ISSN 1392–124X (print), ISSN 2335–884X (online) INFORMATION TECHNOLOGY AND CONTROL, 2013, Vol. 42, No. 3

Covert Channel for Cluster-based File Systems Using Multiple Cover Files

Nerijus Morkevičius

Computer Department, Kaunas University of Technology,
Studentų str. 50-214a, LT-51368, Kaunas, Lithuania
e-mail: nerijus.morkevicius@oksl.ktu.lt

Grigas Petraitis

Bentley Systems Lithuania,
Svitrigailos str. 11H, LT-03228, Vilnius, Lithuania

Algimantas Venčkauskas

Computer Department, Kaunas University of Technology,
Studentų str. 50-212, LT-51368, Kaunas, Lithuania
e-mail: algimantas.venckauskas@ktu.lt

Jonas Čeponis

Computer Department, Kaunas University of Technology,
Studentų str. 50-212, LT-51368, Kaunas, Lithuania

crossref http://dx.doi.org/10.5755/j01.itc.42.3.3328

Abstract. Problems of sensitive information hiding in disk drives using cluster-based file systems are analyzed in this study. A new covert channel method for information hiding in disk drives is proposed and discussed. The method uses multiple cover files and is based on relative allocation of clusters of cover files in relation to one another. The experimental results presented in this paper show that the proposed method is easy to implement, provides good (for the covert channel) storage capacity and has the property of two-fold plausible deniability. The proposed covert channel method can be used for the storage of small and very sensitive information (such as passwords or encryption keys) on removable disk drives.

Keywords: information hiding; steganography; covert channels; disk forensics.

1. Introduction

Secure storage and transmission of sensitive digital information is a very important topic. Traditionally secret information is encrypted by using cryptographically strong algorithms, which are constantly complemented with new methods [14], [13]. Still, in some usage scenarios, data encryption by using cryptographic methods is not sufficient. For example, a very important problem in cryptography is generation, management and storage of key information [21], [22]. Secret keys are small, but very sensitive information which in some cases must be stored in the disk drives. Security provided by traditional encryption methods is usually not sufficient in such situations.

The fact of the existence of encrypted information using cryptographic methods is easily detectable, so, the owner could be forced to reveal it [19]. One of the similar scenarios is the so-called "the prisoner's problem" which was first introduced by Lampson in [12]. The most widely used data hiding strategies, which can solve this problem, are steganography and covert channels [17]. The steganography methods hide sensitive information inside the innocuous information blocks, which are transferred or stored [19]. Covert channels are a subclass of information hiding methods and provide means to hide sensitive information in the media that is neither designed for nor intended to transfer information [20].

Khan *et al.* in [11] describe two levels of security in data hiding methods. Plausible deniability "is a security property of a mechanism that allows parties to claim to others (e.g., a judge) that some information is not in their possession" [15]. Usually the term "plausible deniability" means that the parties can reveal only a part of hidden and/or encrypted data and successfully deny that any further data exists. The fact of the existence of the hidden data is not denied.

On the other hand, "two-fold plausible deniability can be used to provide the covertly communicating parties two chances to deny the presence of hidden information" [11]. The first chance is to deny the fact of the existence of any hidden data. The second is the same as in the plausible deniability scenario. Only the covert channel methods, which do not add any additional information while storing or transferring hidden data, have the property of two-fold plausible deniability.

In this paper we present a new effective and simple to implement method for sensitive information storage on the disk drives by using cluster-based file systems. The experimental results show that the proposed method has the property of two-fold plausible deniability.

2. Related work

The data hiding methods are usually intended to be used in communication protocols for sensitive data transmission [9], [4]. Works [5], [16], [18] provide effective data hiding methods for storing sensitive information in the disk drives. General steganographic methods for data hiding in an unused space of disk structures and file systems are discussed in [5], [10], [18], but in those cases the information can be easily revealed by a third party in case it analyzes the disk. Anderson *et al.* [1] proposed the steganographic file system which uses two different methods for data hiding in the cover files by filling the file system with random data. Other authors [15], [16] introduce further improvements to this method.

All steganographic approaches and other popular data hiding implementations [3] use random data written into various "unused" places of file systems and disks, which is easily detectable by a third party's investigator, so, consequently, they have only the property of plausible deniability.

Two covert channel methods providing two-fold plausible deniability and intended for sensitive data storage in the disk file systems were proposed by Khan *et al.* in [11]. These methods do not require any additional information to be written to the disk. They hide information by using special distribution of disk allocation units (blocks or clusters) containing innocuous cover files. Both methods use one cover file. The real secret binary message is embedded using the distribution of cover file clusters in FAT [2] file system.

The first method applies the fragmentation property of the files as storage media. The cluster of each file starting with the second in FAT file system may be next to the previous cluster (not fragmented) or, if the next cluster is occupied, it may be in some other place on the disk (fragmented). Bits of a secret message are consecutively hidden by the corresponding fragmented or not fragmented clusters of the cover file. If two consecutive bits in the message are the same, then the non-fragmented cluster of the file is written to the disk. This method is relatively simple to implement, but the problem occurs when the non-fragmented cover file cluster has to be written to the cluster already occupied by some other file. The authors of the method provide solutions for such situations.

The second method presented in [11] is based on the approach that the message is divided into small parts, these parts being interpreted as natural numbers, and then each number is expressed by the distance between the fragmented clusters of the cover file. In such a case, the capacity of the covert channel is increased compared to the first approach, but the data embedding algorithm is more complex. The authors discuss two types of collisions which may occur during the message hiding process. Although the authors present the algorithm which allows dealing with all types of collisions, the method is quite complex.

3. The proposed data hiding method

The main idea of the proposed data embedding method is to use more than one cover file and to hide the secret message by using relative positions of the clusters of these files with respect to each other. In such a case, all other files on the file system are ignored and only the clusters belonging to all cover files are analyzed. If we create the array of indexes of these clusters in the file system and sort it in ascending order, we get a sequence of clusters belonging only to the cover files. If for some array index we have a situation when the cluster of one cover file is followed by the cluster of another cover file, then we can interpret this situation as a bit "1" of the hidden message and vice versa.

Moreover, in the case of using more than two cover files, we can assign relative numbers to each cover file and, for each index of the sorted clusters' array, calculate the difference between the assigned numbers of the previous and the current files. By means of this strategy we can embed more than one bit of the secret message into one cluster of the disk file system. In such a case, the names of the cover files become a secret key, because the receiving party has to know the names and the exact order of these files in order to successfully read the hidden message. The formal algorithms of data hiding and retrieval by using two and more cover files are presented in the following subsections.

3.1. Data hiding algorithm

Input:

Message M to be embedded, $M = [b_0, b_1, \dots, b_{n-1}]$, where n is the length of the message in bits, b_i, $i = 0, 1, \dots, n-1$ – message bits.

Cover files $F_0, F_1, \dots, F_{p-1}$, where p is the number of cover files, $p = 2^m$, $m \in N$. Natural number m is considered as secret shared information, part of the secret key. The file names of the cover files F_i are strings t_i, $i = 0, 1, \dots, p-1$. The file order is essential information.

Matrix $C = [c_{ij}]$ contains the numbers of clusters containing cover files F_i. In such way each cover file F_i can be represented as the array of entries of this matrix, $F_i = [c_{i0}, c_{i1}, \dots, c_{iL_i}]$, where L_i is the total number of clusters for file F_i. In the worst case (when all the cover files contain important information and have to be preserved) the lengths of all cover files F_i should be $L_i \geq k$, $k = n/m$.

Array D contains the indices of empty clusters of the disk file system before the hiding procedure, $D = [c_1, c_2, \dots, c_{L_D}]$, $c_1 < c_2 < \dots < c_{L_D}$, where c_i, $i = 1, 2, \dots, L_D$ are the indices of empty clusters in the disk file system. In the worst case, L_D should comply with the following inequality:

$$L_D \geq \sum_{i=0}^{p-1} L_i \,. \qquad (1)$$

Preparation:

Message M is divided into blocks of m bits each, $M = [B_1, B_2, \dots, B_k]$, where $k = n/m$. If the last block is not full, the message is padded with zero bits up to a full block. Each bit block B_i is interpreted as the natural number: $B_i \in N$, $i = 0,1,\dots,k$, $0 \leq B_i \leq p-1$. Where, the value of the block B_0 (initialization vector) is the additional secret information or the last block of the previously embedded message.

Hiding:

```
begin
  $j_i := 0$, $i = 0,1,\dots,p-1$;
  for $i$=1 to $k$ do
      $N_i := B_{i-1} + B_i \pmod{p}$, $0 \leq N_i \leq p-1$;
      $c_i := c_{N_i j_{N_i}}$ (Write one cluster of the cover
              file $F_{N_i}$ to the file system at
              position $c_i$);
      $j_{N_i} := j_{N_i} + 1$;
  end
    Write the remaining parts of the cover files
    $F_0, F_1, \dots, F_{p-1}$ to the file system (if needed);
end
```

Output:

Modified cluster distribution D on the file system containing cover files.

The information needed to retrieve the hidden information (secret key): value m, initialization vector B_0 and the permutation of all cover file names t_i.

3.2. Data retrieval algorithm

Input:

Secret key information: hiding parametre m, $m \in N$; initialization vector B_0 and the permutation of names t_i of all cover files $F_0, F_1, \dots, F_{p-1}$, $p = 2^m$, also, the length n of the hidden message.

The array D of the indexes of clusters in the disk file system containing cover files F_i, $i = 0, 1, \dots, p-1$, $D = [c_1, c_2, \dots, c_{L_D}]$, $c_1 < c_2 < \dots < c_{L_D}$. Each cluster c_l in array D belongs to one and only one of cover files ($c_l = c_{i_l j_l}$, $l = 1,2,\dots,L_D$, where $c_{i_l j_l}$ is the j_l-th cluster of the cover file F_{i_l}).

Preparation:

Message to be retrieved M (memory space for message) is divided into k blocks of m bits each, $M = [B_1, B_2, \dots, B_k]$. Each bit block B_i is interpreted as a natural number.

Retrieving:

```
begin
  $j_i := 0$, $i = 0,1,\dots,p-1$;
  for $l$=1 to k do
      read the $l$-th cluster $c_l$ from the disk file
      system;
      find cover file index $N_l$, such that
      $c_l = c_{N_l j_{N_l}}$;
      $j_{N_l} := j_{N_l} + 1$;
      $B_l := N_l - B_{l-1} \pmod{p}$, $0 \leq B_l \leq p-1$;
  end
    discard trailing bits in the decoded message $M$
    exceeding length $n$;
end
```

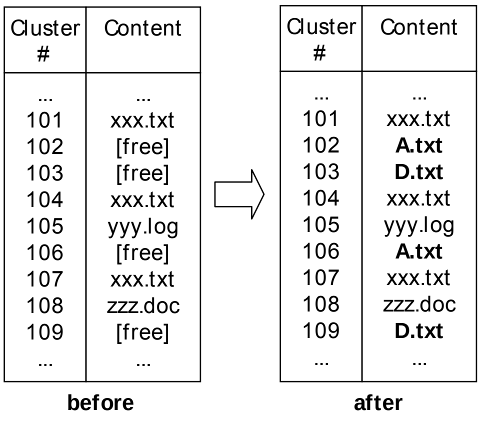
Output:

Retrieved message M.

3.3. Data hiding example

Suppose we want to embed binary message $M = [1,0,0,1,1,1,0,0]$ ($n = 8$) to the cluster-based file system. Assume that we have some free clusters as shown in Fig. 1, "before" part.

We freely choose parameter $m = 2$, so now we need $p = 2^m = 4$ cover files. We choose files with names $t_0 = "A.txt"$, $t_1 = "B.txt"$, $t_2 = "C.txt"$ and $t_3 = "D.txt"$ for this purpose. At least 4 free clusters are needed in the file system used to embed a secret message. Also, all four cover files should be at least 4 clusters long (if we face the worst case scenario, when all message is embedded by using only one file). According to Fig. 1, array of indexes of free clusters is $D = [102,103,106,109,\dots,]$. Now we can divide message M into 2-bit length blocks: $M = [2,1,3,0]$ ($B_1 = 2, B_2 = 1$, etc.). Additionally we have to freely choose the value for the initialization vector $B_0 = 2$.



Cluster #	Content
...	...
101	xxx.txt
102	[free]
103	[free]
104	xxx.txt
105	yyy.log
106	[free]
107	xxx.txt
108	zzz.doc
109	[free]
...	...

before

Cluster #	Content
...	...
101	xxx.txt
102	**A.txt**
103	**D.txt**
104	xxx.txt
105	yyy.log
106	**A.txt**
107	xxx.txt
108	zzz.doc
109	**D.txt**
...	...

after

Figure 1. The proposed data hiding scheme applied to the file system fragment

During the data hiding phase we take the message block B_1 and calculate the index of the cover file: $N_1 \coloneqq B_0 + B_1 (mod 4) = 2 + 2 (mod 4) = 0$. Now we have to write the first cluster of F_0 ("A.txt") into the first free cluster of the file system at the address $102 (c_1 \coloneqq c_{00})$, increase the current cluster counter for file F_0 and proceed to the next message block. One can easily calculate that $N_2 \coloneqq 3$, $N_3 \coloneqq 0$ and $N_4 \coloneqq 3$. So, free clusters should be filled as shown in Fig. 1 "after" part: $c_2 \coloneqq c_{30}$, $c_3 \coloneqq c_{01}$, $c_4 \coloneqq c_{31}$. When the procedure is completed, the modified cluster layout of the file system is $D = [102, 103, 106, 109, \dots,] = [c_{00}, c_{30}, c_{01}, c_{31}]$. On the other hand, $F_0 = [102,106,\dots]$ and $F_3 = [103,109,\dots]$.

The last step is to freely write the remaining clusters of all cover files into free clusters following the filled ones (or just discard the remaining cover file content if it is not important).

For data recovery, we have to know the "secret key", i. e. information which was used for data hiding: $m = 2$, $t_0 = "A.txt"$, $t_1 = "B.txt"$, $t_2 = "C.txt"$, $t_3 = "D.txt"$ and initialization vector $B_0 = 2$ (also, $n = 8$). To retrieve the message we prepare space for message M, and divide it into 2-bit blocks $M = [B_1, B_2, B_3, B_4]$.

Message retrieval is done by sequentially reading clusters c_l belonging to all cover files starting with the smallest cluster number. We know which cover file each cluster belongs to, so we can find the cover file index N_l. From Fig. 1 "after" situation, we know that $c_1 = 102$ and $c_1 = c_{00}$, because cluster with index 102 belongs to file "A.txt", so $N_1 = 0$. Now we can calculate value of the first message block: $B_1 \coloneqq N_1 - B_0 (mod 4) = 0 - 2 (mod 4) = 2$. Similarly, all the remaining message blocks are recovered ($B_2 \coloneqq 3 - 2 (mod 4) = 1$, $B_3 \coloneqq 3$ and $B_4 \coloneqq 0$).

4. Evaluation of the method

The proposed data hiding method in cluster-based file systems using multiple cover files is very straightforward to implement. Unlike two similar methods, it does not create any collisions during the operation, so other files on the disk do not have to be touched during data embedding, which increases the speed of operation.

Our experiments show that the storage capacity of the proposed method, using two cover files, is equal to the capacity of the first method in [11], which is one bit per cluster. The storage capacity when using more files is comparable to the second method in [11], but our method does not depend on the occupation of the file system by other (including cover) files.

The proposed data hiding method requires a secret key to successfully recover the data; this provides extra level of security for the hidden data causing multi-level property of plausible deniability.

Our method does not write any additional information into the disk at all, so none of the byte level disk content analysis methods would detect the fact of the hidden data. File fragmentation is increased by using all similar methods, but this situation is not uncommon in file systems.

The proposed method not only fragments the cover files, but also introduces a special type of fragmentation – the interlaced files. Further in this paper we provide experimental results which show that the increased fragmentation and the interlaced files can be justified by natural causes, which means that the proposed method possesses the property of two-fold plausible deniability, when the fact of the existence of the hidden data could be denied.

4.1. Performance evaluation

The storage capacity of the proposed method can be expressed by the following equation:

$$C = \log_2 p, \qquad (2)$$

where C is the covert channel storage capacity expressed in bits per cluster, p is the number of cover files. Equation (2) is relevant only in the case when the information stored in all cover files is not important, and we can discard the remaining parts of the cover files, after the whole secret message is embedded. If this is not the case, then the total required storage is bounded by inequality (1). Further in this paper we discuss only the first case, which is more likely in real situations.

The study of real file systems of various types shows that the most common gaps between two clusters of the same file are rather small. The most common gap is 8, 16 or 32 blocks as observed by Garfinkel in his study [7]. After the analysis of the available file systems, Khan *et al.* [11] say that if the data hiding algorithm claims to be undetectable by the investigator, then it has to use the most common sizes of gaps between the clusters: "... a good algorithm might use a gap size of 7/8/6 clusters to represent one type of the hidden bit ...".

Considering these conditions, we can compare the storage capacity of the methods under discussion. In Fig. 2 two methods using different parameters are compared. The first three lines (marked as Khan) represent the second (more efficient in terms of capacity) method from [11]. Parameter l in this method represents bits per cluster. In the case when we require the gaps between the clusters to be less than 32, we have three "good" values for l (3, 4 and 5). Other parameters are the same as in Fig. 6 in [11], $\alpha = 0.5$. The last three lines represent the proposed multi-file (MF) data hiding method, using 4, 8 and 2 cover files, respectively.

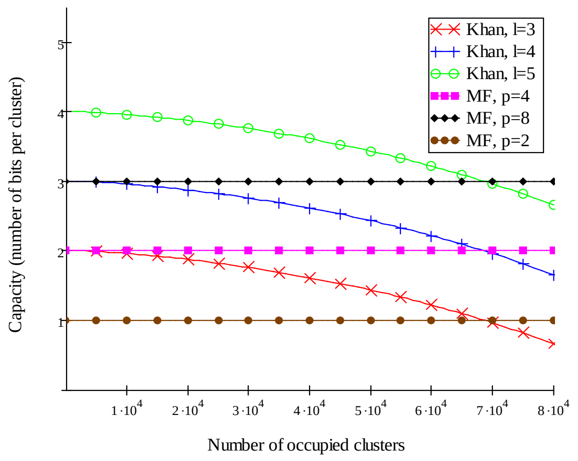


Figure 2. Comparison of the capacity of the data hiding methods

Fig. 2 shows that the storage capacity per cluster of the proposed method does not decrease when the file system is filled, which means that our method is more suitable for small disks (flash cards, USB drives, etc.). Storage capacity is comparable to other methods and gets relatively better when the number of the occupied cluster increases. It is even possible to fully fill the disk with cover files and use each cluster for the storage of hidden information, which is very unlikely when using other methods.

4.2. Stability of the hidden data

The steganography methods, which rely on the hidden information storage in the unused places of file systems, are vulnerable to accidental corruption caused by modification of the file system by OS or other program unaware of the existence of the hidden data. For example, a new file created by the user can overwrite the hidden data stored in the unallocated sectors of the disk.

The proposed method does not store any additional information and all structures of the file system fully comply with the standards (i. e. FAT32), so the hidden information is resistant to common disk modifications (regular file edition, deletion, creation, etc.). On the other hand, some low level tools (such as disk defragmentation utilities) can change the cover file allocation and destroy the hidden data. Modification and appendage to one of the cover files are risky operations which can cause reallocation of the file clusters. Renaming, moving, copying or deleting one of the cover files will destroy the hidden information.

4.3. Security evaluation

The proposed covert channel method is undetectable by disk byte analysis methods. The only noticeable result it creates is the increased fragmentation of files. Fragmentation of files is a natural state of nearly full, frequently used file systems. One question, however, still exists: what parameters of the hiding algorithm should there be that the fragmentation caused by it would be the same as caused by natural reasons?

We have performed the extensive analysis of the used disk drives available in our university campus to find out how the files fragment during the basic usage. We also considered similar studies carried out in [7], [11]. We used "The Sleuth Kit" [6], [8] to collect intermediate statistical information from disk drives. Then this information was transformed and imported into the relational database. All the further analysis was conducted by using common queries and some programming methods against this database. The results confirm the ones presented in other papers.

However, the most important questions were related to the specifics of our method. Do real file systems contain interlaced files? Is there any "normal" disk usage pattern, which causes files to interlace?

To be more specific, we have to formally define the term of interlaced files.

Definition. Two files $F_1 = [c_{10}, c_{1l}, \dots, c_{1L_1}$ and $F_2 = [c_{20}, c_{2l}, \dots, c_{2L_2}$ are considered interlaced if there exists at least one index pair i, j, such that $c_{1i} < c_{2j} < c_{2j+1} < c_{1i+1}$ or $c_{2j} < c_{1i} < c_{2j+1} < c_{1i+1}$. If such a situation is encountered several times, then two files F_1 and F_2 are interlaced corresponding number of times, we will call it the number of interlaces.

On the other hand, each file F_i can be interlaced with a number of other files, denoted by nF_i. If we count all the interlaces of file F_i with all other files, we will get the total number of interlaces for file F_i, denoted by NF_i.

The results of the study of the available file systems are presented in Fig. 3. The bars in this chart represent the relative quantity of the interlaced files. For example, the first bar means that 4.33% of all files are interlaced with at least one other file ($nF \geq 1$). The curve represents the average number of the interlaces (*NF*) for each interlaced file in the corresponding pool. For example, all files which are interlaced with at least three other files (second bar) have on average 1460 interlaces.

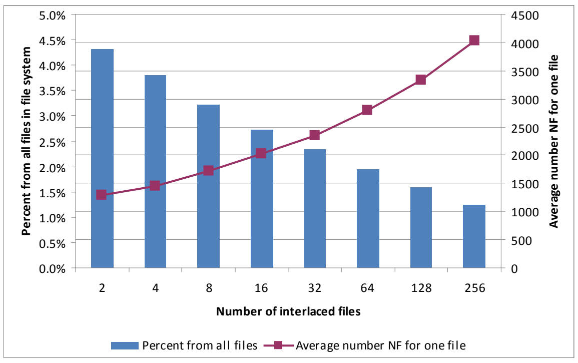


Figure 3. Amount of interlaced files (nF) and average number of interlaces (NF) for one interlaced file

Most interlaced files according to the file type are listed in Table 1. Only the types of files which on average were interlaced with at least three other files and the average number of such files in the analyzed file systems was more than 10 are included.

Table 1. Most interlaced files according to the file type

File type	Avg. number of files on disk	Avg. file size in clusters	Avg. number of interlaced files (*nF*)	Max. number of interlaced files (*nF*)
.msi	14	893	326	5357
.jar	132	204	134	5715
.zip	16	2153	127	5062
.msp	22	2320	114	5247
.msf	15	39	106	3774
.LOG	124	41	102	4659
.otp	12	46	100	3036
.cab	33	885	81	5662
.db	29	67	76	4704
.frx	62	7	61	2728
.mo	103	16	54	3260
.svg	23	7	53	3036
.dll	2159	111	45	6158

Another interesting quality of file types observed during our analysis is a relative tendency to interlace which is summarized in Fig. 4. The numbers represented by bars are calculated by dividing the average number of the interlaced files (*nF*) by the length of the corresponding file. These numbers show which file types are best candidates for cover files. The investigation shows that small files, which are frequently updated during their lifetime, have a bigger tendency to interlace.

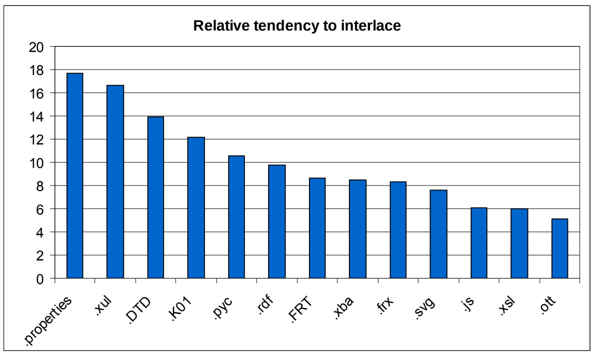


Figure 4. Relative tendency to interlace according to the file type

To answer the second question, we created the software which simulates different situations in Windows 7, Windows XP and Ubuntu 11.04 operating systems using FAT32 and NTFS file systems. The software starts several threads which perform the following actions: 1. Copy several files simultaneously into the same location; 2. Download several files simultaneously into the same location by using BitTorrent protocol; 3. Write several files simultaneously by using “append” method. The main point here is that this software does not perform any direct access to the file system structures; it fully relies on the standard system calls of OS.

The experimental results show that the first two scenarios do not produce any significant fragmentation. The results of the third scenario are summarized in Table 2. As one can see, the files are extremely highly interlaced.

Table 2. Results of parallel work with files

OS	File system type	Number of threads	Average number of fragments	Number of occurrences of interlaced files (*NF*)
Linux	FAT32	2	24774	49546
Linux	NTFS	2	7	12
Windows 7	FAT32	2	25579	51157
Windows 7	NTFS	2	19869	29684
Windows XP	FAT32	2	16457	14812
Windows XP	NTFS	2	24675	49341
Windows 7	FAT32	3	14122	46995
Windows 7	NTFS	3	25326	97875
Windows 7	FAT32	4	11410	44229
Windows 7	NTFS	4	25521	148634

5. Conclusions

In this paper we have presented a new method for secure data hiding in cluster-based file systems. This method uses multiple cover files and is based on the relative allocation of the clusters of cover files in relation to one another. Our method is very easy to implement, because it does not cause any collisions with other files (regular or cover).

The performance of the proposed method does not depend on the size of the file system or free space left on the disk drive (see Fig. 2). It provides the constant bit per cluster storage capacity ratio and enables full filling of the whole file system with the cover files. Based on the situation, one can freely choose the number of cover files whose names and order compose the secret key information needed to decode the message. The choice of the number of the cover files enables a wide range variation of the covert channel storage capacity.

Our experiments show that the storage capacity of the proposed method is comparable or higher to other existing methods, provided they are used with the parameters recommended for secure (undetectable) operation (see Fig. 2).

Although all covert channel techniques provide secure storage capabilities which are inferior to the traditional encryption methods, they are perfect for storing small but very sensitive information. The proposed data hiding method is best suited for secure storage of encryption keys and user passwords on removable media, such as flash drives.

One can choose to use two or four cover files, if she wants to hide bank account's password in her flash drive. On the other hand, if one wants to hide a secret file of considerable size, the better choice is to use 128 or even 256 cover files. Experimental results show (see Fig. 3) that in both cases the fact of existence of hidden information is plausibly deniable.

We have implemented the proposed data hiding method for usage with the FAT32 file system. This file system is most widely used on the small removable media devices; it is supported in almost all operating systems and compatible with widest range of smart devices. On the other hand, experimental results show (see Table 2) that modern NTFS file system is another good candidate for implementation of the proposed covert channel method.

The proposed data hiding technique causes fragmentation of the file system and creates interlaced files. Our experiments confirm that the interlaced files are common in intensively used file systems; moreover, heavily interlaced files can be caused by ordinary programs which constantly append information to several files. As a final conclusion, we can state that the proposed data hiding method has the property of two-fold plausible deniability.

References

[1] **R. J. Anderson, R. M. Needham, A. Shamir**. The steganographic file system. In: *Proceedings of the Second International Workshop on Information Hiding*, London, UK, 1998, pp. 73–82.

[2] **B. Carrier**. *File system forensic analysis*. Addison-Wesley Professional, 2005.

[3] **A. Czeskis, D. J. S. Hilaire, K. Koscher, S. D. Gribble, T. Kohno, B. Schneier**. Defeating encrypted and deniable file systems: TrueCrypt v5.1a and the Case of the Tattling OS and Applications. *HotSec*, 2008, pp. 1–7.

[4] **D. M. Dakhane, S. Patil, M. Patil**. Detection and elimination of covert communication in transport and internet layer - A Survey. In: *IJCA Proceedings on International Conference on Recent Trends in Information Technology and Computer Science (ICRTITCS-2011),* New York, USA, 2012, Vol. 1, pp. 36–41.

[5] **K. Eckstein, M. Jahnke**. Data hiding in journaling file systems. In: *Refereed Proceedings of the 5th Annual Digital Forensic Research Workshop*, New Orleans, Louisiana, USA, 2005, pp. 1–8.

[6] **D. V. Forte**. Open source forensics: The PTK: An alternative advanced interface for Sleuth Kit. *Network Security*, 2008, Vol. 2008, 10–13.

[7] **S. L. Garfinkel**. Carving contiguous and fragmented files with fast object validation. *Digital Investigation*, 2007, Vol. 4, Suppl., 2–12.

[8] **S. L. Garfinkel**. Automating disk forensic processing with SleuthKit, XML and Python. *SADFE*, 2009, 73-84.

[9] **G. Hanaoka, Y. Hanaoka, M. Hagiwara, H. Watanabe, H. Imai**. Unconditionally secure chaffing-and-winnowing: a relationship between encryption and authentication. In: *Proceedings of the 16th international conference on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, Berlin, Heidelberg, 2006, pp. 154–162.

[10] **E. Huebner, D. Bem, C. K. Wee**. Data hiding in the NTFS file system. *Digital Investigation*, 2006, Vol. 3, 211–226.

[11] **H. Khan, M. Javed, S. A. Khayam, F. Mirza**. Designing a cluster-based covert channel to evade disk investigation and forensics. *Computers & Security*, 2011, Vol. 30, 35–49.

[12] **B. W. Lampson**. A note on the confinement problem. *Communications of the ACM*, 1973, Vol. 16, 613–615.

[13] **K. Luksys, E. Sakalauskas**. Matrix power cipher. *Information Technology and Control*, 2012, Vol. 41, 349–355.

[14] **K. Luksys, E. Sakalauskas, A. Venckauskas**. Implementation analysis of matrix power cipher in embedded systems. *Electronics and Electrical Engineering*, 2012, Vol. 118, 95–98.

[15] **A. D. McDonald, M. G. Kuhn**. StegFS: a steganographic file system for Linux. *Information Hiding*, 1999, Vol. 1768, 462–477.

[16] **H. Pang, K. L. Tan, X. Zhou**. StegFS: a steganographic file system. In: *Proceedings of the 19th International Conference on Data Engineering,* 2003, Vol. 1, pp. 657–668.

[17] **F. A. P. Petitcolas, R. J. Anderson, M. G. Kuhn**. Information hiding - a survey. In: *Proceedings of the IEEE: Special Issue on Protection of Multimedia Content,* 1999, Vol. 87, pp. 1062–1078.

[18] **S. Piper, M. Davis, G. Manes, S. Shenoi**. Detecting hidden data in ext2/ext3 file systems. *IFIP Int. Conf. Digital Forensics,* 2005, 245–256.

[19] **N. Provos, P. Honeyman**. Hide and seek: an introduction to steganography. *IEEE Security and Privacy*, 2003, Vol. 1, 32–44.

[20] **G. J. Simmons**. The prisoners' problem and the subliminal channel. *Advances in Cryptology, Proceedings of Crypto 83*, New York, 1984, pp. 51–67.

[21] **A. Venckauskas, N. Jusas, L. Kizauskiene, E. Kazanavicius, V. Kazanavicius**. Security method of embedded software for mechatronic systems. *Mechanika*, 2012, Vol. 18, 196–202.

[22] **A. Venckauskas, N. Jusas, I. Mikuckiene, S. Maciulevicius**. Generation of the secret encryption key using the signature of the embedded system. *Information Technology and Control*, 2012, Vol. 41, 368–375.

Received January 2013.