Multimed Tools Appl (2012) 61:243–261
DOI 10.1007/s11042-010-0704-y

A study on multimedia file carving method

Byeongyeong Yoo · Jungheum Park · Sungsu Lim ·
Jewan Bang · Sangjin Lee

Published online: 19 January 2011

Abstract File carving is a method that recovers files at unallocated space without any file information and used to recover data and execute a digital forensic investigation. In general, the file carving recovers files using the inherent header and footer in files or the entire file size determined in the file header. The largely used multimedia files, such as AVI, WAV, and MP3, can be exactly recovered using an internal format in files as they are continuously allocated. In the case of the NTFS, which is one of the most widely used file system, it supports an internal data compression function itself, but the NTFS compression function has not been considered in file carving. Thus, a large part of file carving tools cannot recover NTFS compressed files. Also, for carving the multimedia files compressed by the NTFS, a recovery method for such NTFS compressed files is required. In this study, we propose a carving method for multimedia files and represent a recovery plan for deleted NTFS compressed files. In addition, we propose a way to apply such a recovery method to the carving of multimedia files.

Keywords Multimedia file · File carving · NTFS compressed file

1 Introduction

According to the increase in digital devices, a large part of information has been stored as a type of digital data. Therefore, digital evidences are recognized as an important

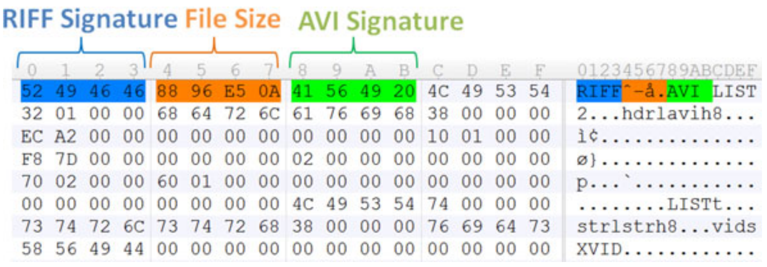
B. Yoo · J. Park · S. Lim · J. Bang · S. Lee (✉)
Center for Information Security Technologies, Korea University, Seoul, Republic of Korea
e-mail: sangjin@korea.ac.kr

B. Yoo
e-mail: pinpanel@korea.ac.kr

J. Park
e-mail: junghmi@korea.ac.kr

S. Lim
e-mail: nemography@korea.ac.kr

J. Bang
e-mail: jwbang@korea.ac.kr



```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
52 49 46 46 88 96 E5 0A 41 56 49 20 4C 49 53 54   RIFFˆ-å.AVI LIST
32 01 00 00 68 64 72 6C 61 76 69 68 38 00 00 00   2...hdrlavih8...
EC A2 00 00 00 00 00 00 00 00 00 00 10 01 00 00   ì¢..............
F8 7D 00 00 00 00 00 00 02 00 00 00 00 00 00 00   ø}..............
70 02 00 00 60 01 00 00 00 00 00 00 00 00 00 00   p...`...........
00 00 00 00 00 00 00 00 4C 49 53 54 74 00 00 00   ........LISTt...
73 74 72 6C 73 74 72 68 38 00 00 00 76 69 64 73   strlstrh8...vids
58 56 49 44 00 00 00 00 00 00 00 00 00 00 00 00   XVID............
```

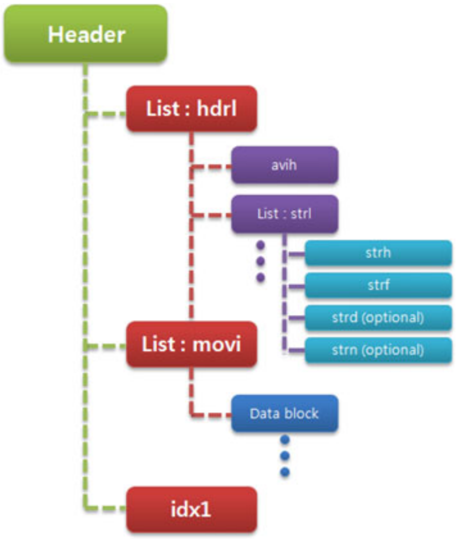
Fig. 1 AVI file header

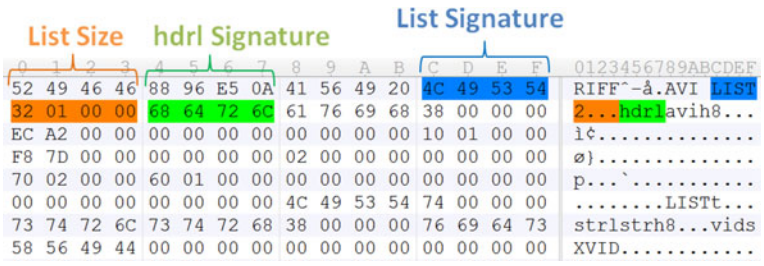
factor in various criminal investigations. As such digital data can be easily fabricated and modified, it is necessary to carefully manage the data in order to use it as legal evidences because it can be very easily damaged using the conventional data management method. Thus, digital forensics has been raised to treat technical and procedural issues for collecting, producing, analyzing, and processing digital evidences. The objective of digital forensics is to clarify realistic truth by analyzing the stored data in a digital device that is related to a criminal case and accept it as an effective evidence in a court.

In a digital forensic investigation, an evidence recovering process is generally implemented through an evidence collection process, and an evidence analysis is carried out using the data in this process. The evidence recovering process is an important step to obtain evidences in a digital forensic investigation. The process can be classified as a method that uses the information or meta data in file systems and a file carving method that recovers files based on signatures and file architectures in the unallocated space of a storage region without any information [4, 7].

Studies on the file carving have been focused on the recovery of document or image files. Thus, there are few studies on the recovery of multimedia files like video

Fig. 2 AVI file structure





```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
52 49 46 46 88 96 E5 0A 41 56 49 20 4C 49 53 54   RIFFˆ–å.AVI LIST
32 01 00 00 68 64 72 6C 61 76 69 68 38 00 00 00   2...hdrlavih8...
EC A2 00 00 00 00 00 00 00 00 00 00 10 01 00 00   ì¢..............
F8 7D 00 00 00 00 00 00 02 00 00 00 00 00 00 00   ø}..............
70 02 00 00 60 01 00 00 00 00 00 00 00 00 00 00   p...`...........
00 00 00 00 00 00 00 00 4C 49 53 54 74 00 00 00   ........LISTt...
73 74 72 6C 73 74 72 68 38 00 00 00 76 69 64 73   strlstrh8...vids
58 56 49 44 00 00 00 00 00 00 00 00 00 00 00 00   XVID............
```

Fig. 3 hdrl list format

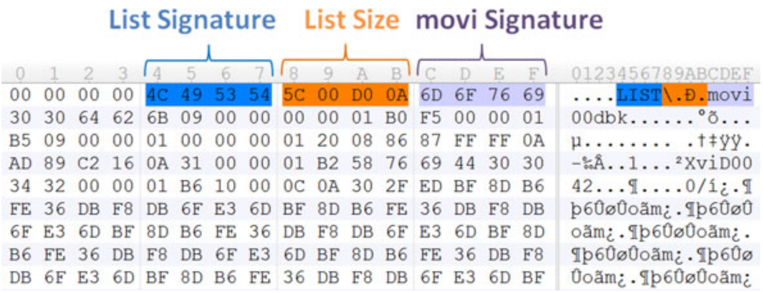
and audio files. The largely used multimedia files, such as AVI, WAV, and MP3, can be exactly recovered using an internal format in files as they are continuously allocated.

Also, the file carving tools widely used at the present time cannot exactly recover NTFS compressed files. It is due to the fact that such carving tools are not considered for that files. As the NTFS compression has been frequently used to save data storage space, there is a high possibility in presenting deleted NTFS compressed files in unallocated space. If multimedia files are stored as the NTFS compressed files, the exact recovery of files is difficult. Therefore, it is necessary to study a recovery method for such deleted NTFS compressed files in file carving.

In this study, we propose a carving method for multimedia files and represent a recovery plan for deleted NTFS compressed files. In addition, we propose a way to apply such a recovery method to the carving of multimedia files.

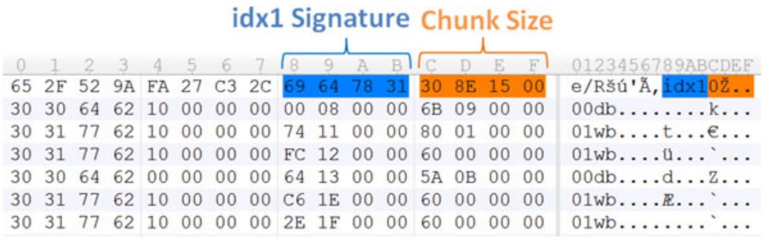
2 Multimedia file carving method

The carving of multimedia files can be performed using an internal format by dividing the files using the Signature of the file header. In general, although the files, which are recovered using the file carving method, can be executed using a specific application of such files if the data of files is perfectly presented, the data of multimedia files can be verified by reproducing them up to the middle of the recovered section through executing it if a space between the header of files and a specific data area exists in most multimedia files.



```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
00 00 00 00 4C 49 53 54 5C 00 D0 0A 6D 6F 76 69   ....LIST\.Ð.movi
30 30 64 62 6B 09 00 00 00 00 01 B0 F5 00 00 01   00dbk......°õ...
B5 09 00 00 01 00 00 00 01 20 08 86 87 FF FF 0A   µ........ .†‡ÿÿ.
AD 89 C2 16 0A 31 00 00 01 B2 58 76 69 44 30 30   -‰Â..1...²XviD00
34 32 00 00 01 B6 10 00 0C 0A 30 2F ED BF 8D B6   42...¶....0/í¿.¶
FE 36 DB F8 DB 6F E3 6D BF 8D B6 FE 36 DB F8 DB   þ6ÛøÛoãm¿.¶þ6ÛøÛ
6F E3 6D BF 8D B6 FE 36 DB F8 DB 6F E3 6D BF 8D   oãm¿.¶þ6ÛøÛoãm¿.
B6 FE 36 DB F8 DB 6F E3 6D BF 8D B6 FE 36 DB F8   ¶þ6ÛøÛoãm¿.¶þ6Ûø
DB 6F E3 6D BF 8D B6 FE 36 DB F8 DB 6F E3 6D BF   Ûoãm¿.¶þ6ÛøÛoãm¿
```

Fig. 4 Movi list format



```
0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
65 2F 52 9A FA 27 C3 2C 69 64 78 31 30 8E 15 00  e/Ršú'Ã,idx10Ž..
30 30 64 62 10 00 00 00 00 08 00 00 6B 09 00 00  00db........k...
30 31 77 62 10 00 00 00 74 11 00 00 80 01 00 00  01wb....t...€...
30 31 77 62 10 00 00 00 FC 12 00 00 60 00 00 00  01wb....ü...`...
30 30 64 62 00 00 00 00 64 13 00 00 5A 0B 00 00  00db....d...Z...
30 31 77 62 10 00 00 00 C6 1E 00 00 60 00 00 00  01wb....Æ...`...
30 31 77 62 10 00 00 00 2E 1F 00 00 60 00 00 00  01wb........`...
```

Fig. 5 Idx1 Chunk format

In this study, we propose a file recovery method that can reproduce data for the carving of multimedia files based on the characteristics of multimedia files. The next section represents the carving of AVI, WAV, and MP3 files.

2.1 AVI files carving

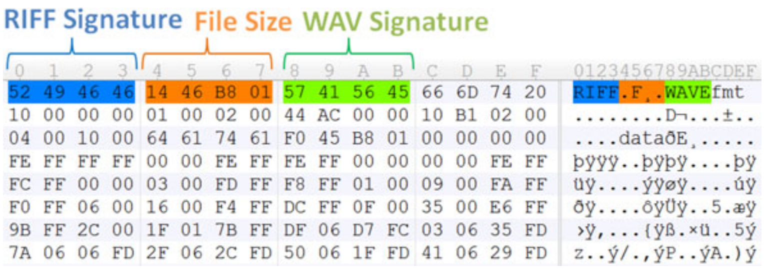
AVI files are one of the most largely used video files at the present time. The AVI files internally use a resource interchange file format (RIFF). The RIFF is a common format used to store various types of data, such as image and audio, as a single file and used as a base format of the multimedia application of Windows. The RIFF consists of several Chunks, Lists, and Data. It is used in various files, but it shows different types of Chunks and Lists used inside the files [1, 2].

For carving AVI files, the Signatures of the header is first to be identified. The AVI file header is a size of 12 Bytes, and Fig. 1 shows its structure.

As shown in Fig. 1, it can be seen that the 4 Bytes 0x52494646 RIFF Signature with offset 0 and the 0x41564920 AVI Signature with offset 8 exist at the AVI file header. After verifying AVI files using such Signatures, it is possible to obtain the data of files using the size of four Bytes with offset 4 that represents the size of the file.

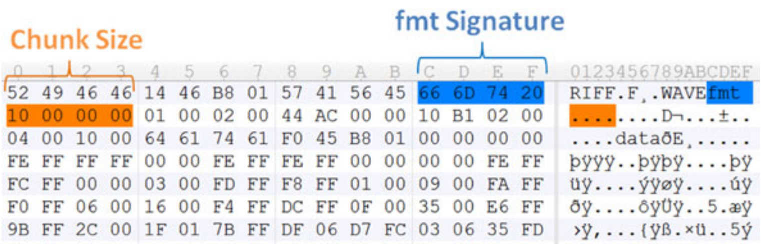
It should verify that whether the minimum data, which is reproducible, exists by checking the format of AVI files before obtaining the data of files. If there is no reproducible data, the file is not recovered.

Figure 2 shows the structure of AVI files. The hdrl List, movi List, and idx1 Chunk are located under the most upper Header. The hdrl List defines the format of data, the movi List stores the actual data of streams, and the idx1 Chunk stores the Data Block information of the movi List. As the original data from the Header to the header of the idx1 Chunk, the all streams that are actually reproduced can be recovered. Also, as a part of the movi List data exist, the recovery can be performed as much as the existing data and that can also be



```
0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
52 49 46 46 14 46 B8 01 57 41 56 45 66 6D 74 20  RIFF.F¸.WAVEfmt
10 00 00 00 01 00 02 00 44 AC 00 00 10 B1 02 00  ........D¬...±..
04 00 10 00 64 61 74 61 F0 45 B8 01 00 00 00 00  ....dataðE¸.....
FE FF FF FF 00 00 FE FF FE FF 00 00 00 00 FE FF  þÿÿÿ..þÿþÿ....þÿ
FC FF 00 00 03 00 FD FF F8 FF 01 00 09 00 FA FF  üÿ....ýÿøÿ....úÿ
F0 FF 06 00 16 00 F4 FF DC FF 0F 00 35 00 E6 FF  ðÿ....ôÿÜÿ..5.æÿ
9B FF 2C 00 1F 01 7B FF DF 06 D7 FC 03 06 35 FD  ›ÿ,...{ÿß.×ü..5ý
7A 06 06 FD 2F 06 2C FD 50 06 1F FD 41 06 29 FD  z..ý/.,ýP..ýA.)ý
```

Fig. 6 WAV header format



```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
52 49 46 46 14 46 B8 01 57 41 56 45 66 6D 74 20  RIFF.F¸.WAVEfmt 
10 00 00 00 01 00 02 00 44 AC 00 00 10 B1 02 00  ........D¬...±..
04 00 10 00 64 61 74 61 F0 45 B8 01 00 00 00 00  ....dataðE¸.....
FE FF FF FF 00 00 FE FF FE FF 00 00 00 00 FE FF  þÿÿÿ..þÿþÿ....þÿ
FC FF 00 00 03 00 FD FF F8 FF 01 00 09 00 FA FF  üÿ....ýÿøÿ....úÿ
F0 FF 06 00 16 00 F4 FF DC FF 0F 00 35 00 E6 FF  ðÿ....ôÿÜÿ..5.æÿ
9B FF 2C 00 1F 01 7B FF DF 06 D7 FC 03 06 35 FD  ›ÿ,...{ÿß.×ü..5ý
```

Fig. 7 fmt Chunk format

reproduced. If the movi List does not exist, the recovery will not be carried out under the judgement that cannot recover the subject files.

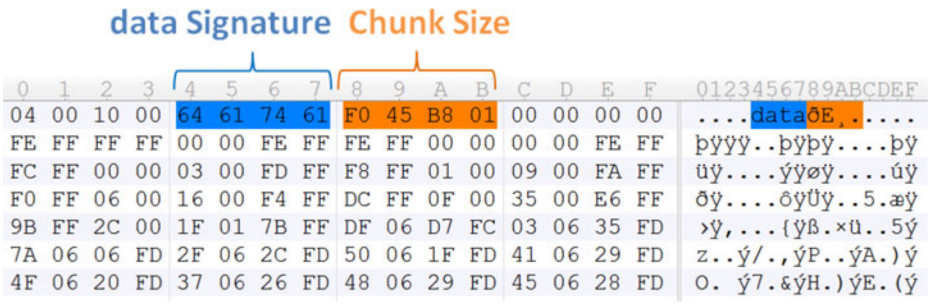
The hdrl List exists right after the header. Figure 3 shows the structure of the hdrl List. It is possible to find the 4 Bytes 0x4c495354 List Signature with offset 0 and the four Bytes 0x6864726c hdrl Signature with offset eight at the beginning of the hdrl List. If the Signature of the hdrl List is checked, the beginning of the movi List can be seen using the List Size with offset four.

Figure 4 shows the structure of the movi List. As shown in Fig. 4, the four Bytes 0x4c495354 List Signature with offset 0 and the four Bytes 0x6d6f7669 movi Signature with offset eight are to be seen at the beginning of the movi List. If the Signature of the movi List is checked and a some part of data exist, the recovery can be attempted. For checking whether files are perfectly presented, it can be verified by moving the position to the beginning of the idx1 Chunk using the List Size with offset four.

Figure 5 shows the structure of the idx1 Chunk. As shown in Fig. 5, if the four Bytes 0x69647831 idx1 Signature with offset 0 is verified at the beginning of the idx1 Chunk, it can be regarded that all practically reproducible data exist.

2.2 WAV file carving

WAV files are one of the most widely used audio files at the present time. As well as the AVI files, the WAV files use the RIFF internally. For carving WAV files, it is necessary to first identify the Signatures in the header. The size of the WAV file header is 12 Bytes. Figure 6 represents the structure of the header.



```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
04 00 10 00 64 61 74 61 F0 45 B8 01 00 00 00 00  ....dataðE¸.....
FE FF FF FF 00 00 FE FF FE FF 00 00 00 00 FE FF  þÿÿÿ..þÿþÿ....þÿ
FC FF 00 00 03 00 FD FF F8 FF 01 00 09 00 FA FF  üÿ....ýÿøÿ....úÿ
F0 FF 06 00 16 00 F4 FF DC FF 0F 00 35 00 E6 FF  ðÿ....ôÿÜÿ..5.æÿ
9B FF 2C 00 1F 01 7B FF DF 06 D7 FC 03 06 35 FD  ›ÿ,...{ÿß.×ü..5ý
7A 06 06 FD 2F 06 2C FD 50 06 1F FD 41 06 29 FD  z..ý/.,ýP..ýA.)ý
4F 06 20 FD 37 06 26 FD 48 06 29 FD 45 06 28 FD  O. ý7.&ýH.)ýE.(ý
```

Fig. 8 Figdata Chunk format

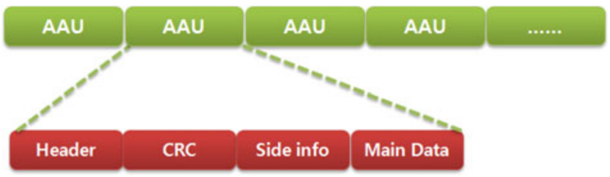


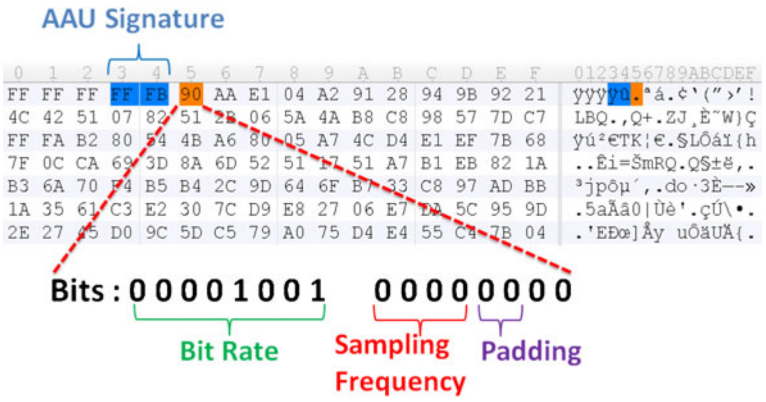
Fig. 9 MP3 file structure

As shown in Fig. 6, the four Bytes 0x52494646 RIFF Signature with offset 0 and the 0x57415645 WAV Signature with offset 8 are verified at the WAV file header. After verifying the WAV file using the Signature, the data of files can be obtained using the four Bytes file size with offset four.

It should verify that whether the minimum data, which is reproducible, exists by checking the format of WAV files before obtaining the data of files. If there is no reproducible data, the file is not recovered. The fmt Chunk and data Chunk are presented under the most upper Header. The fmt Chunk defines the format of data, and the data Chunk stores actual audio data as the unit of a specific block. As WAV files represent no footers of the data Chunk where actual stream data do not exist differed from AVI files, it is not possible to verify whether the data files exist perfectly. However, as the reproducing is possible as a part of data stream exist for a specific level, the recovery is attempted under the consideration that a some part of stream data exist as the Signature of the data Chunk is verified. However, if the Signature is not verified, the recovery will not be performed [6, 10].

The fmt Chunk exists right after the header. Figure 7 shows the fmt Chunk format. It is possible to verify the four Bytes 0x666d7420 fmt Signature with offset 0 at the beginning of the fmt Chunk. If the Signature of the fmt Chunk is verified, the beginning of the data Chunk can be checked using the Chunk Size with offset four.

Figure 8 represents the data Chunk format. As shown in Fig. 8, the four Bytes 0x64617461 data Signature with offset 0 can be verified at the beginning of the data Chunk. If the Signature of the data Chunk is checked and a some part of data exist, the recovery can be attempted. As mentioned above, however, there are no footer, it is difficult to verify whether the data of files exist perfectly.



```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
FF FF FF FF FB 90 AA E1 04 A2 91 28 94 9B 92 21   ÿÿÿÿû.ªá.¢'("›'!
4C 42 51 07 82 51 2B 06 5A 4A B8 C8 98 57 7D C7   LBQ.,Q+.ZJ¸È˜W}Ç
FF FA B2 80 54 4B A6 80 05 A7 4C D4 E1 EF 7B 68   ÿú²€TK¦€.§LÔáï{h
7F 0C CA 69 3D 8A 6D 52 51 17 51 A7 B1 EB 82 1A   ..Êi=ŠmRQ.Q§±ë,.
B3 6A 70 F4 B5 B4 2C 9D 64 6F B7 33 C8 97 AD BB   ³jpôµ´,.do·3È—-»
1A 35 61 C3 E2 30 7C D9 E8 27 06 E7 DA 5C 95 9D   .5aÃâ0|Ùè'.çÚ\•.
2E 27 45 D0 9C 5D C5 79 A0 75 D4 E4 55 C4 7B 04   .'EÐœ]Åy uÔäUÄ{.
```

Fig. 10 AAU header

Table 1 Bit rate for each bit

bits	BitRate, Unit :kbps (kbit/second)
00000000	free
00000001	32
00000010	40
00000011	48
00000100	56
00000101	64
00000110	80
00000111	96
00001000	112
00001001	128
00001010	160
00001011	192
00001100	224
00001101	256
00001110	320
00001111	bad

2.3 MP3 file carving

The MP3 (MPEG Audio Layer-3) is a lossy compression format developed as an MPEG-1 audio specification. It has been considered as the most popular audio file format due to the improvement version of the MP1 and MP2.

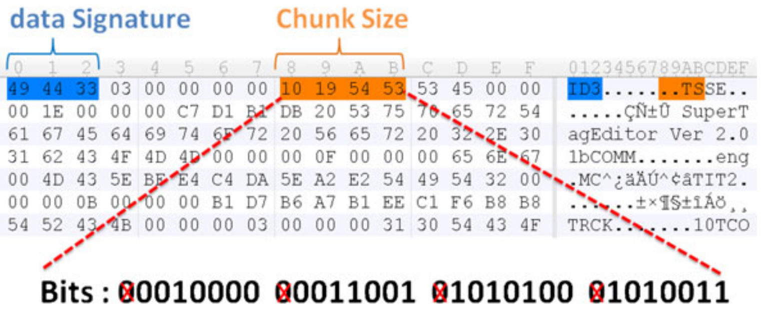
Figure 9 shows the MP3 file structure. The MP3 file stores data as a frame unit called AAU (Audio Access Unit). Also, each AAU consists of Header, CRC, Side Info, and Main Data. The AAU stores actual stream data in the Main Data area, and the stream data can be reproduced using a single AAU only because each AAU stores data independently.

Figure 10 represents the AAU Header format. There exists the two Bytes 0xfffb Signature with offset 0 from the beginning of the Header. The first four bits in the one Byte with offset two shows the Bit Rate, and the next two bits represent the Sampling Frequency, and the next one bit shows the Padding. Tables 1 and 2 represent the Bit Rate and Sampling Frequency, respectively, according to each bit [8].

Then, each AAU size can be calculated using the equation of $[144^{*}\text{Bit Rate}/(\text{Sampling Frequency} + \text{Padding})]$ based on these properties. For instance, the Bit Rate presented in Fig. 10 is 00001001, it becomes 128 kbps. Also, as the Sampling Frequency is 00, it becomes 44100, and the Padding is 0. The the size can be calculated as 417 Bytes by substituting these values to the equation as $[144^{*}128000/(44100 + 0) = 417]$ [8].

Table 2 Sampling frequency for each bit

Bits	Sampling frequency, Unit : hz
00	44100
01	48000
10	32000
11	reserved



```
0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
49 44 33 03 00 00 00 00 10 19 54 53 53 45 00 00  ID3.......TSSE..
00 1E 00 00 00 C7 D1 B1 DB 20 53 75 70 65 72 54  .....ÇÑ±Û SuperT
61 67 45 64 69 74 6F 72 20 56 65 72 20 32 2E 30  agEditor Ver 2.0
31 62 43 4F 4D 4D 00 00 00 0F 00 00 00 65 6E 67  1bCOMM.......eng
00 4D 43 5E BF E4 C4 DA 5E A2 E2 54 49 54 32 00  .MC^¿äÄÚ^¢âTIT2.
00 00 0B 00 00 00 B1 D7 B6 A7 B1 EE C1 F6 B8 B8  ......±×¶§±îÁö¸¸
54 52 43 4B 00 00 00 03 00 00 00 31 30 54 43 4F  TRCK.......10TCO
```

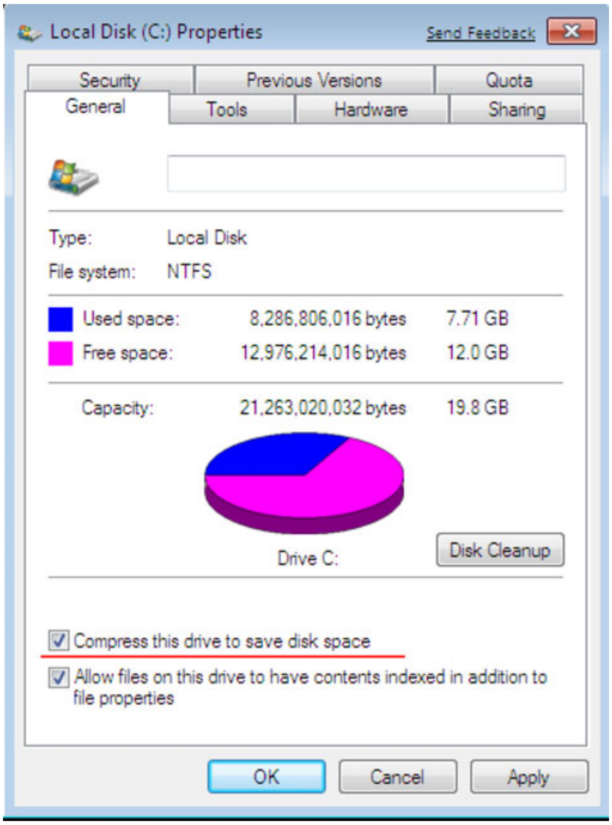
Bits : $\cancel{0}0010000$ $\cancel{0}0011001$ $\cancel{0}1010100$ $\cancel{0}1010011$

Fig. 11 ID3 Tag Ver.2 header

The MP3 file can be classified into three different types, such as the format that is configured by using the AAU only, the ID3 Tag Ver.1 that stores tags at the AAU and the end of the file, and the ID3 Tag Ver.2 that stores tags at the AAU and both the beginning and the end of the file. Because each type shows different configurations, the carving for each type is also a bit different [8].

The carving of the file, which is configured by the AAU only, verifies whether the file is the MP3 file format using the Signature of the AAU Header. Then, the Signature

Fig. 12 Window for the configuration of drive compression



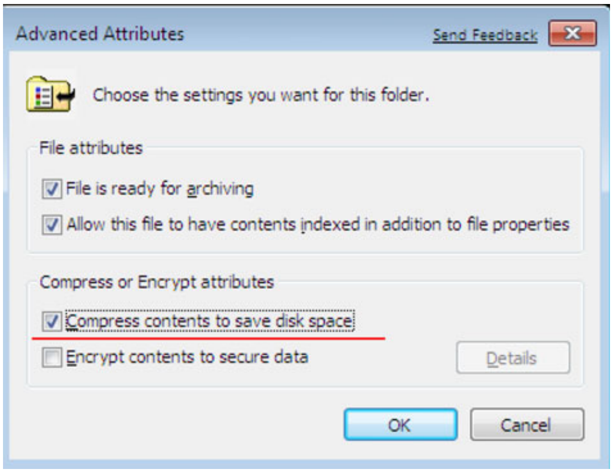
Type: Local Disk
File system: NTFS

🟦	Used space:	8,286,806,016 bytes	7.71 GB
🟪	Free space:	12,976,214,016 bytes	12.0 GB
	Capacity:	21,263,020,032 bytes	19.8 GB

Drive C:

[✓] Compress this drive to save disk space

[✓] Allow files on this drive to have contents indexed in addition to file properties



Advanced Attributes

Choose the settings you want for this folder.

File attributes

- [✓] File is ready for archiving
- [✓] Allow this file to have contents indexed in addition to file properties

Compress or Encrypt attributes

- [✓] Compress contents to save disk space
- [ ] Encrypt contents to secure data

Fig. 13 Window for the configuration of file compression

of the next AAU is to be verified by checking the size of the AAU after that verification. If the Signature is agreed, the next AAU can be searched using the same way. However, the Signature is not the same, the verified AAU is to be recovered only. In the case of the MP3 file that is configured by the AAU only, it is difficult to verify whether the file is perfectly recovered because the footer of the file or the entire size does not exist. However, if all data are continuously allocated, the file can be fully recovered. Also, if all data are not presented, it is possible to recover the data up to the range where the AAU exists.

The carving of the ID3 Tag Ver.1 MP3 file can be performed as the same way used in the MP3 file that is configured by the AAU only. Regarding the difference between these two methods, as the tag data is presented at the end of the file in the ID3 Tag Ver.1, the 0x544155 3 Bytes determined by the Signature of the tag are to be determined as the end of the file and that leads to perfectly recover the file including the tag of 128 Bytes. If the tag data is not verified, the verified AAU is only recovered.

The carving of the ID3 Tag Ver.2 MP3 file is performed by verifying the tag because the tag is presented at the beginning of the file.

Figure 11 shows the tag header of the beginning of the ID3 Tag Ver.2 MP3 file. As the three Bytes 0x494433 with offset 0 is verified from the beginning of the tag header, it is necessary to move the position to the beginning of the AAU through verifying the tag of the four Bytes with offset six. The tag size can be obtained by adding the size of the tag header

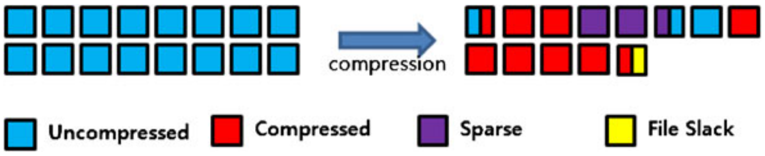


Fig. 14 State of the file compression

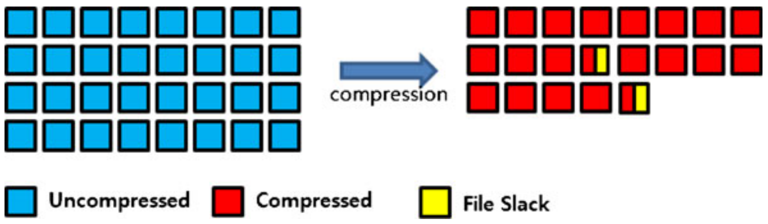


Fig. 15 Allocation state of the newly generated compressed file

(ten Bytes) to the residual bits, which are continuously calculated, except for the left first bit in each byte. Then, the carving is to be carried out using the same way as the ID3 Tag Ver.1 MP3 file.

3 NTFS conpression files carving

Since the NTFS version 5.0, it provides a compression function at a file system level. As the compression is executed at a file system level, it is not necessary to consider whether the subjective files are compressed in application programs [3].

As shown in Fig. 12, the configuration of the NTFS compression is implemented at the disk properties or at the window of the advanced attributes for files or folders as shown in Fig. 13.

Although the NTFS compression file is useful to increase the efficiency of storage space, it represents a difficulty in carving the file. In practice, most carving tools and studies do not consider such NTFS compression file. Therefore, in this research, we describe the basic concept of the NTFS compression and represent a carving method for the file. Also, we propose a practical application of this method to multimedia files.

3.1 NTFS compression

In the NTFS compression, it is not carried out for the entire files, but for the unit size of 16 clusters. Also, this unit is used to process the reading and writing of data. If the size of clusters is defined by 512 bytes, the compression unit is 8 KB (512 * 16) [5, 9].

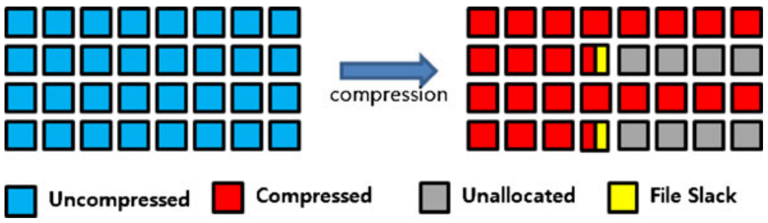


Fig. 16 Allocation state of the compression of normal files

Table 3 Header information of the compression block

Bits	Description
0–11	Compressed data size
12–14	Unknown
15	Data is compressed

The compression unit consists of several compression blocks. After implementing a compression process, as represented in Fig. 14, each compression block shows three different states, such as compressed data, uncompressed data, and data with sparse properties, according to the condition of original files. Also, it may represent file slacks at the end of the compression unit due to the compression [5, 9].

In the file compression policy in NTFS files, it can be varied by the case that newly generates files in the same storage space as the partition or folder, which applies compression properties, and the case that applies the compression for the files, which are already stored in a common partition.

In the case that newly generates compression files, the compression unit is continuously allocated as shown in Fig. 15 as it is allocated as a continuous manner.

In the case that applies the compression for the normal files, which are already stored, the compression is uniformly carried out with a constant interval with 16 clusters as illustrated in Fig. 16. Also, the data is not to be continuously stored due to the generation of unallocated space as much as the reduced data caused by the compression.

The NTFS compression uses an LZNTI algorithm based on the LZ77 algorithm. The LZNT1 algorithm implements the compression with the previously mentioned compression unit, and each compression block in the compression unit consists of a 2-byte header and several data groups. Table 3 shows the information of the header for each bit [5].

The first 12 bits in 16 bits represent the size of the compression block. The starting position of the next compression block can be determined by calculating the position of the present compression block + the size of the present compression block + 3. If the first two bytes in the next compression block is 0x0000, the position ranged from the next value to the end of the cluster is a file slack region. The last bit in 16 bits shows whether the data is compressed. As the subjective bit is one, the data stored in the compression block is compressed. In the case of the bit that is defined by 0, the data is not compressed. In the case of the data that is not compressed, the original data is to be stored without changes. If it is compressed, the data will be compressed according to the LZNT1 compression algorithm [5, 9].

Figure 17 illustrates the composition of the data group in the compression block.

As there is not data group in the compression block, the data consists of tag bytes and normal eight bytes as illustrated in Fig. 17. The data group located at the right side in

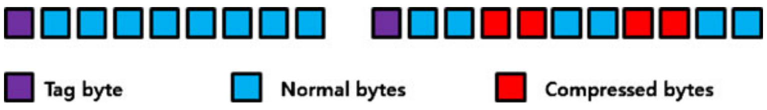


Fig. 17 Composition of the compression block data

```
00000000 (Bits) 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h'
00000000 (Bits) 'i' 'j' 'k' 'l' 'm' 'n' '₩n' 'o'
00000100 (Bits) 'p' 'q' 0x07 0x88 '₩n' 'r' 's' 't' 'u'
```

Fig. 18 Example of the storing of compressed data

Fig. 17 represents two compression bytes in the data group. A single compression byte is composed by two bytes. Therefore, an increase in the compression byte by one increases the size of the data group by one byte. The data group in Fig. 17 has a total of ten bytes because it has two compression bytes [5].

A tag byte plays a role in presenting the position of compression bytes in the data group. As the tag byte is considered as a bit unit, in the case of the bit value that is determined by one, the byte at the subjective position is the compression byte. The data group located at the right side in Fig. 17 represents the compression byte at the third and sixth bytes. Thus, the bit value of the tag byte is 0x00100100 (Bits) [5].

The LZNT1 compression implements a compression work that stores the offset and size of the subjective data as the previously stored data has duplicated contents.

Figure 18 represents an example of the storing of compressed data. The value of '00000000 (Bits)' in Fig. 18 shows the expression of one byte to bits and is a tag byte that is used as an index value, which identifies a compression byte in the last eight bytes. The compressed byte stores the offset and size of the compressed original data. As shown in Fig. 18, the third line, '00000100 (Bits)', shows the third byte, 0x8807, in the last eight bytes. By using it, it is possible to calculate the offset and size of the data [5].

The offset can be calculated by adding 1 after shifting the compressed data 11 times to the right side and multiplies −1 to the calculation. The size can be calculated by adding three after applying 0x07FF and AND with the compressed data. Figure 19 illustrates the results of the offset and size of the compressed data of 0x8807. Using the results of this calculation, it can be seen that the actual data of 0x8807 represents the data, which is moved by −18 from the present location, that is ten for the string of "abcdefghij".

As the compressed data in Fig. 18 is fully decompressed, it represents the string as shown in Fig. 20

3.2 Method for the NTFS compressed file carving

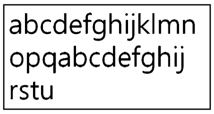
3.2.1 LZNT1 compression determine whether measures

It is necessary to process a procedure that identifies whether the subjective clusters are compressed for carving NTFS compressed files.

As mentioned in Section 3.1, the LZNT1 compression algorithm is composed by a compression unit with 16 clusters, and the compression unit consists of several compression blocks. The position of the next compression block can be identified using

Fig. 19 Calculation of the offset and size of compressed data

```
Offset : 0x8807 >> 11 => (-1) * (17 + 1) => -18
Size : 0x8807 & 0x07FF => 7 + 3 => 10
```



```
abcdefghijklmn
opqabcdefghij
rstu
```

Fig. 20 Decompressed strings

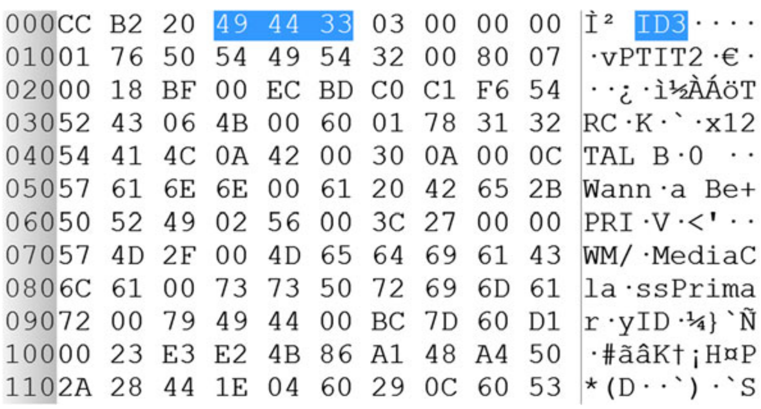
the size information in the header of the compression block in which the last compression block represents the value of 0x0000 at the header of the next compression block. Regarding that a sequential round in the header of the compression block is carried out by using this position, it can be regarded that the corresponding 16 clusters are normal data without the LZNT1 compression as the value of 0x0000, which represents the last compression block, does not exist and that exceeds the range of the compression unit.

In addition, in the case of normal compression blocks, the size information of the header of the compression block should represent the same value as the summation of the size of data groups in actual compression blocks. If the value is different, the corresponding compression unit can be regarded as normal data without the LZNT1 compression.

In the case of the data compression using the LZNT1 compression, the compression unit is to be generally determined by the size of 16 clusters. That is, the original data that is actually compressed has the size below 16 clusters. Therefore, the decompression for a normal compression file cannot represent more than 16 clusters. If the decompressed data shows more than 16 clusters, the data can be considered as normal data without compression.

3.2.2 Carving algorithm of the NTFS compressed file

For carving a NTFS compressed file, the comparison of file signature is to be possibly implemented. The first two bytes in the NTFS compressed file is the header of the compression block, and the third byte represents the index of the compressed data. Thus, the signature of the file is stored from the fourth byte.



```
000CC B2 20 49 44 33 03 00 00 00 Ì² ID3····
01001 76 50 54 49 54 32 00 80 07 ·vPTIT2·€·
02000 18 BF 00 EC BD C0 C1 F6 54 ··¿·ì½ÀÁöT
03052 43 06 4B 00 60 01 78 31 32 RC·K·`·x12
04054 41 4C 0A 42 00 30 0A 00 0C TAL B·0 ··
05057 61 6E 6E 00 61 20 42 65 2B Wann·a Be+
06050 52 49 02 56 00 3C 27 00 00 PRI·V·<'··
07057 4D 2F 00 4D 65 64 69 61 43 WM/·MediaC
0806C 61 00 73 73 50 72 69 6D 61 la·ssPrima
09072 00 79 49 44 00 BC 7D 60 D1 r·yID·¼}`Ñ
10000 23 E3 E2 4B 86 A1 48 A4 50 ·#ãâK†¡H¤P
1102A 28 44 1E 04 60 29 0C 60 53 *(D··`)·`S
```

Fig. 21 Compression block header and the file signature

```
Algorithm Carving_NTFS_Compressed

INPUT

          buffer : Disk Data Buffer
          file : Carved File Buffer
Output

          TRUE : Success
          FALSE : Fail
Begin

          if(NTFS_Compressed_Signature_Find) then
               Decompress
               if(Check_Complete_File)then
                    file <- Save_File
                         return TRUE;
               else
                    while(Get_Next_Compression_Unit)do
                         if(Is_Compressed) then
                             Decompress
                             if(Check_Complete_File)then
                                   file <- Save_File
                                   return TRUE;
                             else
                                   continue;
                             end if;
                         else
                             return FALSE;
                         end if;
                    end while;
               end if;
          end if;
          return FALSE;
End
```

```
Algorithm Get_Next_Compression_Unit

INPUT

          compression_buffer : Compression Unit Buffer
          flag : Compression Unit Acquire Policy
Output

          TRUE : Success
          FALSE : Fail
Begin

          switch (flag)
               case :
                    compression_buffer <- save
                    return TRUE;
               case :
                    compression_buffer <- save
                    return TRUE;
               default :
                    return FALSE;
          end switch;
End
```

Fig. 22 Carving algorithm for the NTFS compressed file

Figure 21 shows the header of the compression block and the file signature. Also, it is possible to identify the two-byte compression block header and the signature of a MP3 file with 0x494443. It is difficult to identify the duplication of the first data in the compression block. Also, because the signature does not apply duplicated values, the data is not compressed but stored. Thus, in a carving process, the start position of the compressed file can be identified by checking the signature from the fourth byte. If the signature of a compressed file is searched, the original data can be obtained by decompressing the corresponding compression block using the LZNT1 algorithm.

Compressed file uses a 16-cluster compression unit, and the compression unit cannot be separated. That is, it is possible to exactly recover the file using a file carving process because the compressed file below 16 clusters is not separated.

In the continuous allocation of the NTFS compressed files with more than 16 clusters, the header of the file can be searched by checking the signature from the fourth byte as the same way as the carving of the compressed files with below 16 clusters. As the header of the file is searched, it is necessary to identify the LZNT1 compression of the next 16 clusters after decompressing the file. If the corresponding clusters are compressed, it can be identified that whether the data is composed by a single perfect file through connecting it to the previously obtained data after decompressing the compressed file. For implementing this work, a general carving method that compares the footer of a file is used. In the results of the comparison, as the file is not a perfect figure, the file recovering is carried out by repeating the comparison using the next 16 clusters until it represents a perfect file. If it is identified as a none-compressed cluster during the comparison, the carving will be halted due to the fact that the data of the corresponding file is vanished during the process (Fig. 22).

In the case of multimedia files, the data can be reproduced even though the entire data are not presented. Thus, it is possible to recover the data up to the section in which the format of multimedia files is verified using the carving method as mentioned above even though all data are not perfectly presented.

As mentioned in Section 3.1, the file allocation policy shows a different way for the new generation of files in the partition where the compression is applied as the same as the compression of the normal files, which are already stored. Therefore, it is necessary to consider two different cases that verify both the next clusters after the present compressed clusters and the clusters after the 16 clusters when the verification of the compression of data is carried out through moving it to the next clusters.

By using this method, it is possible to implement the carving for the continuously allocated compressed files with more than 16 clusters. Figure 12 shows the pseudo-code for the carving algorithm of the NTFS compressed file. Whereas, it did not consider the carving for the discontinuously allocated compressed file more than 16 clusters.

4 Experiments

We compared the multimedia file carving results of proposed method from this paper and commercial file carving tools. Next tables show AVI, WAV, MP3 file carving result Tables 4, 5, 6.

First column of each table represents the capacity and used capacity of each hard disk. And the rest of column data represents the number of recovered files, the file

Table 4 AVI files carving result comparison

Disk\carving tool	The proposed method	Carving tool A	Carving tool B
Disk A (143/250 GB)	9 (9–100%)	12 (8–66.6%)	9 (7–77.7%)
Disk B (231/500 GB)	11 (9–81.8%)	16 (9–56.2%)	11 (7–63.6%)
Disk C (76/100 GB)	5 (5–100%)	7 (5–71.4%)	7 (5–71.4%)

Table 5 WAV files carving result comparison

Disk\carving tool	The proposed method	Carving tool A	Carving tool B
Disk A (143/250 GB)	34 (31–91.1%)	43 (31–72%)	36 (25–69.4%)
Disk B (231/500 GB)	49 (41–83.6%)	67 (41–61.1%)	52 (32–61.5%)
Disk C (76/100 GB)	29 (27–93.1)	37 (27–72.9%)	31 (22–70.9%)

number and percentage of normal execution. Recovered files number of the proposed method from this paper less than commercial file carving tools. But the number of normal executed file is same and rate is high. Therefore, recovered file verification time is reduced. And, after delete the NTFS compression multimedia files, we performed a file carving. As a result, if using proposed method from this paper, we confirmed that deleted NTFS compression files are recovered. But, the commercial file carving tools did not recover the files.

5 Conclusion

In this research, we proposed a carving method for multimedia files. Also, we proposed a carving method for the NTFS compression file and its practical application method for multimedia files. According to the increase in the performance of computers and in the storage capacity of hard disks, storing user's data as video and audio files has been increasing in recent years. Therefore, it can be considered that the importance of such multimedia files has also been increased. The method proposed in this study is able to perfectly recover AVI, WAV, and MP3 files as the files are continuously allocated. In addition, although the files are discontinuously allocated or the entire data of the files are not allocated, it can be verified by recovering the data, which are continuously presented, only due to the characteristics of multimedia files.

In addition, in the case of the multimedia files that are stored as the NTFS compression format, this study proposed a carving method using the characteristics of the NTFS compression file. Although the NTFS compression represents an efficient way to use storage space, it shows a difficulty in obtaining deleted data. However, it is possible to recover the average files and multimedia files that are stored as the NTFS compression format using the method proposed in this study.

Table 6 MP3 files carving result comparison

Disk\carving tool	The proposed method	Carving tool A	Carving tool B
Disk A (143/250 GB)	121 (121–100%)	145 (121–83.4%)	112 (98–87.5%)
Disk B (231/500 GB)	23 (23–100%)	25 (23–92%)	17 (15–88.2%)
Disk C (76/100 GB)	0 (0–100%)	0 (0–100%)	0 (0–100%)

5.1 Future work

We are study on the carving method for other multimedia files such as MPEG(Moving Picture Experts Group), WMV(Windows Media Video), FLV(Flash Video). And we will develop a carving tool using multimedia files carving method.

Acknowledgment This work was supported by the IT R&D program of MKE/KEIT[10035157, Development of Digital Forensic Technologies for Real-Time Analysis].

References

1. AVI File Format, http://www.alexander-noe.com/video/documentation/avi.pdf
2. AVI RIFF File Reference, http://msdn.microsoft.com/en-us/library/ms779636(VS.85).aspx
3. Brian Carrier, File System Forensic Analysis, Addison-Wesley Professional, 22 March 2005
4. Garfinkel SL (2007) Carving contiguous and fragmented files with fast object validation. Digital Invesigation 4(1)
5. Joachim Metz, “carving NTFS-compressed files”, Hoffmann Investigations, 2 September 2009, http://www.forensicfocus.com/carving-ntfs-compressed-files
6. Library of Congress, “WAVE Audio File Format”, 6 December 2009
7. Mikus NA (2005) An analysis of disc carving techniques, Master's thesis. Naval Postgraduate School, March 2005
8. MP3 File Format, http://mpgedit.org/mpgedit/mpeg_format/mpeghdr.htm
9. Sanderson P (2000) NTFS compression—a forensic view, Sanderson Forensics, October 2002
10. WAVE File Format, http://web.archive.org/web/19991115123323/http://www.borg.com/~jglatt/tech/wave.htm

Byeongyeong Yoo received his B.S. degree in Computer Science from Sangmyung University, He is now studying master course in Graduate School of Information Management and Security, Korea University. He is currently working for Digital Forensic Research Center in Korea University. He has performed projects related to file system analysis and file carving. His research interests are digital forensics, file system analysis, file carving.

Jungheum Park is a doctoral student in Graduate School of Information Management and Security at Korea University and a senior researcher of Digital Forensic Research Center in Korea University since 2009. He has performed projects related to data carving, registry analysis, electronic document analysis and anti-anti forensics. His research interests are digital forensics, file carving and anti-anti forensics.

Sungsu Lim received his B.S. degree in Electronics engineering from Yonsei University, He is now studying master course in Graduate School of Information Management and Security, Korea University. He is currently working for Digital Forensic Research Center in Korea University. He has performed projects related to database forensics, file carving and forensic corpora. His research interests are digital forensics, virtualization, file carving, database forensics.

Jewan Bang is a doctoral student in Graduate School of Information Management and Security at Korea University and a senior researcher of Digital Forensics Research Center in Korea University since 2009. He has performed projects related to Embedded system forensics. His research interests are digital forensics, data recovery and reverse engineering.

Sangjin Lee Received his Ph.D. degree from Korea University. He is now a Professor in Graduate School of Information Management and Security at Korea University and the head of Digital Forensic Research Center in Korea University since 2008. He has published many research papers in international journals and conferences. He has been serving as chairs, program committee members, or organizing committee chair for many domestic conferences and workshops. His research interests include digital forensic, steganography, cryptography and cryptanalysis.