Digital Investigation 18 (2016) 1–7

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Integrity verification of the ordered data structures in manipulated video content

Jieun Song [a], Kiryong Lee [a], Wan Yeon Lee [b], Heejo Lee [a, *]

[a] Korea University, South Korea
[b] Dongduk Women's University, South Korea

ARTICLE INFO

Article history:
Received 12 November 2015
Received in revised form 31 May 2016
Accepted 6 June 2016
Available online 16 June 2016

Keywords:
Digital forensics
Data structure
Video forgery
Integrity verification

ABSTRACT

Video content stored in Video Event Data Recorders (VEDRs) are used as important evidence when certain events such as vehicle collisions occur. However, with sophisticated video editing software, assailants can easily manipulate video records to their advantage without leaving visible clues. Therefore, the integrity of video content recorded through VEDRs cannot be guaranteed, and the number of related forensic issues increases. Existing video integrity detection methods use the statistical properties of the pixels within each frame of the video. However, these methods require ample time, because they check frames individually. Moreover, the frame can easily be replaced and forged using the appropriate public software. To solve this problem, we propose an integrity checking mechanism using the structure of ordered fields in a video file, because existing video editing software does not allow users to access or modify field structures. In addition, because our proposed method involves checking the header information of video content only once, much less detection time is required compared with existing methods that examine the entire frames. We store an ordered file structure of video content as a signature in the database using a customized automated tool. The signature appears according to the video editing software. Then, the suspected video content is compared to a set of signatures. If the file structure matches with a signature, we recognize a manipulated video file by its corresponding editing software. We tested five types of video editing software that cover 99% of the video editing software market share. Furthermore, we arranged 305,981 saving options for all five video editing suites. As a result, we obtained 100% detection accuracy using stored signatures, without false positives, in a collection of 305,981 video files. The principle of this method can be applied to other video formats.

Introduction

Vehicle accidents are an unavoidable part of our daily lives. When we become involved in a vehicle accident, there is no doubt that one of the most important problems, with the exception of saving lives, is clarifying responsibility for the accident, including any legal issues. In the last decade, Video Event Data Recorders (VEDRs) have been used as effective and trustworthy witnesses to vehicle accidents and even other criminal incidents. VEDRs are also known as Car Dashboard Camcorders, Vehicle Road Dash Video Camera, Car Black Boxes, and Driving Recorders. In this paper, we refer to such devices collectively as VEDRs. The installation and use of VEDRs is increasing rapidly, and there are a number of countries that mandate the installation of such devices. However, sophisticated video editing software makes it easy to manipulate video content without leaving visible clues. Therefore, it is not possible to be entirely convinced of the integrity of video content

* Corresponding author.
E-mail address: heejo@korea.ac.kr (H. Lee).

http://dx.doi.org/10.1016/j.diin.2016.06.001
1742-2876/

recorded by VEDRs (Poisel and Tjoa, 2011; Lee et al., 2015). Some efforts have been directed at ensuring the integrity of video content in digital images and video files in advance (Hu et al., 2015). However, such technologies have a limitation wherein the video recorder system must be pre-processed when it is established, for example, by calculating hash values, embedding watermarks into the video, and so on. Therefore, digital video forensics has become a popular technique for video integrity verification without pre-registration or pre-embedded information; this process is referred to as passive-blind video forensics.

Existing methods mainly focus on digital still images. Zheng used the statistical anomalies of each pixel in an image (Zheng et al., 2015; Rad and Wong, 2015). Qu used the statistical characteristics of image compression (Qu et al., 2014; Cao et al., 2014). Geradts checked the trace caused by a camera color filter and charged-couple device (CCD) (Geradts et al., 2001). Carvalho examined the geometric relationship between an object and light (Carvalho et al., 2015). The principles behind these methods can be applied to video content. However, such evidence can easily be replaced and forged in the image source using publicly available software (Sencar and Memon, 2008). In addition, existing methods must check entire frames in order to detect video forgeries (Wang and Farid, 2007). On an average, 1 min of video content contains 1800 frames, and thus these methods must repeatedly check all 1800 frames. If the video content has low resolution, the detection rate for these methods is reduced significantly.

To address these problems, we propose a mechanism for detecting the modification of video content with video editing software. We focus on the ordered data structures in a video file for integrity verification. The internal ordered data structures are particularly valuable and contain distinct information on video editing software authentication. Existing video editing software and metadata editors do not modify the file's data structures (Gloe, 2012). Therefore, data structure information is reliable for checking video integrity. In addition, this method requires relatively little time for detection, because it simply checks the header information of video content a single time. Moreover, this method is not affected by the resolution, because it checks the data header information exclusively.

We investigate the video file structure characteristics for each type of video editing software that would leave traces from processing the video editing software. Because such traces are an inherent characteristic of each respective video editing software suite, we can detect the specific video editing software used to manipulate the video, in addition to whether the video was, indeed, manipulated. To evaluate the accuracy of this technique, we examined 296 unmodified Audio Video Interleave (AVI) video files. We performed this examination using popular versions of video editing software, namely, Adobe Premiere CS3, Adobe Premiere CS4, Adobe Premiere CS5, Adobe Premiere CS6, Adobe Premiere CC, Sony VEGAS 9, Sony VEGAS 10 Sony VEGAS 11, Sony VEGAS 12, Sony VEGAS 13, Edius 6, Edius 7, Avid Media Composer (Avid MC) 5, Avid MC 6, and Avid Studio 1. These software suites comprise 99% of the video editing software market share. (Notably, we excluded Final Cut, because it does not support the AVI format without an additional plugin.) Although the same video editing software is used in certain cases, depending on the rendering option, the files can appear to have different ordered data structures. Therefore, we arranged 305,981 saving options for all five video editing software suites.

As a result, we found that the AVI data structures in modified video files appear consistently according to each video editing software suite. Each resulting data structure is unaffected by the original video file structure. These ordered data structures are stored in a database as the signature of each type of video editing software. Moreover, given the need to include other video editing software, the database can be extended easily. We used our own customized file parser to read, extract, and detect video content. As a result, we obtained 100% detection accuracy using the stored signatures, with no false negatives (FNs) or false positives (FPs) in our experimental environments. Using the proposed method, we can check the integrity of suspected video, and we can also find the video editing software used for the manipulation.

The main contribution of this paper is the proposal of a method for detecting the integrity violator of video content. To the best of our knowledge, this work is the first attempt to generate a video editing software signature database. Our method does not require pre-processing, and its detection accuracy is not affected by the resolution.

In the following section, we explain the general AVI file format structures. Then, we introduce the practical test setup and explain the proposed signature algorithm. Based on this method, we analyze the original video contents and store video editing software specifications into a database. Using this database, we then describe our evaluation of its detection accuracy. Finally, we summarize our investigation and provide a discussion of the experimental results.

Background

The majority of the multimedia formats used in VEDRs are AVI files. The second most common format is MP4. Only a few test devices use WMA containers. Nevertheless, the proposed method can be applied for AVI, MP4 and WMA formats. Without loss of generality, we discuss the case of the AVI format in this paper. For clarification, we explain the general AVI file format structure.

General AVI file format structure

AVI is a multimedia container format introduced by Microsoft in November 1992. AVI files can contain audio and video data. A ″chunk” refers to a fragment of information that is used in many multimedia formats, such as MP3 and AVI. Each chunk is identified by a four-byte delimiter. The four-byte delimiter is called a Four-Character Code (FourCC) tag. This tag is used to identify the stream type, data chunks, index entries, and other information. An AVI file takes the form of a single chunk in an RIFF formatted file, which is then subdivided into chunks. Every chunk has the following basic structure (see Fig. 1): the ChunkID contains four ASCII characters that identify the chunk field name; the ChunkSize is the length of the data stored in the ChunkData field, excluding any padding areas;

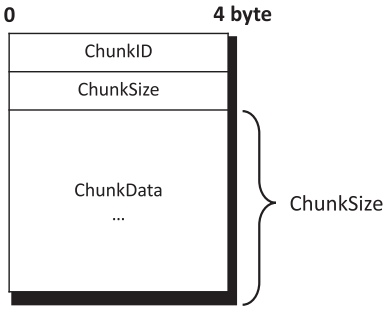


Fig. 1. The basic structure of a chunk.

the ChunkData is zero or several bytes of data. The data is always padded to the nearest WORD boundary. If the data is of an odd length in the ChunkSize, an extra byte of NULL padding is added to the end of the data. Subchunks have the same structure as chunks. A subchunk is simply any chunk contained within another chunk.

Fig. 2 illustrates the general AVI file structure, which consists of four chunks: the RIFF, which contains the AVI file type identifier; the header list (hdrl), which contains the metadata of the file; the movie recorder list (movi), which contains the data stream of the video content; and the index data (idx1), which contains the index of each video frame. There is no strict order to this sequence, and variable optional chunks (e.g., JUNK) can occur (Microsoft Developer Network).

Video editing software

We collected 232 original videos from an internet VEDR community site,[1] along with 64 videos from a forensic center. We classified them according to a sequence of field data structure types.

For each original video field data structure type, video editing software was used to manipulate the video content. We examined the videos using five types of video editing software, representing 99% of the video editing software market share (viz., Sony VEGAS, Adobe Premiere, Edius, Avid MC, and Avid Studio). However, even when using the same video editing software, different field data structures can appear according to the saving option. Therefore, we examined all available saving options. Table 1 lists the number of saving options for each type of video editing software. For example, in the case of Adobe Premiere Pro CC, we can select among 2 types of video formats, 2 filter options, 15 video codecs, 3 field types, 2 depths, 9 sample rates, and 5 sample types. Consequently, considering the number of cases for the video rendering options, we tested $2 \times 2 \times 15 \times 3 \times 2 \times 9 \times 5 = 16{,}200$ options. The total saving options for all five types of video editing software is thus 305,981. For efficiency and database expandability, we then developed our own file parser to generate the database and detect manipulated video content. The computer environment for our experiment was set to the minimum requirements for the video editing software. We

[1] The video content used in our experiment can be found at http://cafe.naver.com/blackboxclub.

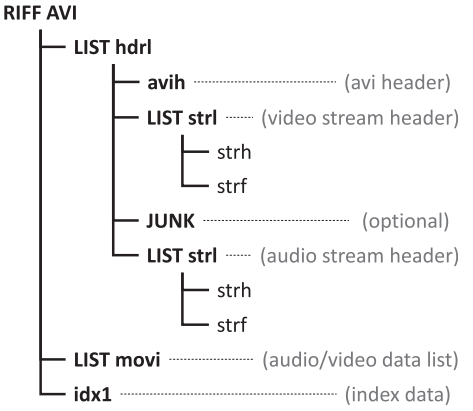


Fig. 2. General AVI file structure.

Table 1
Rendering options.

Software	Version	# Of rendering options
Adobe Premiere	CS3	16,200
	CS4	24,300
	CS5	18,225
	CS6	32,400
	CC	16,200
Sony VEGAS	9	24,576
	10	24,576
	11	24,576
	12	32,768
	13	32,768
Avid MC	5	3072
	6	2048
	7	1024
Edius	6.5.2	8192
	7	32,768
Avid Studio	1.1	12,288
Total	16	305,981

experimented in Windows 7 (64-bit version) with 4 GB of memory, after installing the Windows 7 Service Pack 1 and the QuickTime codec.

Automated signature collection method

We analyzed hundreds of thousands of video content samples. Therefore, for efficiency, we created a customized file parser. This parser extracts field structures and stores them into the database as a signature, which makes it easy to extend the database. We extracted field data structures from manipulated video content using our customized parser. If the extracted field data structure is new, we store it, as shown in Fig. 3. This principle was used while collecting original VEDR field data structures.

Algorithm 1. Script for generating the database.

```
procedure MAKE_DATABASE(fabricated files by tools)
    while file in fabricated files do
        signature.empty()                    // reset signature
        while file != EOF do
            read(file, type, 4)              // read 4 bytes for chunk type
            read(file, size, 4)              // read 4 bytes for chunk size
            signature.add(type)              // add type to signature
            file = file + size               // to jump a next chunk
        database.add(signature)              // add the signature to database
```

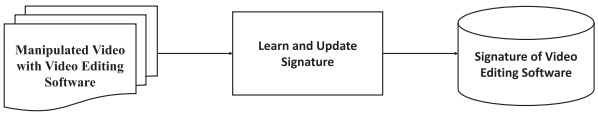


Fig. 3. Signature database generating process.

Algorithm 1 shows the pseudocode for the process of storing a signature in the database. First, we read four bytes for the chunk type from the video content, and read the next four bytes for the chunk size. This way, the process reads all of the chunks, and then adds the signature to the database. This parser can easily collect field data structures and add new signatures.

Existing methods require considerable time, because they check image frames individually. For example, when a 60-second video is recorded at 30 frames per second, existing methods must investigate $30 \times 60 = 1800$ frames. However, the proposed mechanism requires far less time, because it exclusively reads the header information of the video file's metadata.

Video editing software observation results

There were 12 different AVI field data structures in the original 296 video contents, as shown in Table 2. In order to generate manipulated video content, we manipulated 12 types of original video content. We applied all 12 types to one type of video editing software. The resulting field data structure is not affected by that of the original video content. Therefore, we manually tested 305,981 options in a single file. To improve the rendering speed, we used a small AVI file. As a result, all video editing software suites left distinct traces. That is, no video editing software was able to exactly duplicate the field data structure of the original VEDR or any other video editing software signatures.

Table 2
12 types of AVI field structures in the 296 videos.

	Sequence of AVI metadata fields
1	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl auds JUNK odml LISTINFO INAM JUNK LISTmovi idx1
2	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl auds JUNK odml LISTINFO ISFT JUNK LISTmovi idx1
3	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl auds LISTodml JUNK LISTmovi idx1 LISTINFO ISFT
4	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl vids LISTstrl auds LISTstrl txts LISTmovi idx1
5	RIFFAVI LISThdrl avih LISTstrl vids JUNK odml LISTINFO ISFT JUNK LISTmovi idx1
6	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl auds JUNK LISTmovi idx1 SLLT SGLT
7	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl auds LISTstrl txts LISTmovi idx1
8	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl vids LISTstrl txts LISTmovi idx1
9	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl auds JUNK LISTmovi idx1
10	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl auds JUNK LISTmovi
11	RIFFAVI LISThdrl avih LISTstrl vids LISTstrl auds LISTmovi idx1
12	RIFFAVI LISThdrl avih LISTstrl vids JUNK LISTmovi idx1

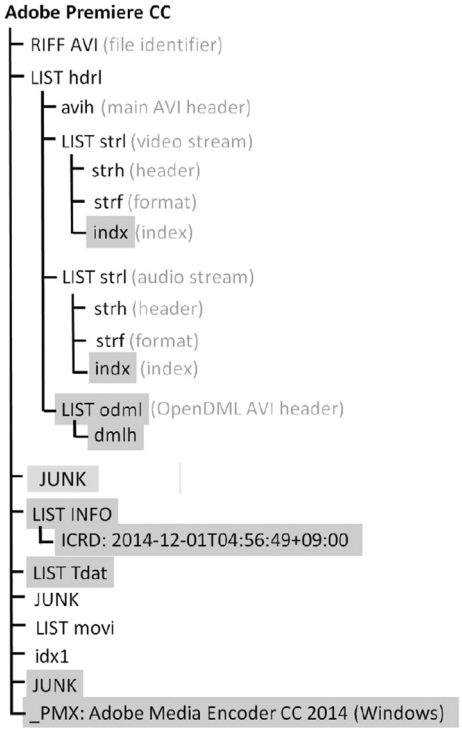


Fig. 4. AVI file structure after editing with Adobe Premiere CC.

One example file structure after editing the original video with Adobe Premiere CC is shown in Fig. 4. The familiar tree-like graphs depict AVI file structures after editing with Adobe Premiere CC. Adobe Premiere CC adds another LIST with OpenDML AVI header data, indx with AVI index data, and ICRD with a modified time. In addition, a list PMX with details about the employed encoder software and version (here: Adobe Media Encoder CC 2014 Windows) is present in videos edited with Adobe Premiere CC. Furthermore, the PMX contains the overall information about the modified video (the horizontal and vertical size of the video, modified time, etc.). The PMX video information is the same in the video stream and the LIST INFO.

Table 3 lists an example of the results modified with Adobe Premiere CC. The original video was modified using

Table 3
Differences in video streams between pre-edited AVI files and AVI files edited with Adobe Premiere CC.

Items	Values (pre-editing)	Values (post-editing)
Type	video (vids)	video (vids)
Codec	H264	**UYVY**
Flags	0	0
Priority	0	0
Language	0	0
Initial Frames	0	0
Scale	33,333	**1**
Rate	1,000,000	**25**
Start of Stream	0	0
Length	720	**523**
Buffer Size	0	**691,200**
Quality	−1	**0**
Sample Size	0	0

Table 4
Sequence of AVI metadata fields acquired with video editing software.

Software	Version	Sequence of AVI metadata fields
Adobe Premiere	CS3	RIFFAVI LISThdrl avih LISTstrl vids strf indx LISTodml dmlh JUNK JUNK LISTmovi idx1 JUNK PrmL Cr8r PrmA PARf _PMX LISTTdat
		RIFFAVI LISThdrl avih LISTstrl vids strf PrmL Cr8r PrmA PARf LISTTdat JUNQ LISTmovi idx1 _PMX
	CS4	RIFFAVI LISThdrl avih LISTstrl vids strf LISTstrl auds strf Cr8r LISTTdat JUNQ LISTmovi idx1 _PMX
		RIFFAVI LISThdrl avih LISTstrl vids strf LISTstrl auds strf JUNQ LISTmovi idx1 _PMX
	CS5	RIFFAVI LISThdrl avih LISTstrl vids strf LISTstrl auds strf Cr8r LISTINFO ICRD LISTTdat _PMX CS5 JUNK LIST movi idx1
		RIFFAVI LISThdrl avih LISTstrl vids strf indx LISTstrl auds strf indx LISTodml dmlh JUNK Cr8r LIST INFO ICRD LISTTdat JUNK LISTmovi idx1 JUNK _PMX CS5
	CS6	RIFFAVI LISThdrl avih LISTstrl vids strf LISTstrl auds strf Cr8r LISTINFO ICRD LISTTdat JUNK LISTmovi idx1 _PMX CS6
		RIFFAVI LISThdrl avih LISTstrl vids strf indx LISTstrl auds strf indx LISTodml dmlh JUNK Cr8r LISTINFO ICRD LIST Tdat JUNK LIST movi idx1 JUNK _PMX CS6
	CC	RIFFAVI LISThdrl avih LISTstrl vids strf LISTstrl auds strf LISTINFO ICRD LISTTdat _PMX JUNK LIST movi idx1
		RIFFAVI LISThdrl avih LISTstrl vids strf indx LISTstrl auds strf indx LISTodml dmlh JUNK LISTINFO ICRD LISTTdat JUNK LISTmovi idx1 JUNK _PMX
Sony VEGAS	9	RIFFAVI LISThdrl avih LISTstrl vids strf indx LISTstrl auds strf JUNK indx LISTodml dmlh JUNK LISTmovi idx1 JUNK LISTINFO TCOD
		RIFFAVI LISThdrl avih LISTstrl vids strf LISTstrl auds strf JUNK LISTmovi idx1 LISTINFO TCOD
	10, 11, 12, 13	RIFFAVI LISThdrl avih LISTstrl vids strf indx LISTstrl auds strf JUNK indx LISTodml dmlh JUNK LISTVDEX LISTINFO TCOD JUNK LISTmovi idx1 JUNK
		RIFFAVI LISThdrl avih LISTstrl vids strf LISTstrl auds strf JUNK LISTmovi idx1 LISTINFO TCOD LISTVDEX
Avid MC	5, 6, 7	RIFFAVI LISThdrl avih LISTstrl vids strf strn JUNK LIST movi idx1
Edius	6.5.2	RIFFAVI LISThdrl avih LISTstrl vids strf indx LISTstrl auds strf indx LISTodml dmlh JUNK LISTmovi idx1
	6.5.2, 7	RIFFAVI LISThdrl avih LISTstrl vids strf JUNK LISTstrl auds strf JUNK JUNK odml JUNK LISTmovi idx1 PrmA PARf LISTTdat LISTCtEx LISTCdat CtsI
Avid Studio	1.1	RIFFAVI LISThdrl avih LISTstrl vids strf indx LISTstrl auds strf JUNK indx LIST odml dmlh LISTmovi idx1

Adobe Premiere CC. As a result, the codec changed to UYVY, and the scale, rate, length, buffer size, and quality were changed to 1, 25, 523, 691, 200, and 0, respectively. Even if the value of the input video content is different, it has the same result value. This shows that the resulting field data structure and value depend on the video editing software.

Table 4 lists the sequence for each type of video editing software used in the study. A total of 305,981 configurations of manipulated video are summarized in 18 signatures, and each configuration is unique. Analyzing our new artifacts allows us to correctly distinguish between authentic and forged video content. Therefore, this database can be used for the verification of integrity. Moreover, we based our detection method on this database. We introduce the detection algorithm and evaluate its detection rate in the following section.

Detection algorithm and evaluation

Using the video editing software signature database, we can detect manipulated video content. In order to automate such detection, we designed an automated detection parser.

Algorithm 2. Script for verifying the integrity of video.

```
procedure CHECK_FILE(suspicious file)
    signature.empty()          // reset signature
    while suspicious file != EOF do
        read(file, type, 4)    // read 4 bytes for chunk type
        read(file, size, 4)    // read 4 bytes for chunk size
        signature.add(type)    // add type to signature
        file = file + size     // to jump a next chunk
    while db in database do
        if signature == db then
            return false       // If matched, suspicious file is fabricated
    return true                // If not matched, suspicious file is normal
```

Algorithm 2 shows the pseudocode for the process of detecting integrity based on our database. The basic principle to this detection algorithm is the same as the principle for the database algorithm. It reads four bytes for the chunk name from suspicious video content. Then, it reads the next four bytes for the size. Therefore, the algorithm reads all field lists and compares them with the database. If the field structure for the suspicious video content matches, we determine that the video content has been manipulated or that the video content is not the original. The procedure for examining the integrity of a video file is shown in Fig. 5.

Fig. 6 shows the interface for the automated detection tool that applies these principles. The image on the left represents the original video content, and the image on the right represents the video content modified with Adobe Premiere CC. We tested our dataset, and the results show 100% detection accuracy, with no FNs or FPs, as shown in

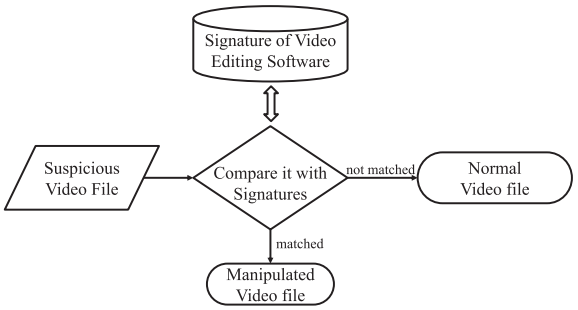
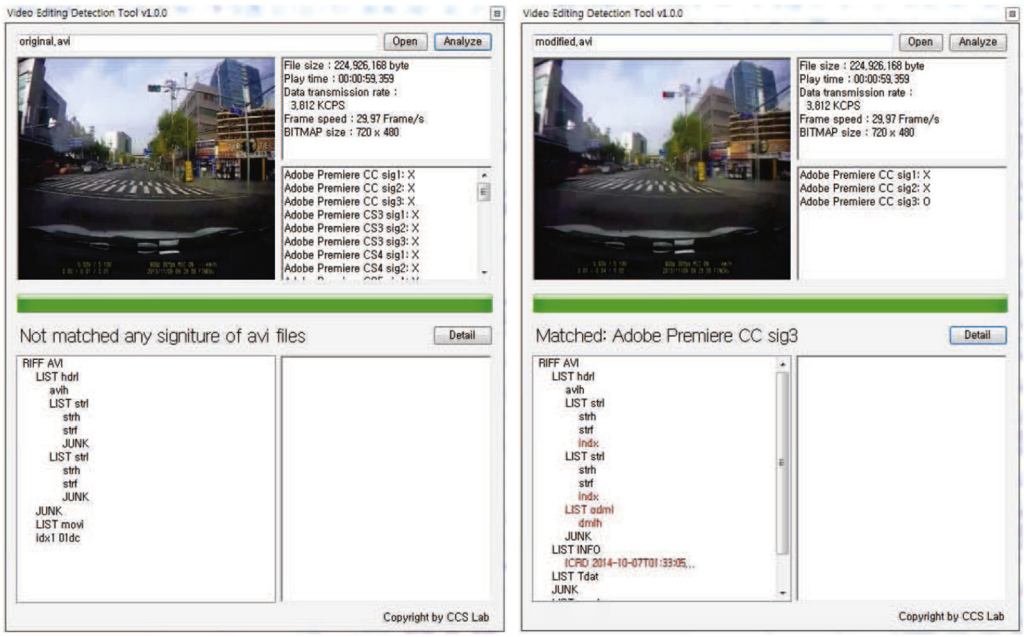


Fig. 5. Integrity detection scheme.



(a) Prove to be an original (b) Prove to be a manipulation

Fig. 6. Integrity detection scheme.

Table 5
Video editing tools covering 99% of the video editing tool market share (According to the report "Analysis of the Global Video Nonlinear Editing Market", Nov, 2012).

	Input	Detection rate (%)		Input	Detection rate (%)
Original	296	100	Vegas12	32,768	100
CS3	16,200	100	Vegas13	32,768	100
CS4	24,300	100	Avid MC5	3,072	100
CS5	18,225	100	Avid MC6	2,048	100
CS6	32,400	100	Avid MC7	1,024	100
CC	16,200	100	Edius6	8,192	100
Vegas9	24,576	100	Edius7	32,768	100
Vegas10	24,576	100	Avid Studio	12,288	100
Vegas11	24,576	100			

Table 5. The automated detection tool is available on the web site specified in the footnote[2]

In existing methods that use the statistical properties of the pixels, when analyzing a low-resolution image, the detection rate is reduced significantly. VEDRs or CCTVs can record dark roads or dynamic images. In such cases, there is less resolution, and this degrades the detection rate. However, the proposed method uses only the file structure information, and is thus unaffected by the resolution.

Discussion

In addition to AVI, we have applied the same approach to MP4 media format. We collected three different types of MP4 field data structures (Table 6). We manipulated three types of original video contents by Adobe Premiere series. As a result, we have constructed the signature from the field data structures of the content by Adobe Premiere series as shown in Table 7. While the proposed method can be applicable for detecting Adobe Premiere series as a working example, extension to other video formats can be considered as our future study.

Table 6
3 types of original MP4 field structures.

	Sequence of MP4 metadata fields
1	ftyp mdat moov mvhd iods trak tkhd mdia mdhd hdlr minf vmhd dinf dref stbl stsd stts stsz stsc stco stss trak tkhd mdia mdhd hdlr minf smhd dinf dref stbl stsd stts stsz stsc stco
2	ftyp mdat moov mvhd iods trak tkhd mdia mdhd hdlr minf vmhd dinf dref stbl stsd stts stsz stsc stco stss
3	ftyp mdat moov mvhd iods trak tkhd mdia mdhd hdlr minf smhd dinf dref stbl stsd stsz stts stsc stco trak tkhd mdia mdhd hdlr minf vmhd dinf dref stbl stsd stsz stts stsc stco stss trak tkhd mdia mdhd hdlr minf nmhd dinf dref stbl stsd stsz stts stsc stco

Summary and conclusion

As video editing software advances, malicious individuals can easily modify VEDR video content without leaving visible clues. Therefore, it is important to verify the integrity of videos used as evidence in criminal investigations. Existing methods require considerable detection time, and low-resolution images can degrade the detection rate.

[2] Video samples and the detection tool can be found at http://ccs.korea.ac.kr/vedit.

Table 7
Sequence of MP4 metadata fields acquired with Adobe Premiere editing software.

Software	Version	Sequence of MP4 metadata fields
Adobe Premiere	CS4	ftyp moov mvhd trak tkhd edts elst mdia mdhd hdlr minf vmhd dinf dref stbl stsd stts ctts stss sdtp stsc stsz stco trak tkhd edts elst mdia mdhd hdlr minf smhd dinf dref stbl stsd stts stsc stsz stco
	CS5	ftyp moov mvhd iods trak tkhd mdia mdhd hdlr minf vmhd dinf dref stbl stsd stts stsc stsz stco stss edts elst udta uuid mdat
	CS6	ftyp moov mvhd trak tkhd edts elst mdia mdhd hdlr minf vmhd hdlr dinf dref stbl stsd stts stss sdtp stsc stsz stco ctts trak tkhd edts elst mdia mdhd hdlr minf smhd hdlr dinf dref stbl stsd stts stsc stsz stco udta uuid
	CC	ftyp moov mvhd trak tkhd edts elst mdia mdhd hdlr minf vmhd hdlr dinf dref stbl stsd stts stss sdtp stsc stsz stco ctts trak tkhd edts elst mdia mdhd hdlr minf smhd hdlr dinf dref stbl stsd stts stsc stsz stco udta uuid free

For higher efficiency and faster detection, we proposed a detection algorithm that can easily collect signatures into a database and detect manipulated video content. In this paper, we focused on the AVI file format, which is one of the most popular formats. We examined videos from 296 video files acquired with 43 VEDRs and manipulated with 5 different video editing software suites. In addition, we arranged 305,981 saving options for these 5 software suites. As a result of our analysis, the field data structure characteristics provided valuable clues for checking video integrity and verifying the authenticity of video content. Our approach remains effective even with videos posted online or with frames removed manually. However, some online sites (e.g., YouTube) apply a re-rendering process when users upload videos.

Our main findings can be summarized as follows:

- Different video editing software suites have unique field structures. Therefore, by analyzing the field structures, we can determine whether video editing software was used to manipulate video.
- The AVI format does not have a strict standard. There is a wide variety of field structures in the order and type of data structure in AVI files.
- Manipulating video content changes the data structure of the video according to the video editing software and saving options. The structure of the edited video content is consistent regardless of the original field structure.
- Depending on the codec, even were a user to select identical options upon saving an edited video, the data structure would nevertheless remain unique.

Using our customized parser, we can automatically check the integrity of a video in order to determine whether video editing software was used. With the proposed method, it easy to expand the database and detect manipulated video content. The parser simply checks the field structures, thus ensuring efficiency and accuracy.

Malicious individuals might attempt to bypass this method by manipulating video content in a particular manner. However, this will prove challenging. To bypass this method, the malicious individuals need to know about hex code. Moreover, even were they to manipulate the hex code, video content will not play after simply changing the file's structure. In order to play the manipulated video, such individuals would require a comprehensive understanding of video formats.

The proposed method can detect the integrity of video using ordered data structures. However, applying our method to other formats requires added considerations, and we shall pursue this in future research. Moreover, we will apply this principle to a method for detecting the type of video recording device. Such a method can be applied to integrity verification and device authorization.

Acknowledgments

This research was supported by the Public Welfare & Safety Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2012M3A2A1051118).

References

Cao G, Zhao Y, Ni R, Li X. Contrast enhancement-based forensics in digital images. Inf Forensics Secur IEEE Trans 2014;9(3):515–25.
Carvalho T, Farid H, Kee E. Exposing photo manipulation from user-guided 3d lighting analysis. In: IS&T/SPIE electronic imaging, international society for optics and photonics; 2015. p. 940902.
Geradts ZJ, Bijhold J, Kieft M, Kurosawa K, Kuroki K, Saitoh N. Methods for identification of images acquired with digital cameras. In: Enabling technologies for law enforcement, international society for optics and photonics; 2001. p. 505–12.
Gloe T. Forensic analysis of ordered data structures on the example of jpeg files. In: Information forensics and security (WIFS), 2012 IEEE international workshop on, IEEE; 2012. p. 139–44.
Hu W-C, Chen W-H, Huang D-Y, Yang C-Y. Effective image forgery detection of tampered foreground or background image based on image watermarking and alpha mattes. Multimedia Tools and Applications. 2015. p. 1–22.
Lee S, Song JE, Lee WY, Ko YW, Lee H. Integrity verification scheme of video contents in surveillance cameras for digital forensic investigations. IEICE TRANSACTIONS Inf Syst 2015;98(1):95–7.
Microsoft Developer Network, Avi riff file reference, http://msdn.microsoft.com/en-us/library/ms779636(VS.85).aspx.
Poisel R, Tjoa S. Forensics investigations of multimedia data: a review of the state-of-the-art. In: IT security incident management and IT forensics (IMF), 2011 sixth international conference on, IEEE; 2011. p. 48–61.
Qu Z, Luo W, Huang J. A framework for identifying shifted double jpeg compression artifacts with application to non-intrusive digital image forensics. Sci China Inf Sci 2014;57(2):1–18.
Rad RM, Wong K. Digital image forgery detection by edge analysis. In: Consumer electronics-Taiwan (ICCE-TW), 2015 IEEE international conference on, IEEE; 2015. p. 19–20.
Sencar HT, Memon N. Overview of state-of-the-art in digital image forensics. Algorithms, Archit Inf Syst Secur 2008;3:325–48.
Wang W, Farid H. Exposing digital forgeries in interlaced and deinterlaced video. Inf Forensics Secur IEEE Trans 2007;2(3):438–49.
Zheng J, Zhu T, Li Z, Xing W, Ren J. Exposing image forgery by detecting traces of feather operation. J Vis Lang Comput 2015;27:9–18.