Computing and Informatics, Vol. 38, 2019, 343–366, doi: 10.31577/cai_2019_2_343

MULTI-CARRIER STEGANOGRAPHIC ALGORITHM USING FILE FRAGMENTATION OF FAT FS

Liberios Vokorokos, Branislav Madoš, Norbert Ádám
Anton Baláž, Jaroslav Porubän, Eva Chovancová

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovak Republic
e-mail: {liberios.vokorokos, branislav.mados, norbert.adam,
anton.balaz, jaroslav.poruban, eva.chovancova}@tuke.sk

Abstract. Steganography is considered to be not only a science, but also a craft of concealing ongoing communication by hiding messages in unsuspicious cover documents, such as texts, digital images, audio and video sequences. Its essential feature is the constant search for – often exceptionally creative – possibilities of concealing information. In computers, steganography often uses secondary memory and exchangeable memory media utilising file systems. This paper deals with the current state of the issues related to information hiding by means of hard disks, being the most important source of forensic data. This paper focuses on information hiding using the File Allocation Table (FAT) file system. It also proposes a novel multi-carrier algorithm of hiding information in file fragmentation. The algorithm provides flexibility of encoding the information to be hidden and makes steps toward optimization that allows reduction of interference with the current state of the file system, represented by the statistical values of the file fragmentation parameters.

Keywords: Steganography, file system, file allocation table, FAT, fragmentation

Mathematics Subject Classification 2010: 68-R10

1 INTRODUCTION

Currently, computer system security and telecommunications' specialists are strongly focusing on encryption. This method of protection, if used as the sole solution, is considered to be insufficient in certain cases. Encrypted information attracts attention, and sometimes the existence of encrypted communication may even represent a piece of very valuable information. A solution to this problem may be the use of steganography. Steganography can be defined as the art of hiding the presence of communication by embedding secret messages into innocent, innocuos looking cover documents, such as texts, digital images, sound and video files [1].

Steganography has seen a significant development with the advent of computers, computer networks and especially the Internet; this development includes also the introduction of specific steganographic procedures, connected exclusively to the use of computers. Digital steganography employs traditional digital media, such as text, bitmap or vector graphic images, audio and video files. Excellent carriers of concealed messages are graphic image files, most frequently using the following techniques: Least Significant Bit (LSB) modification, frequency domain techniques [2, 3] and spread spectrum techniques. An overview of these techniques may be found in [4]. However, steganography is constantly seeking new, yet unused media and communication channel types, providing a possibility to expand to further territories. An example of this is the appearance of mobile phones, especially smartphones equipped with modern operating systems, such as Android or iOS. An overview on the use of steganography in smartphones is available in [5]. Very useful general overview of the development of steganography is available in [6, 7].

This paper focuses on the use of secondary storage devices and especially file systems to hide confidential information into file fragmentation. It also describes a multi-carrier algorithm which, unlike the other available algorithms, uses a set of files to conceal the information. On the contrary to the existing solutions – such as the steganographic file system proposed in [8], which adds further files to the file system and increases the fragmentation of files – the algorithm presented in this paper aims to minimize the interference with the fragmentation of files and allows to make steps to keep the statistical data concerning file fragmentation untouched, both in the whole file system and in the subset of files used to encode the information. It does not require placing fragments of two or more files into relative positions (i.e. as interlaced files) in the same part of the file system, as the solution proposed by Morkevičius et al. in [9].

In spite of the fact that the FAT file system is not a default file system in modern operating systems anymore, they still support it. Moreover, the FAT file system is still widely used on diskettes, pen drives, various memory media using flash memory chips, as well as solid-state disks (SSD). It is also utilised in numerous types of mobile devices, including mobile phones, MP3 players, cameras, embedded devices and consumer electronics devices, such as set-top boxes and multimedia players. In industry and research and development area software solutions along with specialized hardware are used with secondary storage [10, 11, 12] that often implements FAT

as file system because of relatively easy implementation. Therefore we selected the FAT32 variant of the file system to verify the possibilities of implementation of the algorithm proposed in this paper.

The rest of this paper is organised as follows:

- Section 2 includes the works related to the usage of hard disks and file systems for the purposes of steganography and cryptography, focusing on the algorithms utilising the FAT table and file fragmentation to conceal the information.
- Section 3 contains the proposal of the algorithm developed within this research, allowing hiding information into the file allocation table of the FAT file system. The algorithm encodes the information in the fragmentation of a set of files stored in the file system.
- Section 4 discusses the possibilities of making steps toward optimisation of the encoding of the confidential message with the goal to minimise changes in the file fragment parameters, as well as the possibilities of compensating these changes while keeping the original statistical values of the file fragmentation parameters after the encoding of the confidential message as much as possible.
- Finally, Section 5 lists the achieved results and shows the future research perspectives within this field.

2 RELATED WORKS

Storage devices utilising file systems – especially hard disks – are still the most important storage media used both in enterprise-grade and consumer-grade equipment. Significant amounts of confidential data – both private data and also strategically and financially valuable data of businesses and the state administration – are stored on hard disks [13]. Thus, hard disks belong to the most important sources of forensic analysis. Both the physical and logical structure of hard disks and the methods of storing information on the disk allow relatively many ways of concealing information. These approaches allow hiding the existence of the hidden data from the operating system or the users employing conventional file managers and other software working with folders and files of the file system. Therefore, special software is needed for this purpose. Some of these methods allow fully transparent disk usage, i.e. avoiding random destruction of the hidden information due to conventional usage of the media. On the other hand, other approaches may be vulnerable to random destruction of the concealed data by standard use.

Hard disk drives (HDD) and solid-state drives (SSD) may contain a so-called Host Protected Area (HPA), commonly referred to as "Hidden Protected Area". This part of the disk is not available to the user, to the BIOS, the operating system (OS) or any not-HPA-aware standard software. Therefore, the content of this area may not be read or modified using standard methods. Computer manufacturers may use the HPA to store data protected from the interference of normal users, such as diagnostic software or software used to restore the standard software installation to

its factory-state. It is possible to create software allowing access to the disk sectors of the aforementioned area, storing information in these and reading information from them. Steganography makes a good use of this [14, 15].

A further possibility of using hard disks in steganography is dividing physical storage space used to store data on the disk – i.e. sectors and clusters – into contiguous regions called partitions. By not including a set of sectors in any partition, these sectors may be used to store data, invisible to the standard access methods of the operating system. This unallocated space is called the *disk slack* or the *volume slack*. The related issues are described in detail in [16].

Since the sectors belonging to the partition may be addressed only by whole clusters and the size of the partition may be defined in such a way that when dividing the number of sectors of the partition by the number of sectors in the cluster the remainder is not an integer, there may remain some sectors at the end of the partition not addressable using standard file system methods, thus, this remaining area may be used to store the confidential information. These sectors are then commonly referred to as the *partition slack* [17].

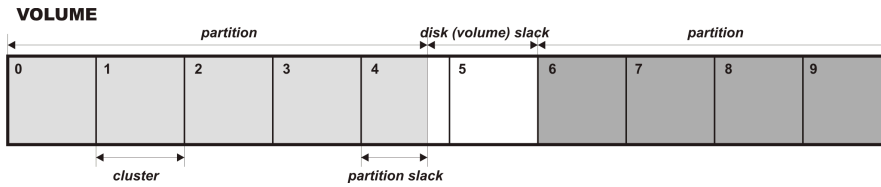


Figure 1. A volume with two partitions, showing the existence of a partition slack and disk slack and/or volume slack

Figure 1 shows a volume having two partitions. The first of them does not occupy the whole last cluster in the volume, so it cannot use it. The allocated but unusable part of the cluster is the *partition slack*. A further part of the disk not allocated to any of the partitions is the *disk slack/volume slack*.

If there are clusters in the partition not used for the storage of regular data – i.e. "empty clusters" – these are not accessible by standard means and thus may be used to store confidential information. Even if this area is large, allowing the storage of huge amounts of data – potentially up to hundreds of GB – any unused cluster may be used anytime to store regular file content and so the hidden data may be overwritten, leading to their loss.

The DOS/Windows operating system reserves the first sector of the hard disk for the Master Boot Record (MBR), which stores the information required to load the operating system and also the disk partitioning information. Even if the size of the MBR itself is small and it takes up only a single sector, the whole track, on which it is stored, is reserved and the sector containing the track cannot be addressed and used by the file system. This allows the existence of an eventually large space on the disk, usually amounting to tens of sectors, which may be used to store the concealed information [18].

Analogous to the MBR is the case of the Extended Master Boot Record (EMBR) of the extended partition. This space is commonly referred to as the *MBR slack* and the *EMBR slack*, respectively. The hidden data, stored in the MBR are protected not only from formatting, but also from the change of partition count and size on the disk.

In the file systems used by the Windows operating system, the size of a cluster may range from 512 B to 64 kB, while each file written into the file system may use one or more clusters. Since the remainder of the division of the length of the stored files in bytes and the size of the cluster is not necessarily 0, the last cluster used to store the file is being used only partly. Therefore, it may happen that one or more sectors in the last cluster are unused and so this space may be used to store the hidden information. This space is called the *file slack*.

Last sector used in the cluster to store the regular data of the file need not be fully used; also this unused space may be used to store hidden information. This space is commonly referred to as the *RAM slack*. The term *RAM slack* is a historical term – in the past, upon writing the last part of the file from the operating memory to the sector on the disk, 512 B of operating memory was copied to this space, even though some bytes had nothing in common with the content of the file. When using the *file slack* or the *RAM slack*, it is very probable that the concealed data shall be overwritten any time the size of the regular file occupying the cluster changes.

Figure 2 shows the FILE.EXT file occupying two clusters (each consisting of four sectors), while the last cluster of the file is not being fully occupied. Its first sector is not completely filled with file data and so the empty part is the *RAM slack*. The following three sectors of the cluster are fully unused – these are the *file slack*.

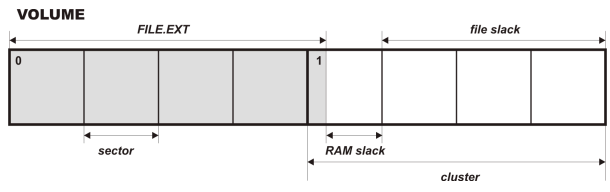


Figure 2. A file occupying two clusters, with the RAM slack and the file slack depicted

In [19], Aycock and de Castro proposed to utilise the fact that the order of files and directories displayed to the user does not correspond to their order of storage in the directories of the File Allocation Table (FAT). Permutations of their storage order allow concealing information.

As a feature of the FAT file system, the FAT table allows marking clusters in the damaged parts of the storage media as bad clusters, indicating their inappropriateness for data storage. By falsely marking fully functional clusters as bad clusters one may ensure that these will not be used by the file system. However, these are

fully functional and one may hide information in these, using a special software tool [19, 20].

A multi-carrier steganographic algorithm, storing information in file fragmentation and the relative location of these fragments in the file system was proposed by Morkevičius et al. in [9].

The utilisation of the file system as a carrier to hide information is being used not only in steganography, but also in cryptography. There are numerous implementations of cryptographic file systems available, such as the Cryptographic File System (CFS) for the UNIX operating system [21], the Transparent Cryptographic File System (TCFS) for the Linux operating system [22], the Encrypting File System (EFS) for the Microsoft Windows 2000/XP operating systems [23] or the Secure File System (SFS) for the same [24, 25]. Further cryptographic file systems include the E4M [26] and PGPDisk [27] systems.

Cryptographic file systems encode individual files and whole disk partitions, protecting user data from unwanted recovery of their content. On the other hand, the use of cryptographic functionality tells the eventual attacker that the data protected are truly confidential and important. Such a situation draws attention of the potential attackers and may inspire them to try to crack the encryption or to force the authorised user to decrypt the data under pressure. A solution to this problem may be the use of steganography, which allows masking the existence of concealed files from unauthorised users. This allows also the use of plausible deniability concept, i.e. allowing the authorised user to deny the existence of confidential data or, eventually, disclose only the existence of less important data. The attacker might not be sure and cannot prove, whether there are any further, more important data, remaining concealed.

Anderson et al. proposed in [28] a steganographic file system. In this, the user may access the requested file only by knowing its name and the appropriate password. The proposed file system inspired Van Schaik and Schmeddle to implement such a steganographic file system for the LINUX operating system [25]. In [29], Hand and Roscoe proposed an enhancement to this scheme for peer-to-peer platforms by replacing simple file replication with the Information Dispersal Algorithm (IDA).

A further implementation of a steganographic file system based on [28] was StegFS, proposed in [30]. This was an extension of the standard file system of the LINUX operating system by encryption functions allowing plausible deniability. An implementation of the steganographic file system for the Windows environment is ScramDisk, presented in [31].

A steganographic file system based on JPEG files was proposed in [32, 33]. It allowed the creation of a virtual disk, a Virtual File System (VFS), hidden in multiple cover media – images in JPEG format. The hidden content is available only to the user knowing the correct key.

The following section describes the proposed algorithm, aimed at hiding information in the FAT file system by using file fragmentation.

3 NEW MULTI-CARRIER FILE FRAGMENTATION BASED STEGANOGRAPHIC ALGORITHM

For the purposes of the algorithm, the information stored in the FAT system (files) form the set V. The total file count is x, thus $x = |V|$ (cardinality). So, the following applies:

$$V = \{F_o, F_1, \ldots, F_{x-1}\}. \tag{1}$$

On hard disks of real-life personal computers (with an installed operating system, software and user data files), the size of this set amounts to hundreds of thousands.

The files stored in the FAT are separated into fragments. A fragment consists of data of a specific file stored in a contiguous sequence of clusters allocated so that in the FAT table the record of the cluster at the address n points to the next cluster at the address $n + 1$. An exception to this rule is the last cluster of the fragment at the address n, pointing to the first cluster of the next fragment, which must not be located at the address $n + 1$. A fragment may be limited from above and from below by unallocated clusters or clusters allocated by another fragment. It may be limited also by other fragments of the same file, while the fragment is not a subset of another fragment. Thus, we may define the set A, formed by all fragments of all files stored in the file system.

$$A = \{f_o, f_1, \ldots, f_{y-1}\}. \tag{2}$$

Obviously, $y = |A|$, while the value depends on the current file fragmentation and is variable. From below, it is limited by the value of x – the number of files stored in the file system – because if a file is not fragmented, we may assume that it consists of a single fragment. If all files were unfragmented, the number of fragments in the file system would be equal to the number of files. In theory, the upper limit of the total fragment count is the sum of clusters allocated to the individual files of the set V, being an extreme state, when each fragment of each file consists of a single cluster. A limiting factor is also the maximum amount of clusters available to the file system. Each fragment may be part only of a single file.

In the proposed algorithm, each $f_z \in A : z \in \langle 0, y - 1\rangle$, $z \in \mathbb{N}^0$ fragment is an ordered set of fragment parameters:

$$f_z = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\} \tag{3}$$

where

- α_1 is the address of the first cluster of the fragment in the FAT table,
- α_2 is the length of the fragment in clusters,
- α_3 is the address of the last cluster of the fragment in the FAT table,
- α_4 is the length to the next fragment of the file in clusters,
- α_5 is the address of the first cluster of the next fragment of the file.

In case of the last fragment in the sequence of fragments of the given file, the parameters α_4 and α_5 are set to 0. The set of fragment parameters is not complete, currently it is only a proof of concept.

The algorithm defines the O set of operations

$$O = \{\beta_1(x), \beta_2(x), \beta_3(x), \beta_4(x), \beta_5(x), \beta_6(x)\} \tag{4}$$

where

- $\beta_1(x)$ is writing (reading) 0, if x is even and 1 if x is odd,
- $\beta_2(x)$ is writing (reading) 1, if x is even and 0 if x is odd,
- $\beta_3(x)$ is a shift to the next file in the file set, if x is even and to the previous file in the file set, if x is odd,
- $\beta_4(x)$ is a shift to the previous file in the file set, if x is even and to the next file in the file set, if x is odd,
- $\beta_5(x)$ is a shift ahead in the file by 1 fragment, if x is even and by 2 fragments, if x is odd,
- $\beta_6(x)$ is a shift ahead in the file by 2 fragments, if x is even and by 1 fragment, if x is odd.

Some of the operations are aimed at writing/reading bits of the concealed information into/from the fragment parameters. Other operations serve the purpose of determining the position shifts between the files and fragments at the time of performing the respective steps of the algorithm for the purpose of writing and/or reading confidential information. The set of operations – similarly to the set of fragment parameters – is not complete and currently still a proof of concept.

For the purposes of the proposed algorithm, the Carthesian product of the sets $O \times f_z$ where $z \in \langle 0, y-1\rangle$, $z \in \mathbb{N}^0$ allows the creation of the R set of encoding rules. Since neither the set of operations, nor the set of fragment parameters are not fully defined, similarly, the set R is not final either and currently still the proof of concept:

$$R = \{\beta_1(\alpha_1), \beta_1(\alpha_2), \beta_1(\alpha_3), \ldots, \beta_6(\alpha_3), \beta_6(\alpha_4), \beta_6(\alpha_5)\}. \tag{5}$$

The encoding rule $\beta_1(\alpha_1)$ may be then interpreted as the application of the operation $\beta_1(x) \in O$, where x is the $\alpha_1 \ \in f_z : z \in \langle 0, y-1\rangle$, $z \in \mathbb{N}^0$.

The interpretation of the encoding rules is different when hiding and extracting confidential information. For example, to the encoding rule $\beta_1(\alpha_1)$ the following applies:

> When writing confidential information, this encoding rule must ensure that the 0 bit is written – after the application of the encoding rule – by setting the address of the first cluster of the fragment used for writing the bit even, and that the 1 bit is written – after the application of the encoding rule – by

setting the address of the first cluster of the fragment used for writing the bit to an odd value.

If the current value of this fragment parameter does not correspond to the value required by the rule, the value of this fragment parameter shall be changed to the appropriate value.

When reading the hidden information, the application of this encoding rule does not change the value of the given fragment parameter. Depending on whether the fragment parameter is even/odd, a 0/1 value is read as the part of the hidden message.

3.1 Information Hiding

To apply the algorithm hiding the confidential message M represented by a final stream of bits into the fragmentation of the files stored in the file system, we have to know the sets V, A, $\forall F_n \in V$ and $\forall f_z \in A$, characterising the current state of the specific file system, into which the information shall be hidden. Moreover, also the above sets O and R have to be known and the input parameters of information hiding have to be known as the ordered set H:

$$H = \{V', R', f_{(s)}\} \tag{6}$$

where

- V' is an ordered set of files and $V' \subset V$,
- R' is an ordered set of encoding rules and $R' \subset R$,
- f_s is a fragment of a file, for which $\exists F_n \in V' : f_s \in F_n$.

Thus, from the V set of files we have to select certain files and create the subset V', into which the hidden information shall be encoded and order them according to the required order. Then, the R' subset of information encoding rules has to be selected from the R set of rules; this subset has to be ordered, too. Finally, the f_s starting fragment has to be selected, from which the encoding shall be performed. This fragment must belong to one of the selected files, though it need not be the first fragment of the given file. The information hiding parameters included in the set H form a steganographic key, used to store the concealed message – it has to be known also to extract the hidden information.

The information hiding algorithm consists of the following steps:

Step 1. Files of the set V', into which the confidential information shall be hidden, have to be ordered by their selected feature or by any permutation of their order; this order is one of the elements of the steganographic key. After ordering the files, each file shall be assigned an identifier: $fi \in \langle 0; x' - 1\rangle$, where $x' = |V'|$.

Step 2. For $\forall F_n \in V'$, its file fragments have to be ordered in the order of accessing them when reading the file. Fragment f_n is the successor of fragment f_m, if

$\exists\alpha_5 \in f_m \wedge \exists\alpha_1 \in f_n : \alpha_5 = \alpha_1$. Next, to each fragment, an ordinal number has to be assigned: $fr \in \langle 0; d-1\rangle$, where d is the number of fragments of the file F_n.

Step 3. In this step, the identifier fi of the file containing the starting fragment f_s and the fr ordinal number of the fragment in the corresponding file is found; these are registered as fi_a and fr_a, if $fi_a = fi$ and $fr_a = fr$ of starting fragment f_s. The variables fi_a and fr_a will later serve as pointers to the location of the currently processed fragment.

Step 4. In this step, the pointer m_a pointing at the current bit of the string M to be encoded is set to 0. Thus, it points to the first bit of the string M.

Step 5. To the current fragment, i.e. the one, to which the pointers fi_a and fr_a point to, we apply all encoding rules of the set R' – being part of the key – one by one, in the specified order (this order is important, because one can encode multiple bits of the concealed information using multiple rules into a single fragment, therefore the order of their encoding into the fragment must be known). If the specific rule serves for writing a bit into the corresponding fragment feature, the bit shall be written and the pointer m_a shall be incremented by 1. If, upon application of any of the rules, the last bit of the confidential message is encoded, the execution of the algorithm ends.

Step 6. The application of rules in Step 5, leading to the change of some fragment parameters requires an additional compensation of this change in some other fragment of the particular file to make sure that the information stored in the file is not corrupted and to prevent the corruption of the part of the hidden message M, already encoded in the set V'. An example may be a change of the length of the specific fragment – a lengthening by one cluster requires a shortening of another fragment of the same file by a cluster. Subsequently, $\forall f_z \in A$ which is representing changed fragment must be updated to store all changes of the file fragment parameters.

Step 7. If no rule applied in Step 5 contains information as to what shift of the current file pointer – fi_a – has to be performed, this pointer shall be incremented by 1 (shift to the next file in the set V'). If the current file pointer is set to the last file, the execution continues with the first file, thus if $fi_a = x' - 1$ then $fi_a = 0$ shall apply and vice versa: if $fi_a = 0$ and a shift backwards by a file is required, $fi_a = x' - 1$ shall be set. (This principle is applied also to the shifts within the files, performed by applying the rules specified in Step 5).

Step 8. If no rule applied in Step 5 contains any information as to what shift of the pointer to the current fragment – fr_a – has to be performed, this pointer shall be incremented by 1 (shift to the next fragment).

Step 9. Continue with Step 5.

The application of the rule storing the value of the bit of the confidential message into the corresponding fragment parameter requires setting the appropriate value

of the parameter. For example, if it is the address of the first cluster of the given fragment and it should be even but it is not, the fragment has to be shifted by a cluster towards the lower address or a higher address, depending on which is better, considering the current situation in the file system. The advantage of this approach is that – in up to 50 % of the cases – the specific fragment parameter contains the appropriate value in the given state of the file system, so it need not be changed at all.

Upon the application of the rules determining the file containing the fragment, in which the encoding performed in the next application of Step 5 of the algorithm shall happen, i.e. the rules modifying the pointer fi_a and upon the application of the rules determining the fragments count of the shift, i.e. the rules modifying the pointer fr_a, there are two alternatives. The first is to accept the current setting of the fragment of the parameter used by the specific encoding rule (i.e., leave the parameter unchanged); the second alternative is to change it (from even to odd and vice versa). The choice of accepting or modifying the parameter shall then modify the further execution of the encoding procedure. This freedom is the strength of the proposed algorithm – as far as information hiding is concerned – and it also provides sufficient flexibility to perform the least possible intervention into the current state file fragmentation during the encoding.

3.1.1 Usage Example

The following example shows a FAT file system containing four files, that are forming set V:

$$V = \{F_0, F_1, F_2, F_3\}.$$

The file system contains 16 file fragments

$$A = \{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15}\}.$$

The structure of the four respective files, which are constructed by the use of fragments from the set A, is defined by the sets F_0 to F_3:

$$\begin{aligned} F_0 &= \{f_0, f_1, f_2, f_3\}, \\ F_1 &= \{f_4, f_5, f_6, f_7\}, \\ F_2 &= \{f_8, f_9, f_{10}, f_{11}\}, \\ F_3 &= \{f_{12}, f_{13}, f_{14}, f_{15}\}. \end{aligned}$$

The parameters of the respective fragments are defined by the sets f_0 to f_{15}

$$f_0 = \{2, 4, 5, 1, 7\}, \qquad f_8 = \{55, 3, 57, 1, 59\},$$
$$f_1 = \{7, 3, 9, 1, 11\}, \qquad f_9 = \{59, 2, 60, 2, 63\},$$
$$f_2 = \{11, 2, 12, 5, 18\}, \qquad f_{10} = \{63, 1, 63, 3, 67\},$$
$$f_3 = \{18, 4, 21, 0, 0\}, \qquad f_{11} = \{67, 4, 70, 0, 0\},$$
$$f_4 = \{28, 3, 30, 1, 32\}, \qquad f_{12} = \{48, 2, 49, 1, 51\},$$
$$f_5 = \{32, 4, 35, 2, 38\}, \qquad f_{13} = \{51, 3, 53, 24, 78\},$$
$$f_6 = \{38, 2, 39, 3, 43\}, \qquad f_{14} = \{78, 4, 81, 3, 85\},$$
$$f_7 = \{43, 5, 47, 0, 0\}, \qquad f_{15} = \{85, 2, 86, 0, 0\}.$$

The binary string to be stored consists of three bits: $M =$ "011". Next, the set V', R' and the fragment $f_{(s)}$ have to be selected. The V' set of files, used to store the hidden information, was selected as F_0, F_1, F_2 in the aforementioned order:

$$V' = \{F_0, F_1, F_2\}.$$

Two encoding rules – $\beta_1(\alpha_1)$ and $\beta_3(\alpha_2)$ – were selected, forming the set R' in the aforementioned order:

$$R' = \{\beta_1(\alpha_1), \beta_3(\alpha_2)\}.$$

The selected encoding rule $\beta_1(\alpha_1)$ ensures that the bits of the confidential message shall be written to the position of the starting cluster of the fragment as follows: will it be stored at an even address in the FAT table, the bit of the encoded message shall be set to 0; will the starting cluster of the fragment be stored at an odd address in the FAT table, the bit of the encoded message shall be set to 1. According to rule $\beta_3(\alpha_2)$, if the length of the fragment is even, the next fragment should be in the file being at the next position in the V' set of files; if the length of the fragment is odd, the next fragment should be stored in the file being at the previous position in the V' set of files. No rule of the set R' specifies how many fragments should the algorithm jump when shifting after the encoding step, therefore we will select the basic shift as a shift by one fragment ahead.

As the starting fragment f_s we selected fragment f_5, being the second fragment in file F_1.

Figure 3 shows the current state of the file system. It contains four files – F_0 to F_3 – while each of the files is fragmented into four fragments. Each line contains fragments belonging to the particular file in the order of appearance in the file. The length of the rectangle representing the fragment shows its length in clusters, with the value printed just below it. Within the fragment, the figure shows its first cluster with its address in the file allocation table (FAT).

Figure 4 shows the situation after encoding the confidential message. The fragments affected by encoding – either the confidential message was encoded in their

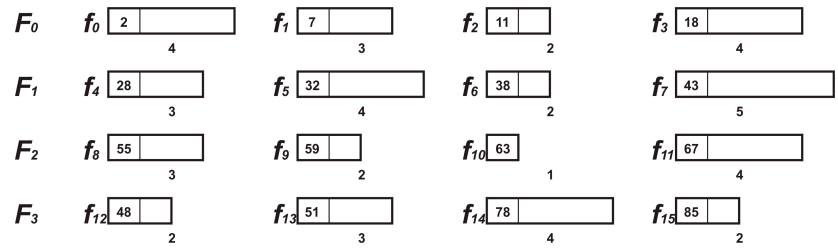


Figure 3. The current state of file fragmentation in the file system, before encoding the confidential message

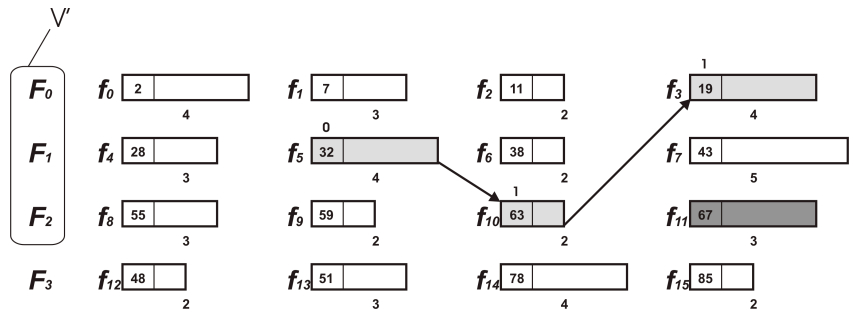


Figure 4. The confidential message $M =$ "011" stored in the fragmentation of files forming set V'

parameters, their position or their length was changed in the file system – have been set in grey.

Encoding started in fragment f_5, its first cluster is stored as an even address and the application of rule $\beta_1(\alpha_1)$ in Step 5 of the algorithm encoded the first bit of the message – bit 0 – into the fragment. The parameter of the fragment was set correctly (i.e. even), no change was needed. The following parameter of fragment f_5, used for the purpose of encoding, was the fragment length. The current fragment length is even, and the decision was taken not to change it. By applying rule $\beta_3(\alpha_2)$, the next fragment to be used for encoding was fragment f_{10}, due to the shift to the next file and next fragment.

The application of Step 5 of the algorithm to fragment f_{10} required the application of rule $\beta_1(\alpha_1)$, which encoded a further bit of the confidential message – bit 1. So it was necessary to start the fragment on an odd address in the FAT. The current setting of the parameter met the requirement, no change was necessary. We also applied rule $\beta_3(\alpha_2)$ and decided that the current value of the parameter shall be modified from odd to even. So a shift to the subsequent file was coded, however, this caused that in the next step, file F_0 became the current file (containing fragment f_3), since file F_2 was the last in the list of files in set V'. Since fragment f_{10}

was prolonged by one cluster, fragment f_{11} of file F_2 had to be shortened by one cluster to maintain the file length in clusters.

The application of Step 5 of the algorithm to fragment f_3 required the application of rule $\beta_1(\alpha_1)$. This stored another bit of the message into the fragment. This was bit 1. This required a change of the address of the first cluster of fragment f_3 to an odd value and to shift the whole fragment by a cluster. Since the last bit of the confidential message was written, the execution of the algorithm ended.

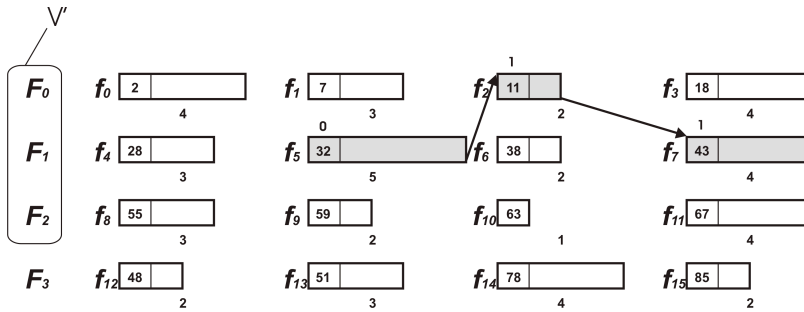


Figure 5. The confidential message $M =$ "011" stored in the fragmentation of files forming set V'

Figure 5 shows an alternative encoding of the confidential message into the fragments of files forming set V'. The length of fragment f_5 was modified to be able to continue with fragment f_2, the parameters of which were left unchanged. The next fragment, into which the information was encoded, was fragment f_7.

Its parameters were left unchanged due to the encoding of the confidential message; however, it had to be shortened by a cluster to maintain the length of file F_4, since its fragment f_5 became longer by a cluster.

3.2 Information Extraction

Information extraction is analogous to information hiding. However, one needs not know the total current state of the file system, i.e. the sets V, A, $\forall F_n \in V$ and $\forall f_z \in A$. It is enough to know the steganographic key H, i.e. the sets V', R', f_s and also $\forall F_n \in V'$ a $\forall f_z \in F_n : F_n \in V'$. So it is necessary to know the set of files, into which the confidential information was hidden, the fragments forming these files and their specific order. We also need to know the set of rules used to encode the information and also their order; moreover, one has to know the starting fragment, used to implement the encoding.

When extracting the hidden information, an algorithm with steps identical to the steps of the information hiding algorithm shall be used. When applying the rules of Step 5, we only read the bits of the message M by applying the corresponding rules and performing jumps to their respective files and fragments by applying the correct rules, without performing any changes to the file system. When reading the

message, Step 6 is not being performed. We repeatedly apply Steps 5, 7, 8 and 9, until the last bit of the hidden message M is read.

4 RESULTS AND DISCUSSION

One of the goals connected with the algorithm was to provide a flexible way of encoding the information into the parameters of the fragments of files stored in the file system. As the example in Section 3.1.1 shows, the algorithm met this requirement when it allowed encoding alternatives. Two of these are depicted in Figures 4 and 5. This allows comparison of the individual encoding alternatives, considering the number of changes of the respective fragment parameters and finding the optimum encoding, which would modify the least possible individual file fragment parameters in the current state of the file system, the specified steganographic key and the confidential message M, i.e., perform only minimal file fragmentation parameter changes.

In the example in Section 3.1.1, a single rule $\beta_3(\alpha_2)$, allowing modification of the encoding before each application in Step 5 of the proposed algorithm, was used. Each application of this rule allows the existence of two alternatives of the following encoding procedure – in case of a message of n bits it means 2^{n-1} encoding alternatives ($n-1$, since the last iteration of the application of the steps ends in Step 5, by encoding the last bit of the message, before the last application of the rule $\beta_3(\alpha_2)$. In the specific example, the number of alternative encodings is 2^2, i.e. 4, since message M is 3 bits long.

If the complete steganographic key is not specified – such as the V' set of files, on which the encoding should be implemented, is missing – during the search for the optimum encoding, the search for this set may be included in the search for the optimum encoding. In the example specified in Section 3.1.1 we may search for an appropriate permutation of the three files selected from the overall total file count (four) stored in the file system. The total number of applicable alternatives is then $(x).(x-1).(x-2)$, where $x = |V|$. Thus, the specified example allows 24 alternatives.

The use of rule $\beta_3(\alpha_2)$ without specifying set V' allows – in the example of Section 3.1.1 – a total of $(x).(x-1).(x-2).2^{n-1}$, i.e. $24.4 = 96$ encoding alternatives. To each of these alternatives, a natural number representing the number of changes to be made to the file fragment parameters to encode the specified string into the set of files using the given alternative may be assigned. Subsequently, the alternative with the lowest change count may be selected. Alternative in Figure 4 required three fragment parameter changes – in two cases, the fragment length in clusters changed and in one case, the fragment position shifted by a cluster. The alternative specified in Figure 5 required two parameter changes, with two fragment length modifications. The alternative specified in Figure 5 involved fewer file fragmentation parameter changes, so its use may be considered more advantageous.

If, during the search for the optimum encoding alternative, not even the starting fragment of the encoding is specified, the total count of encoding alternatives N_{alt} amounts to the following:

$$N_{alt} = \frac{x!}{(x - x')!} . 2^{m^{(n-1)}} . y \tag{7}$$

where

- $x = |V|$ is the number of files in the file system,
- $x' = |V'|$ is the number of files, into which the information shall be encoded,
- n is the bit count of the M confidential message,
- m is the number of rules used to modify the encoding procedure,
- y is the number of fragments of set V', where the message encoding may start.

From the above it is evident that for real-life file systems containing tens or hundreds of thousands of files and the minimum length of the confidential message being tens of bits, the search for the optimum encoding alternative is an exceptionally computing-intensive task, so it is rather a theoretical concept than a useful procedure.

However, one may search effectively for suboptimal solutions, e.g. when encoding the confidential information into the specific fragment, alternatives for defined number of applications of Steps 5 to 9 of the algorithm shall be searched for selecting the optimum encoding alternative for the given fragment and only this search window is considered. This limits the computing requirements of the procedure and allows controlling it by setting the size of the aforementioned search window.

A further design ambition related to the algorithm was to allow the least possible interference with the statistical parameters of file fragmentation in the file system and thus lower the chances of recognition of the use of this algorithm by using steganalytic methods. When encoding confidential information in the specific example set out in Section 3.1.1, we used fragment parameters α_1 and α_2, i.e. the position of the first cluster of the fragment and fragment length, checking them for being even or odd. We may also monitor the statistical values of these parameters for the set V of all files, as well as the set of files used to store the confidential information, i.e. set V'. Table 1 summarises these parameters for the situation before the encoding (Original state) of the confidential information, following the encoding by the alternative specified in Figure 4 and following the encoding by the alternative specified in Figure 5, respectively.

As it is evident from Table 1, following the encoding of the confidential information using the alternative specified in Figure 4, the statistical values of parameter α_1 change in comparison to the original state, when the number of even settings of the parameter decreases by 6.25 % (from 43.75 % to 37.50 %), and the number of odd settings of the parameter increases by 6.25 % (from 56.25 % to 62.50 %) in the set V.

		Parameter α_1				**Parameter α_2**			
		Even		**Odd**		**Even**		**Odd**	
		No.	**%**	**No.**	**%**	**No.**	**%**	**No.**	**%**
Original state	V	7	43.75	9	56.25	10	62.50	6	37.50
	V'	5	41.67	7	58.33	7	58.33	5	41.67
Encoding I	V	6	37.50	10	62.50	10	62.50	6	37.50
(Figure 4)	V'	4	33.33	8	66.67	7	58.33	5	41.67
Encoding II	V	7	43.75	9	56.25	10	62.50	6	37.50
(Figure 5)	V'	5	41.67	7	58.33	7	58.33	5	41.67

Table 1. Statistical values of parameters α_1 and α_2 for the sets V and V' before encoding confidential information and after the encoding using the two alternatives

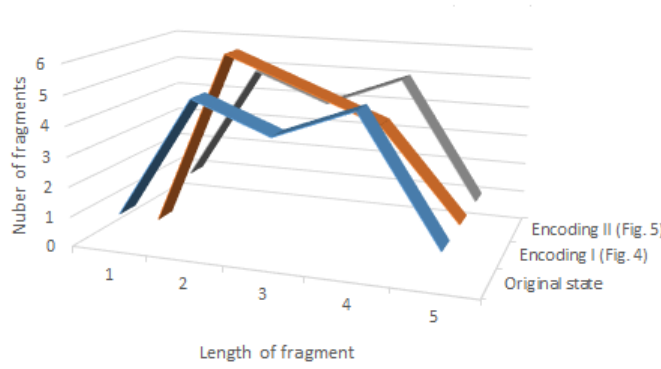
The number of even settings of the parameter decreased by 8.34 % (from 41.67 % to 33.3 %), and the number of odd settings of the parameter increased by 8.34 % (from 58.33 % to 66.67 %) in the set V'.

The statistical values of parameter α_2 remained unchanged, both in set V and also in set V'. In this aspect, the alternative specified in Figure 5 is more advantageous, since the statistical values of parameter α_1 and parameter α_2 remained unchanged, both in the case of set V as well as in the case of set V'.

Analogous to the previous case of searching the optimum encoding considering the minimum amount of fragment parameter changes, in this case we could assign each of the alternatives a value showing the degree of equality of the statistical values of the monitored parameters before and after the encoding and select the optimum alternative, thus the one with the highest degree of equality. However, the amount of encoding alternatives is identical to the previous case of searching for the optimum encoding, thus also the computing requirements of this procedure are beyond the limits of practical use.

Nevertheless, we may introduce a procedure allowing the compensation of changes to the respective fragment parameters during their execution in Step 5 of the proposed algorithm. If in the current Step 5 of the encoding a fragment with the identifiers fi_a and fr_a is selected, i.e., the fragment belonging to the file fi_a, being the at position fr_a in the file, and some of its parameters have to be changed, e.g. the parameter α_2 – its length – has to be changed from even to odd (for instance), this change may be compensated by changing the parameter from odd to even in any other fragment, to which it applies that its $fr > fr_a$ and fi may be of any permissible value, so the fragment may be part of any file of the set V'. If we add the condition of $fi = fi_a$, thus the compensating fragment (the parameter change of which serves as the compensation of the change of the fragment parameter, into which the information is encoded) and the fragment used for encoding must be from the same file, the changes of statistical values of the respective parameters shall be compensated in each individual file from the set V'.

When evaluating the statistical values of the fragment parameters, we may also consider their absolute length in clusters. The state before encoding the confiden-



Length of fragment	1	2	3	4	5
Original state	1	5	4	5	1
Encoding I (Fig. 4)	0	6	5	4	1
Encoding II (Fig. 5)	1	5	4	5	1

Original state Encoding I (Fig. 4) Encoding II (Fig. 5)

Figure 6. Distribution of fragments by their length values before encoding the confidential information and after the encoding using the first and the second alternative

tial message and after its encoding using the example in Section 3.1.1 is shown in Figure 6.

It is evident that following the encoding depicted in Figure 5, the lengths of the individual fragments are compensated to make them identical with the values they had before the encoding of the confidential information (original state). One could design an algorithm taking also the distribution of fragment lengths into account and perform compensations of these fragment parameter changes during the encoding of the information. However, practical testing performed on real-life secondary memory devices using the FAT32 file system showed that the fragment lengths were from a quite large interval – $\langle 1; 1\,862 \rangle$ – and it was not always possible to compensate change of length of one fragment by changing the length of another fragment in the way that statistical values of the set of fragments stays unchanged. Fragment length changes amounting to a single cluster do not represent significant changes in the file system as a whole, therefore we refrained from the effort to compensate the changes of this parameter.

5 CONCLUSIONS

In the introductory part, this paper analysed the current state of using secondary data storage devices, especially hard disks and the FAT file system in steganography.

Then, an algorithm using the fragmentation of a set of files stored in the FAT file system as a carrier of storing confidential information was proposed. As a proof of the concept, a set of fragment parameters, a set of available operations and a set of encoding rules were specified in the algorithm. As part of further research, these sets shall be completed with the aim to find the optimum elements. The paper also included a usage example for the information hiding algorithm and it also evaluated the computation requirements of finding the optimum encoding of the confidential information, as well as the compensation possibilities of the algorithm, considering the changes of the respective fragment parameters of the files stored within the file system.

The advantage of this algorithm is the flexibility of encoding information into a set of files, which significantly increases the complexity of extracting the confidential information using brute force attacks and allows the application of plausible deniability. The algorithm does not store any additional information on the disk and allows finding alternative encodings of the confidential information, which decreases the number of fragment parameter changes during the encoding procedure and simultaneously allows compensation of these changes to minimise the changes to the statistical values of the file fragment parameters.

A disadvantage of the algorithm is, similarly to all algorithms aiming at storing information in the fragmentation of files, the loss of information upon defragmenting the file system. This risk is limited by multiple factors. First of all, the user may forbid the process of defragmenting. The advantage of using the FAT32 file system is the possibility to use it on secondary memory media, such as pen drives, various kinds of memory cards and SSD devices, where defragmenting is suppressed due to the technology of the memory chips used. Many devices, such as mobile multimedia players, digital cameras, set-top boxes and other devices – using even traditional hard disks – often use firmware incapable of defragmenting.

Future research should focus on the development of methods allowing search of suboptimal solutions of encoding confidential information minimising the amount of interference with the file fragment parameters and procedures allowing the compensation of these changes with acceptable algorithmic complexity.

Acknowledgements

This work was supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10 and KEGA 008TUKE-4/2013 Microlearning environment for education of information security specialists. The projects are being solved at the Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice. This work was supported by KEGA Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic under Grant No. 077TUKE-4/2015 "Promoting the interconnection of Computer and Software Engineering using the KPIkit". This support is very gratefully acknowledged.

REFERENCES

[1] Nag, A.—Singh, J. P.—Khan, S.—Ghosh, S.—Biswas, S.—Sarkar, D.—Sarkar, P. P.: A Weighted Location Based LSB Image Steganography Technique. In: Abraham, A., Lloret Mauri, J., Buford, J. F., Suzuki, J., Thampi, S. M. (Eds.): Advances in Computing and Communications (ACC 2011). Springer, Berlin, Heidelberg, Comunications in Computer and Information Science, Vol. 191, 2011, pp. 620–627, doi: 10.1007/978-3-642-22714-1_64. ISBN: 978-3-642-22713-4 (print), ISBN: 978-3-642-22714-1 (online).

[2] Katzenbeisser, S.—Petitcolas, F. A. P.: Information Hiding Techniques for Steganography and Digital Watermarking. Artech House Publishers, Norwood, Massachusetts, USA, 2000. ISBN: 1-58053-035-4.

[3] Nag, A.—Biswas, S.—Sarkar, D.—Sarkar, P. P.: A Novel Technique for Image Steganography Based on Block-DCT and Huffman Encoding. International Journal of Computer Science and Information Technology, Vol. 2, 2010, No. 3, pp. 103–112, doi: 10.5121/ijcsit.2010.2308.

[4] Zielińska, E.—Mazurczyk, W.—Szczypiorski, K.: Trends in Steganography. Communications of the ACM, Vol. 57, 2014, No. 3, pp. 86–95, doi: 10.1145/2566590.2566610.

[5] Nag, A.—Biswas, S.—Sarkar, D.—Sarkar, P. P.: A Novel Technique for Image Steganography Based on DWT and Huffman Encoding. International Journal of Computer Science and Security, Vol. 4, 2011, No. 6, pp. 561–570.

[6] Mazurczyk, W.—Caviglione, L.: Steganography in Modern Smartphones and Mitigation Techniques. IEEE Communications Surveys and Tutorials, Vol. 17, 2015, No. 1, pp. 334–357, doi: 10.1109/COMST.2014.2350994.

[7] Petitcolas, F. A. P.—Anderson, R. J.—Kuhn, M. G.: Information Hiding – A Survey. Proceedings of the IEEE, Vol. 87, 1999, No. 7, pp. 1062–1078, doi: 10.1109/5.771065.

[8] Anderson, R.—Needham, R.—Shamir, A.: The Steganographic File System. In: Aucsmith, D. (Ed.): Information Hiding (IH 1998). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1525, 1998, pp. 73–82, doi: 10.1007/3-540-49380-8_6.

[9] Morkevičius, N.—Petraitis, G.—Venčkauskas, A.—Čeponis, J.: Covert Channel for Cluster-Based File Systems Using Multiple Cover Files. Information Technology and Control, Vol. 42, 2013, No. 3, pp. 260–267, doi: 10.5755/j01.itc.42.3.3328. ISSN: 1392-124X (print), ISSN: 2335–884X (online).

[10] Kainz, O.—Jakab, F.—Michalko, M.—Feciľak, P.: Detection of Persons and Height Estimation in Video Sequence. International Journal of Engineering Sciences and Research Technology, Vol. 5, 2016, No. 3, pp. 603–609, doi: 10.5281/zenodo.48321. ISSN: 2277-9655.

[11] Ševčík, J.—Kainz, O.—Feciľak, P.—Jakab, F.: System for EKG Monitoring: Solution Based on Arduino Microcontroller. International Journal of Advanced Research in Artificial Intelligence (IJARAI), Vol. 4, 2015, No. 9, pp. 22–25. ISSN: 2165-4069.

[12] Kovalčík, M.—Feciľak, P.—Jakab, F.—Dudiak, J.—Kolcun, M.: Cost-Effective Smart Metering System for the Power Consumption Analysis of Household. International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 5, 2014, No. 8, pp. 135–144, doi: 10.14569/IJACSA.2014.050821. ISSN: 2156-5570.

[13] Cheddad, A.—Condell, J.—Curran, K.—McKevitt, P.: Digital Image Steganography: Survey and Analysis of Current Methods. Signal Processing, Vol. 90, 2010, No. 3, pp. 727–752, doi: 10.1016/j.sigpro.2009.08.010.

[14] Sutherland, I.—Davies, G.—Blyth, A.: Malware and Steganography in Hard Disk Firmware. Journal in Computer Virology, Vol. 7, 2011, No. 3, pp. 215–219, doi: 10.1007/s11416-010-0149-x.

[15] Sutherland, I.—Davies, G.—Pringle, N.—Blyth, A.: The Impact of Hard Disk Firmware Steganography on Computer Forensics. Journal of Digital Forensics, Security and Law, Vol. 4, 2009, No. 2, pp. 73–84, doi: 10.15394/jdfsl.2009.1059. ISSN: 1558-7215 (print), ISSN: 1558-7223 (online).

[16] Gupta, M. R.—Hoeschele, M. D.—Rogers, M. K.: Hidden Disk Areas: HPA and DCO. International Journal of Digital Evidence, Vol. 5, 2006, No. 1, pp. 1–8.

[17] Carrier, B.: File System Forensic Analysis. Addison Wesley Professional, 2005, 600 pp., ISBN: 0-32-126817-2.

[18] Balan, C.—Vidyadharan, D. S.—Dija, S.—Thomas, K. L.: Combating Information Hiding Using Forensic Methodology. Proceedings of the Sixth International Workshop on Digital Forensics and Incident Analysis (WDFIA 2011), 2011, Kingston University, London, UK, pp. 69–75. ISBN: 978-1-84102-285-7.

[19] Aycock, J.—de Castro, D. M. N.: Permutation Steganography in FAT Filesystems. In: Shi, Y. (Ed.): Transactions on Data Hiding and Multimedia Security X. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 8948, 2015, pp. 92–105, doi: 10.1007/978-3-662-46739-8_6.

[20] Liu, S.-F.—Pei, S.—Huang, X.-Y.—Tian, L.: File Hiding Based on FAT File System. Proceedings of the 2009 IEEE International Symposium on IT in Medicine and Education, Jinan, China, Vol. 1, 2009, pp. 1198–1201, doi: 10.1109/ITIME.2009.5236280.

[21] Sheetz, M.: Computer Forensics: An Essential Guide for Accountants, Lawyers, and Managers. John Wiley and Sons, New Jersey, USA, 2015, 176 pp., ISBN: 978-0-471-78932-1.

[22] Blaze, M.: A Cryptographic File System for Unix. Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS '93), 1993, pp. 9–16, doi: 10.1145/168588.168590.

[23] Persiano, G. et al.: TCFS – Transparent Cryptographic File System. DIA, Università Degli Studi Di Salerno, Italy.

[24] Encrypting File System for Windows 2000, Microsoft Windows 2000 White Paper, Microsoft Corporation, 1998.

[25] Gutmann, P.: University of Auckland, New Zealand. The secure FileSystem (SFS) for DOS/Windows. `http://www.cs.auckland.ac.nz/pgut001/sfs/index.html`, September 1996.

[26] E4M Disk Encryption, online: http://www.e4m.net.

[27] PGPDisk, online: http://www.pgpi.org/products/pgpdisk/.

[28] Hughes, J. P.—Feist, C. J.: Architecture of the Secure File System. 2001 Eighteenth IEEE Symposium on Mass Storage Systems and Technologies, San Diego, CA, USA, 2001, pp. 277–290, doi: 10.1109/MSS.2001.10020.

[29] Hand, S.—Roscoe, T.: Mnemosyne: Peer-to-Peer Steganographic Storage. In: Druschel, P., Kaashoek, F., Rowstron, A. (Eds.): Peer-to-Peer Systems (IPTPS '02). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2429, 2002, pp. 130–140, doi: 10.1007/3-540-45748-8_13.

[30] Van Schaik, C.—Schmeddle, P.: A Steganographic File System Implementation for Linux. University of Cape Town, South Africa, October 1998.

[31] McDonald, A. D.—Kuhn, M. G.: StegFS: A Steganographic File System for Linux. In: Pfitzmann, A. (Ed.): Information Hiding (IH 1999). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1768, 2000, pp. 463–477, doi: 10.1007/10719724_32.

[32] Jókay, M.—Košdy, M.: Steganographic File System Based on JPEG Files. Tatra Mountains Mathematical Publications, Vol. 57, 2013, No. 1, pp. 65–83, doi: 10.2478/tmmp-2013-0036. ISSN: 1210-3195.

[33] Jókay, M.—Košdy, M.—Čavoj, M.: Steganographic File System Embedded in Static Images. Central European Conference on Cryptology 2013, Telč, Czech Republic, 2013, pp. 76.

Liberios Vokorokos graduated (M.Sc.) with honours at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice in 1991. He defended his Ph.D. in the field of programming device and systems in 2000, his thesis title was "Diagnosis of Compound Systems Using the Data Flow Applications". He was appointed Professor for computer science and informatics in 2005. Since 1995 he has been working as an educationist at the Department of Computers and Informatics. His scientific research is focused on parallel computers of the data flow type. In addition, he also investigates the questions related to the diagnostics of complex systems. Currently he is Dean of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice. His other professional interests include the membership in the Advisory Committee for Informatization at the Faculty and Advisory Board for the Development and Informatization at the Technical University of Košice.

Branislav Madoš graduated (Ing.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2006. He defended his Ph.D. in the field of computers and computer systems in 2009, his thesis title was "Specialized Architecture of Data Flow Computer". Since 2010 he has been working as Assistant Professor at the Department of Computers and Informatics. His scientific research is focused on the parallel computer architectures and architectures of computers with data driven computational model and computer security using cryptographic and steganographic methods.

Norbert Ádám graduated (M.Sc.) with distinction at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2003. He defended his Ph.D. in the field of computers and computer systems in 2007, his thesis title was "Contribution to Simulation of Feed-Forward Neural Networks on Parallel Computer Architectures". Since 2006 he has been working as Professor Assistant at the Department of Computers and Informatics. Since 2008 he is the Head of the Computer Architectures and Security Laboratory at the Department of Computers and Informatics. His scientific research is focused on the parallel computers architectures.

Anton Baláž received his Master's degree in informatics in 2004 from the Faculty of Electrical Engineering and Informatics, Technical University of Košice. In 2008 he received his Ph.D. in the area of computer security. Since 2007 he has been working as Professor Assistant at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics, Technical University of Košice.

Jaroslav Porubän received his M.Sc. in 2000 and his Ph.D. in computer science in 2004. Since 2013 he is the Head of the Department of Computers and Informatics at Technical University of Košice. His research is focused on the fields of empirical software engineering, domain-specific and programming languages, and human-computer interaction. He was involved in the research projects dealing with implementation of domain-specific and programming languages, language evolution and composition, and software engineering. In 2018 he established open laboratory OpenLab for evaluation of next generation human-computer interaction concepts.

Eva Chovancová graduated (Ing.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2009. She defended her Ph.D. in the field of computers and computer systems in 2012, her thesis title was "Specialized Processor for Computing Acceleration in the Field of Computer Vision". Since 2012 she has been working as Assistant Professor at the Department of Computers and Informatics. Her scientific research is focused on the multicore computer architectures.