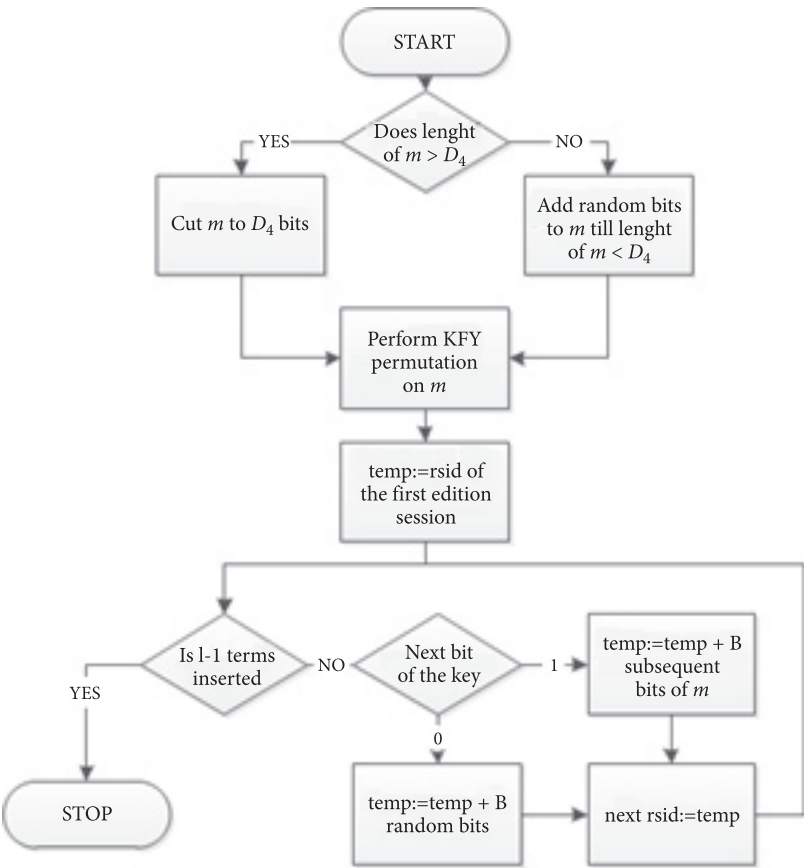


Fig. 4. The embedding method for the 4<sup>th</sup> protection level

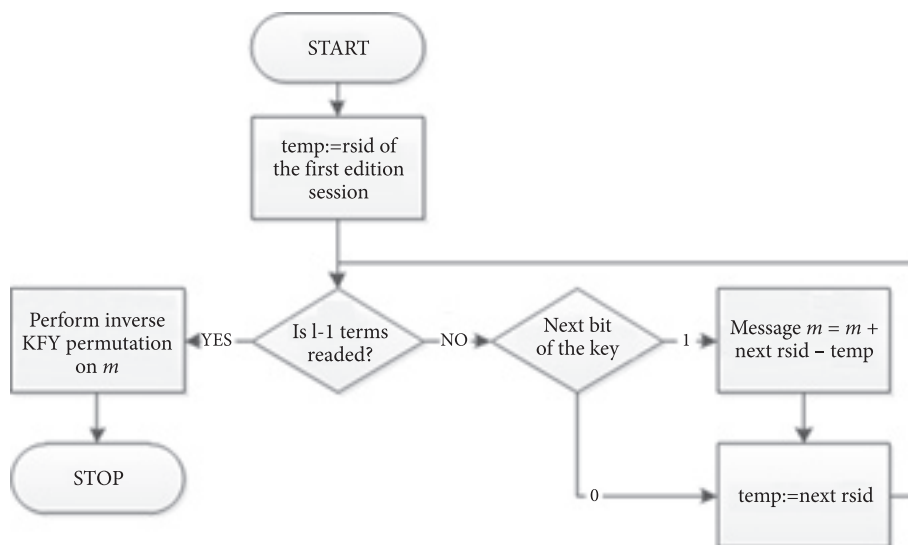


Fig. 5. The decoding method for the 4<sup>th</sup> protection level

## 6. Description of implementation

The implementation of steganographic system described in this paper has been made using Microsoft Visual Studio 2010 and C# programming language. The docx files are zip archives, so external library DotNetZip v 1.9 [4], responsible for compression and decompression of docx files, was added to project. The program has four main features:

- reading docx files and decompressing archives;
- analysis of file properties;
- hiding data in previously opened file and saving changes to a new file;
- reading hidden data from previously opened file.

The function reading docx files is validating the loaded files and then decompressing contents of the archive to a new directory named like the loaded file. If those operations were successful, next step is analysis of a file.

The main goal of file analysis is to determine the amount of data which can be hidden in it. The function uses regular expressions for load all rsids form settings. xml and then counts them. Depending on the protection level, the amount of data possible to hide in a file is calculated according to formula (2) or (3). The use of the 4<sup>th</sup> protection level requires the creation of binary string, which will be the key for hiding data. The key is generated as a hash of message entered in the appropriate field of the program. The program uses hash function SHA-512 [5], which is implemented in the System.Cryptography .NET Framework 4 library. The function also calculates the Hamming weight of the key string. After a successful completion

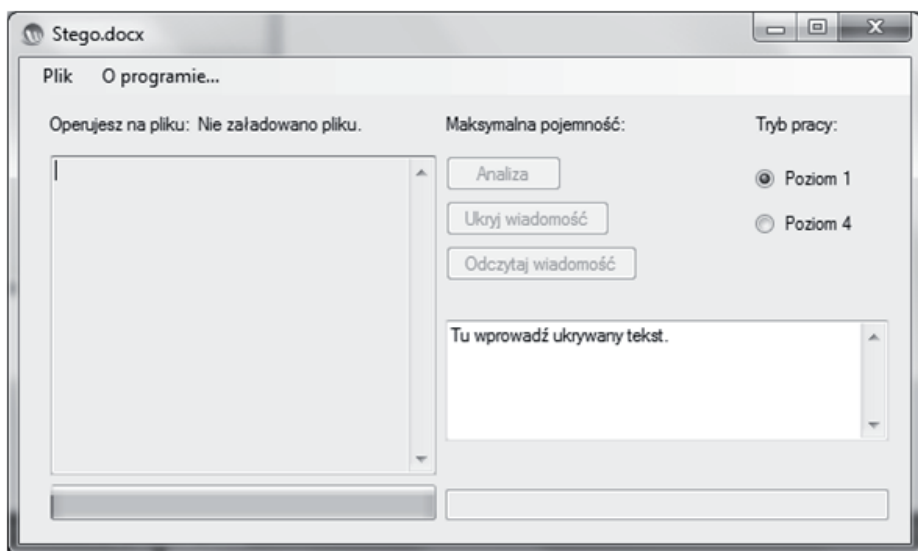


Fig. 6. Interface of Stego.docx

of the analysis it is possible to hide the message in the opened file or to read previously hidden message.

Hiding messages in the file is possible only after successful completion of its analysis. Message hidden by the Stego.docx application is a UTF-8 text. Embedding function has to match the size of entered message to the size calculated according to formula (2) or (3). Padding of a message to the specified size is performed using the pseudo-random bits generated with `RNGCryptoServiceProvider` class from .Net Framework 4. Implementation of KFY permutation used in the 4<sup>th</sup> protection level has been taken from [6]. After successful embedding message (changing `rsid` values in `settings.xml` and `document.xml`), application creates a new file "...\_stego.docx". All temporary files are deleted, the modification, last access date and the creation date are copied from a source file.

The algorithm of decoding the message hidden in the .docx file is presented in section 5. First, the analysis of the file must be performed. The carrier file was not tagged, so the user should know whether the file contains hidden message. If a user enters a wrong key or file that does not contain hidden message, then the application will read a random string. After a successful reading of the message, application removes all temporary files.

## 7. Security of proposed stegosystem

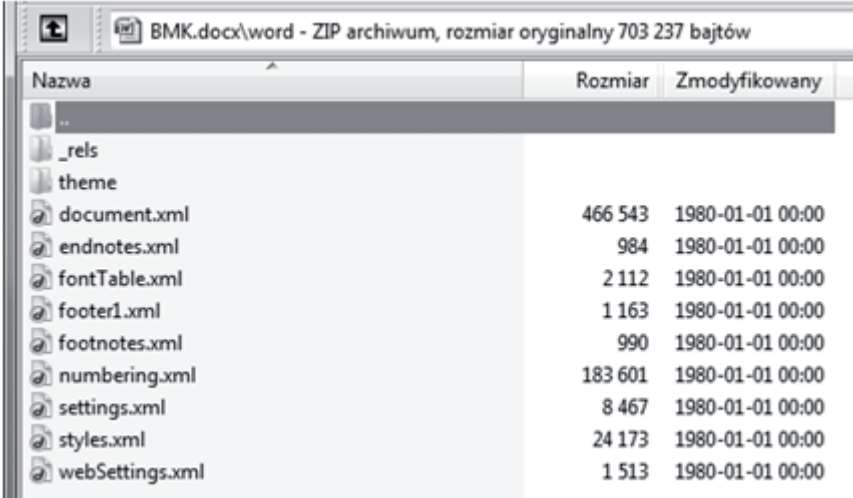
Security of the steganographic system can be considered in the same means as for cryptographic systems, using Kerkchoff's principles:

- The steganographic algorithms are publicly known.
- Steganalysist has unlimited access to the stegosystem data and unlimited time and resources for steganalysis.
- Security of the system is based only on the stegosystem keys, which are secret.

The proposed stegosystem is implemented on two security levels. The fourth protection level of Stego.docx meets all the principles mentioned above. The rsid values (random 32-bit integers) are changed in such a way, that their statistical properties are preserved. Selection of rsid values is key-dependent (key is generated with SHA-512).

## 8. Obtained results

Verifying the correctness of Stego.docx application was made using the BMK.docx file. The content of the /word directory has been presented below.



Nazwa	Rozmiar	Zmodyfikowany
..		
_rels		
theme		
document.xml	466 543	1980-01-01 00:00
endnotes.xml	984	1980-01-01 00:00
fontTable.xml	2 112	1980-01-01 00:00
footer1.xml	1 163	1980-01-01 00:00
footnotes.xml	990	1980-01-01 00:00
numbering.xml	183 601	1980-01-01 00:00
settings.xml	8 467	1980-01-01 00:00
styles.xml	24 173	1980-01-01 00:00
webSettings.xml	1 513	1980-01-01 00:00

Fig. 7. Content of /word directory — BMK.docx

Due to the similarity of operation and large size of screen captures, obtained results are presented only for the 4<sup>th</sup> protection level. Subsequent images present the behaviour of application during file analysis, embedding and decoding the message.

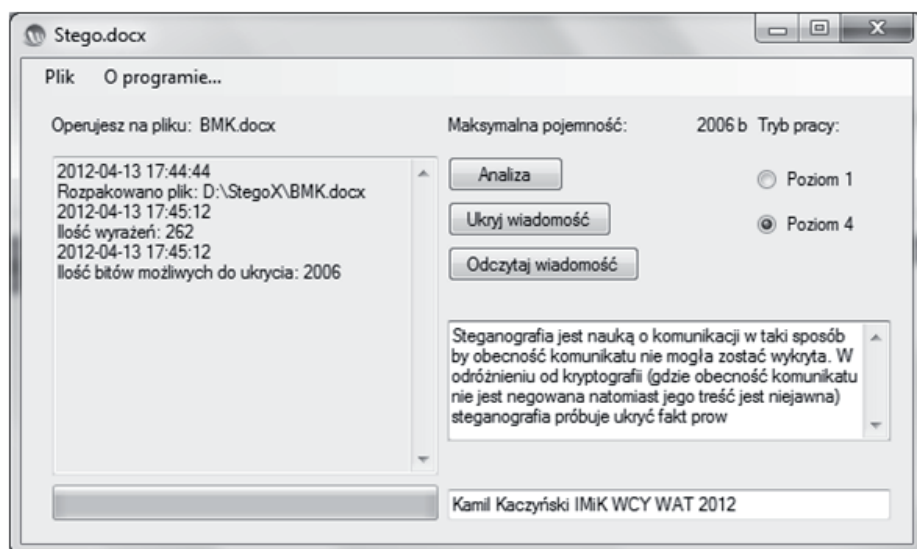


Fig. 8. Stego.docx after performing analysis of BMK.docx file

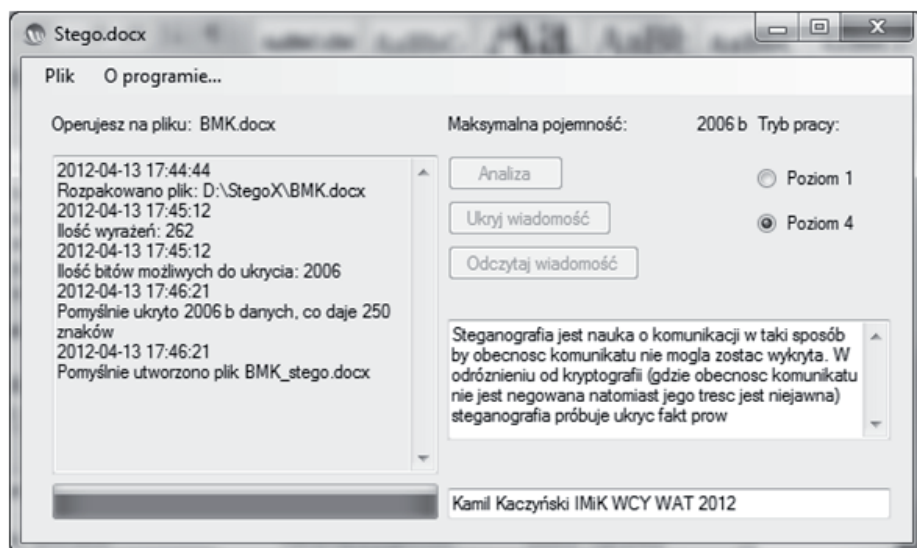


Fig. 9. Stego.docx after embedding the message

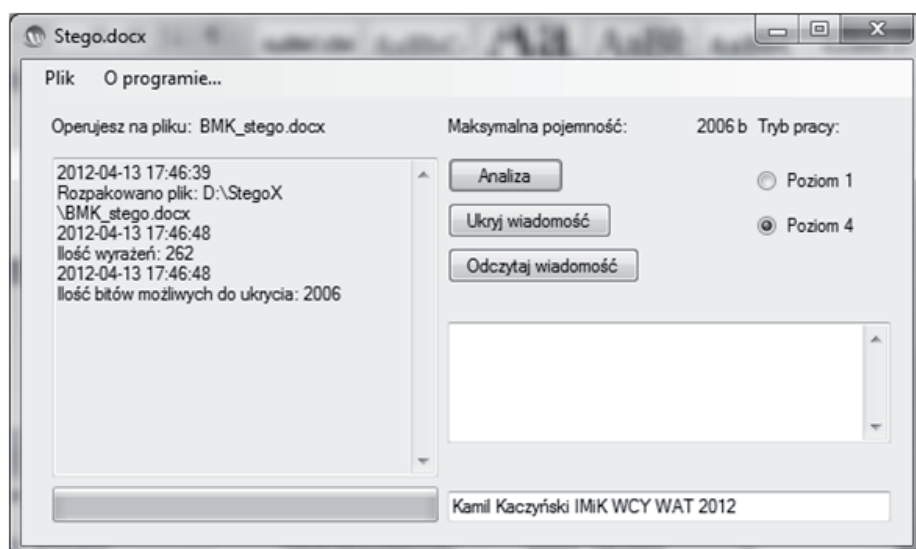


Fig. 10. Stego.docx after performing analysis of BMK\_stego.docx file

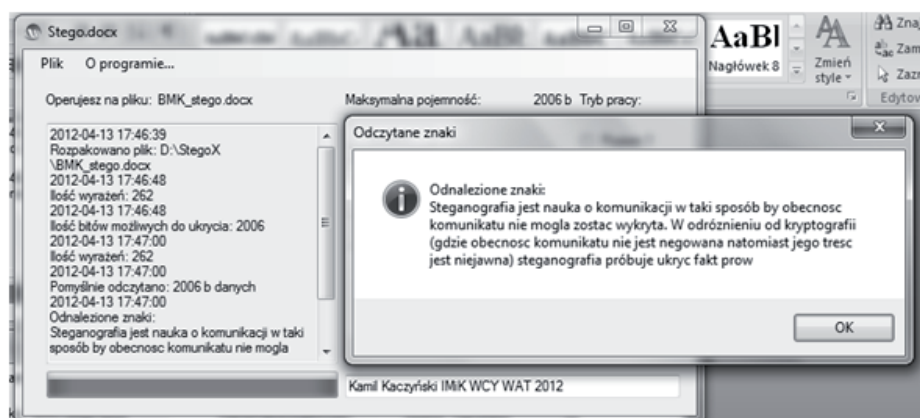
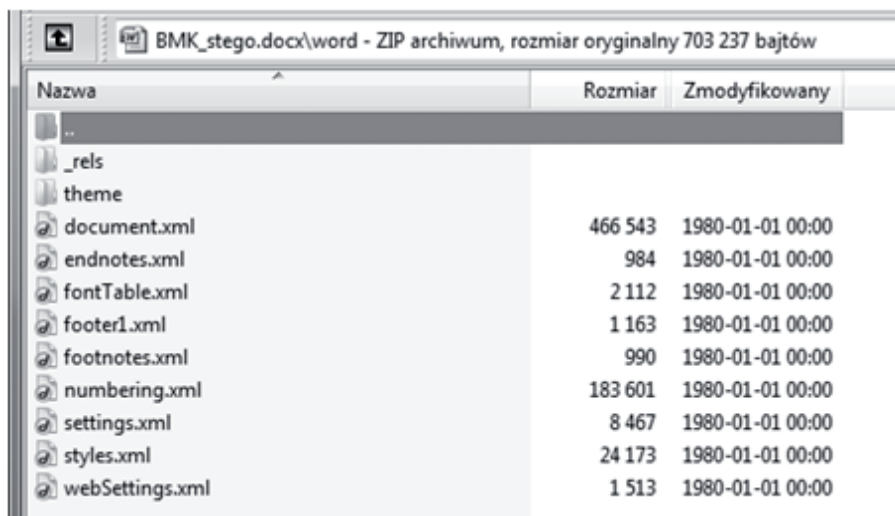


Fig. 11. Stego.docx after successful decoding of message





Nazwa	Rozmiar	Zmodyfikowany
..		
_rels		
theme		
document.xml	466 543	1980-01-01 00:00
endnotes.xml	984	1980-01-01 00:00
fontTable.xml	2 112	1980-01-01 00:00
footer1.xml	1 163	1980-01-01 00:00
footnotes.xml	990	1980-01-01 00:00
numbering.xml	183 601	1980-01-01 00:00
settings.xml	8 467	1980-01-01 00:00
styles.xml	24 173	1980-01-01 00:00
webSettings.xml	1 513	1980-01-01 00:00

Fig. 12. Content of /word directory — BMK\_stego.docx

As can be seen from the attached screenshots, Stego.docx correctly reads the message which was hidden in the BMK\_stego.docx file. The screenshot presented below shows the contents of \word directory of BMK\_stego.docx file.

The above screenshots show, that the size and date of file modification from \word directory are identical for files BMK.docx and BMK\_stego.docx. The size of uncompressed archives are also identical, and are equal to 703,237 B. Unfortunately due to the compression, the size of both compressed documents is slightly different — BMK.docx has a size of 97,797 B, when BMK\_stego.docx has a size of 97,110 B. The difference in size is caused by applied compression methods — ECMA 376 standard does not define the specific parameters of ZIP compression. It should be noted, that BMK\_stego.docx is fully supported by Microsoft Word. Editing the document increases the number of document edition sessions. Such a change does not affect the data embedded with the 1<sup>st</sup> protection level algorithm, but increasing the number of rsids for the 4<sup>th</sup> protection level causes an erroneous reconstruction of the original sequence by the inverse KFY permutation.

The amount of data that can be embedded is strictly dependent on the number of the document edition sessions. To hide a large amount of data, only files edited for a long time should be chosen.

Developed stegosystem is the first attempt of using Microsoft Word files for hiding data on the 1<sup>st</sup> and the 4<sup>th</sup> protection level. The stegosystem meets all safety criteria, so it could be used as a hidden communication system. In future versions of the system, an effort should be made to increase the capacity of .docx document. This can be achieved by the use of images, which are an integral part of a text document.

Received August 31 2012, revised September 2012.

#### REFERENCES

- [1] G.J. SIMMONS, *The prisoners' problem and the subliminal channel*, Advances in Cryptology: Proceedings of Crypto 83 (D. Chaum, ed.), Plenum Press, 1984, 51-67.
- [2] I. CHVARKOVA, S. TSIKHANENKA, V. SADAU, *Steganographic Data Embedding Security Schemes Classification*, Steganography: Digital Data Embedding Techniques, Intelligent Systems Scientific Community, 15.02.2008.
- [3] P.E. BLACK, *Fisher-Yates shuffle*, Dictionary of Algorithms and Data Structures, 19.12.2005.
- [4] <http://dotnetzip.codeplex.com/>. Retrieved: 12.02.2012.
- [5] Federal Information Processing Standards Publication 180-2: SECURE HASH STANDARD (SHS). 1.08.2002.
- [6] <http://www.codinghorror.com/blog/2007/12/the-danger-of-naivete.html>. Retrieved: 10.03.2012.
- [7] Standard ECMA-376 3rd edition, Office Open XML Formats. 06.2011.
- [8] I.J. COX, M.L. MILLER, J.A. BLOOM, J. FRIDRICH, T. KALKER, *Digital Watermarking and Steganography*, second edition, Morgan Kaufmann, 2008.

K. KACZYŃSKI

#### **Stego.docx — system ukrytej komunikacji wykorzystujący pliki .docx**

**Streszczenie.** W artykule przedstawiono propozycję systemu steganograficznego wykorzystującego pliki .docx. Przedstawiona została specyfikacja standardu Office Open XML i WordProcessingML oraz klasyfikacja systemów steganograficznych. Wykonano implementację zaproponowanego stego-systemu z wykorzystaniem środowiska Microsoft Visual Studio 2010 i .NET Framework 4.0. Analiza uzyskanych wyników potwierdziła prawidłowość działania opisanego stegosystemu.

**Słowa kluczowe:** steganografia, stegoanaliza, docx, Office, Word, Microsoft