Chapter 14
Integrity Verification Through File Container Analysis

Alessandro Piva and Massimo Iuliani

In the previous chapters, multimedia forensics techniques based on the analysis of the data stream, i.e., the audio-visual signal, aimed at detecting artifacts and inconsistencies in the (statistics of the) content were presented. Recent research highlighted that useful forensic traces are also left in the file structure, thus offering the opportunity to understand a file's life-cycle without looking at the content itself. This Chapter is then devoted to the description of the main forensic methods for the analysis of image and video file formats.

14.1 Introduction

Most forensic techniques look for traces within the content itself that are, however, mostly ineffective in some scenarios, for example, when dealing with strongly compressed or low resolution images and videos. Recently, it has been shown that also the image or video file container maintain some traces of the content history. The analysis of the file format and metadata allows to determine their compatibility, completeness, and consistency based on the expected media's history. This analysis proved to be strongly promising since the image and video compression standards leave some freedom in the file container's generation. This fact allows forensic practitioners to identify media's container signatures related to a specific brand, model, social media platform, etc. Furthermore, the cost for analyzing a file header is usually negligible. This is even more relevant when dealing with digital videos where the stream analy-

A. Piva (✉) · M. Iuliani
Department of Information Engineering, University of Florence, Via di S. Marta, 3, 50139 Florence, Italy
e-mail: alessandro.piva@unifi.it

H. T. Sencar et al. (eds.), *Multimedia Forensics*, Advances in Computer Vision and Pattern Recognition, https://doi.org/10.1007/978-981-16-7621-5_14

sis can quickly become unfeasible. Indeed, recent methods highlight that video can be analyzed in seconds, independently of their length and resolution.

It is worth also mentioning that most of the achieved results highlighted that most interesting containers' signatures usually have a low intra-variability, thus suggesting that this analysis can be promising even when only a few training data are available. On the other side, these techniques' main drawback is that they only allow forgery detection while localization is usually beyond its capabilities.

Furthermore, the design of practical media containers' parsers is not to be underestimated since it requires a deep comprehension of image and video formats, and the parser should be updated when new models are introduced in the market or unknown elements are added to the media header. Finally, we are currently not aware of any publicly available software that would allow users to consistently forge such information without advanced programming skills. This makes the creation of plausible forgeries undoubtedly a highly non-trivial task and thereby reemphasizes that file characteristics and metadata must not be dismissed as unreliable source of evidence for the purpose of file authentication per se. The Chapter is organised as follows: in Sects. 14.1.1 and 14.1.2 the main image and video file format specifications are summarized. Then, in Sect. 14.2 and 14.3 the main methods for the analysis of image and video file formats are described. Finally, in Sect. 14.4 the main findings of this research area are provided, along with some possible future works.

14.1.1 Main Image File Format Specifications

Multimedia technologies allow digital image generation based on several image formats. DSLR cameras and proficient acquisition devices can store images in uncompressed formats (e.g., DNG, CR2, BMP). When necessary, lossless compression schemes (e.g., PNG) can be applied to reduce the image size without impacting its quality. Furthermore, lossy compression schemes (e.g., JPEG, HEIF) are available to strongly reduce image size with minimal impact on visual quality.

Nowadays, most of the devices on the market and social media platforms generate and store JPEG images. For this reason, most of the forensic literature focuses on this image type. The JPEG standard itself JPEG (1992) defines the pixel data encoding and decoding, while the full-featured file format implementation is defined in the *JPEG Interchange Format* (JIF) ISO/IEC (1992). This format is quite complicated, and some simpler alternatives are generally preferred for handling and encapsulating JPEG-encoded images: the *JPEG File Interchange Format* (JFIF) Eric Hamilton (2004) and the *Exchangeable Image File Format* (Exif) JEITA (2002), both built on JIF.

Each of the formats defines the basic structure of JPEG files as a set of marker segments, either mandatory or optional. An identifier of the marker id indicates the beginning of each marker segment. Abbreviations for marker ids are denoted by 16 bit short values starting with 0xFF. Marker segments can encapsulate either data compression parameters (e.g., in DQT, DHT, or SOF) or application-specific

metadata, like thumbnails or EXIF metadata (e.g., in APP0(JFIF) or APP1(EXIF)). Additionally, some markers consist of the marker id only and indicate state transitions necessary to parse the file format. For example, SOI and EOI indicate the start and end of a JPEG file, requiring all other markers to be placed between these two mandatory markers.

The JIF standard also allows application markers (APPn), populated with entries in the form of key-value pairs. The values can vary from human-readable strings to complex structures like binary data. The APP0 segment defines pixel densities and pixel ratios, and an optional thumbnail of the actual image can be placed here. In APP1, the EXIF standard enables cameras to save a vast range of optional information (EXIF-Tags) related to the camera's photographic settings when and where the image was taken. The information are split into five main image file directories (IFDs): (i) Primary; (ii) Exif; (iii) Interoperability; (iv) Thumbnail; and (v) GPS. However, the content of each IFD is customizable by camera manufacturers, and the EXIF standard also allows for the creation of additional IFDs. Other metadata are customizable within the file header, such as XMP and IPTC that provide additional information like copyright or the image's editing history. Furthermore, image processing software can introduce a wide variety of marker segment sequences.

These differences in the file formats and the not strictly standardization of several sequences may expose forensic traces within the image file container Thomas Gloe (2012). Indeed, not all segments are mandatory and different combinations thereof can occur. Some segments can appear multiple times (e.g., quantization tables can be either encapsulated in one single or multiple separate DQT segments). Furthermore, the sequence of segments is generally not fixed (with some exceptions) and customizable. For example, JPEG thumbnails exist in different segments and employ their own complete sequence of marker segments. Eventually, arbitrary data can exist after EOI.

In the next sections, we report the main technologies and the results achieved based on the technical analysis of these formats.

14.1.2 Main Video File Format Specifications

When a camera acquires a digital video, image sequence processing and audio sequence processing are performed in parallel. After compression and synchronization, the streams are encapsulated in a multimedia container, simply called a video container from now on. There are several compression standards; however, H264/AVC or MPEG-4 Part 10 International Telecommunication Union (2016), and H265/HEVC or MPEG-H Part 2 International Telecommunication Union (2018) are the most relevant since most mobile devices implement them. Video and audio tracks, plus additional information (such as metadata and sync information), are then encapsulated in the video container based on specific format standards. In the following, we describe the two most adopted formats.

MP4-like videos

Most smartphones and compact cameras output videos in `mp4`, `mov`, or `3gp` format. This video packaging refers to the same standard, ISO/IEC 14496 Part 12 ISO/IEC (2008), that defines the main features of *MP4 File Format* ISO/IEC (2003) and *MOV File Format* Apple Computer, Inc (2001). The *ISO Base format* is characterized by a sequence of atoms or boxes. A unique 4-byte code identifies each node (atom) and each atom consists of a header and a data box, and possibly nested atoms. As an example, we report in Fig. 14.1 a typical structure of an MP4-like container. The file type box, `ftyp`, is a four-letter code that is used to identify the type of encoding, the compatibility, or the intended usage of a media file. According to the latest ISO standards, it is considered a semi-mandatory atom, i.e., the ISO expects it to be present and explicit as soon as possible in the file container. In the example given in Fig. 14.1 (a), the fields of the `ftyp` descriptor explain that the video file is MP4-like and it is compliant to the MP4 Base Media v1 [IS0 14496-12:2003] (here *isom*) and the 3GPP Media (.3GP) Release 4 (here *3gp4*) specifications.[1] The movie box, `moov`, is a nested atom containing the metadata needed to decode the data stream, which is embedded in the following `mdat` atom. It is important to note that `moov`

Fig. 14.1 Representation of a fragment of an MP4-like video container, of an original video acquired by a Samsung Galaxy S3

[1] The reader can refer to http://www.ftyps.com/ for further details.

may contain multiple `trak` box instances, as shown in Fig. 14.1. The `trak` atom is mandatory, and its numerosity depends on the number of streams included in the file; for example, if the video contains a visual-stream and an audio-stream, there will be two independent `trak` atoms. A more detailed description of these structures can be found in Carlos Quinto Huamán et al. (2020).

AVI videos

Audio video interleave (AVI) is a container format developed by Microsoft in 1992 for Windows software Microsoft. Avi riff file (1992). AVI files are based on the RIFF (resource interchange file format) document format consisting of a RIFF header followed by zero or more lists and chunks.

A chunk is defined as *ckID*, *ckSize*, and *ckData*, where *ckID* identifies the data contained in the chunk, *ckSize* is a 4-byte value giving the size of the data in *ckData*, and *ckData* is zero or more bytes of data.

A list consists of *LIST*, *listSize*, *listType*, and *listData*, where *LIST* is the literal code LIST, *listSize* is a 4-byte value giving the size of the list, *listType* is a 32-bit unsigned integer, and *listData* consists of chunks or lists, in any order.

An AVI file might also include an index chunk, which gives the location of the data chunks within the file. All AVI files must keep these three components in the proper sequence: the *hdrl* list defining the format of the data and is the first required LIST chunk; the *movi* list containing the data for the AVI sequence; the *idx1* list containing the index.

Depending on the specific camera or phone model, additional lists and JUNK chunks may exist between the *hdrl* and *movi* lists. These optional data segments are either used for padding or to store metadata. The *idx1* chunk indexes the data chunks and their location in the file.

14.2 Analysis of Image File Formats

The first studies in the image domain considered JPEG quantization tables and image resolution values Hany Farid (2006); Jesse D. Kornblum (2008); H. Farid (2008). Just with these few features, the authors demonstrated that it is possible to link a probe image to a set of devices or editing software. Then, in Kee Eric (2011), the authors increased the set of features by also considering JPEG coding data and Exif metadata. These studies provided better results in terms of integrity verification and device identification. The main drawback of such approaches is that a user can easily edit the metadata's information, limiting these methods' reliability. Following studies demonstrated that the file structure also contains a lot of traces of the content's history. These data are also more reliable, being much more challenging to extract and modify for a user than metadata. Furthermore, available editing software and metadata editors do not allow the modification of such low-level information Thomas Gloe (2012). However, like the previous ones, this method is based on a manual

analysis of the extracted features, to be compared with a set of collected signatures. In Mullan Patrick et al. (2019), the authors present the first automatic method for characterizing the source device, based on the analysis of features extracted from the JPEG header information. Finally, another branch of research demonstrated that the JPEG file format can be exploited to understand whether a probe image has been exchanged through a social network service. In the following, we describe the primary studies of this domain.

14.2.1 Analysis of JPEG Tables and Image Resolution

In Hany Farid (2006), the author shows, for the first time, that manufacturers typically configure the compression of devices differently according to their own needs and preferences. This difference, primarily manifested in the JPEG quantization table, can be used to identify the device that acquired a probe image.

To carry out this analysis, the authors collect a single image, at the highest quality, from each of 204 digital cameras. Then, they extract the JPEG quantization table, noticing that 62 cameras show a unique table.

The remaining cameras fall into equivalence classes of sizes between 2 and 28 (i.e., between 2 and 28 devices share the same quantization tables). Usually, cameras that share the same tables belong to the same manufacturer. Conversely, different makes and models usually share the same table. These results highlight that JPEG quantization tables can partially characterize the source device. Thus, they effectively narrow the source of an image, if not to a single camera make and model, at least to a small set of possible cameras (on average, this set size is 1.43).

This study was performed by also saving the images with five Photoshop versions (at that time, version CS2, CS, 7, 4, and 3) at each of the 13 available levels of compression available. As expected, quantization tables at each compression level were different from one another. Moreover, at each compression level, the tables were the same for all five versions of Photoshop. More importantly, no one of these tables can be found in the images belonging to the 204 digital cameras. These findings allow arguing the possibility to link an image to specific editing software, or at least to a set of possible editing tools.

Similar results have been presented in Jesse D. Kornblum (2008). The authors examined images from devices (a Motorola KRZR K1m, a Canon PowerShot 540, a FujiFilm Finepix A200, a Konica Minolta Dimage Xg, and a Nikon Coolpix 7900) and images edited by several software programs such as libjpeg, Microsoft Paint, the Gimp, Adobe Photoshop, and Irfanview. The analysis carried out on this dataset shows that, although some cameras always use the same quantization tables, most of them use a different set of quantization tables in each image. Moreover, these tables usually differ according to the source or editing software.

In H. Farid (2008), the author expands the earlier work by considering a much larger dataset of images and adding the image resolution to the quantization table as discriminating features for source identification.

The author analyzes over 330 thousand Flickr images from 859 different camera models from 48 manufacturers. Quantization tables and resolutions allowed to obtain 10153 different image classes.

The resolution and quantization table were then combined to narrow the search criteria to identify the source camera. The analysis revealed that 2704 out of 10153 entries (26.6%) have a unique paired resolution and quantization table. In these cases, the features can correctly discriminate the source device. Moreover, 37.2% of the classes have at most two matches, and 44.1% have at most three matches. These results show that the combination of JPEG quantization table and image resolution allows matching the source of an image to a single camera make and model or at least to a small set of possible devices.

14.2.2 Analysis of Exif Metadata Parameters

In Kee Eric (2011), the authors expand the previous analysis by creating a larger image signature for identifying the source camera with greater accuracy than only using quantization tables and resolution. This approach considers a set of features of the JPEG format, namely properties of the run-length encoding employed by the JPEG standard and some Exif header format characteristics. In particular, the signature is composed of three kinds of features: image parameters, thumbnail parameters, and Exif metadata parameters. The image parameters consist of the image size, quantization tables, and the Huffman codes; the Y, Cb, and Cr 8×8 quantization tables are represented as a vector of 192 values; the Huffman codes are specified as six vectors (one for the dc DCT coefficients and one for the ac DCT coefficients of each of the three channels Y, Cb, and Cr) storing the number of codes of length from 1 to 15. This part of the signature is then composed of 284 values: 2 image dimensions, 192 quantization values, and 90 Huffman codes.

The same components are extracted from the image thumbnail, usually embedded in the JPEG header. Thus, the thumbnail parameters are other 284 values: 2 thumbnail dimensions, 192 quantization values, and 90 Huffman codes.

A compact representation of EXIF Metadata is finally obtained by counting the number of entries in each of the five image file directories (IFDs) that compose the EXIF metadata, as well as the total number of any additional IFDs, and the total number of entries in each of these. It worth noticing that some manufacturers adopt metadata representation that does not conform to the EXIF standard, yielding errors when parsing the metadata. The authors also consider these errors as discriminating features, so the total number of parser errors, as specified by the EXIF standard, is also stored. Thus, they consider 8 values from the metadata: 5 entry values from the standard IFDs, 1 for the number of additional IFDs, 1 for the number of entries in these additional IFDs, 1 for the number of parser errors.

The image signature is then composed of 284 header features extracted from the full resolution image, 284 header features from the thumbnail image, and another 8 from the EXIF metadata, for a total of 576 features. These signatures are extracted

Table 14.1 The percentage of camera configurations with an equivalence class size from 1 to 5. Each row corresponds to different subsets of the complete signature

	Equivalence class size				
	1	2	3	4	5
Image	12.9%	7.9%	6.2%	6.6%	3.4%
Thumbnail	1.1%	1.1%	1.0%	1.1%	0.7%
EXIF	8.8%	5.4%	4.2%	3.2%	2.6%
Image+thumbnail	24.9%	15.3%	11.3%	7.9%	3.7%
All	69.1%	12.8%	5.7%	4.0%	2.9%

from the image files and analyzed to verify if they can assign the image to a given camera make and model.

Tests are carried out on over 1.3 million Flickr images, acquired by 773 camera and cell phone models, belonging to 33 different manufacturers.

These images span 9, 163 different distinct pairings of the camera make, model, and signature, referred to as a camera configuration. It is worth noting that each camera model can produce different signatures based on the acquisition resolutions and quality settings. Indeed, in the collected dataset, we find an average of 12 different signatures for each camera make and model.

The camera signature analysis on the above collection shows that the image, thumbnail, and EXIF parameters are not distinctive individually. However, when combined, they provide a highly discriminative signature. This result indicates that the three classes of parameters are not highly correlated, and hence their combination improves overall distinctiveness, as show in Table 14.1.

Eventually, the signature was tested on images edited by Adobe Photoshop (versions 3, 4, 7, CS, CS2, CS3, CS4, CS5 at all qualities), by exploiting, in this case, image and thumbnail quantization tables and Huffman codes only. No overlaps were found between any Photoshop version/quality and camera manufacturer. The Photoshop signatures, each residing in an equivalence class of size 1, are unique. This means that any photo-editing with Photoshop can be easily and unambiguously detected.

14.2.3 Analysis of the JPEG File Format

In Thomas Gloe (2012), the author makes a step forward with respect to the previous studies. They analyze the JPEG file format structure to identify new characteristics that are more discriminating than JPEG metadata only. In this study, a subset of images extracted from the Dresden Dataset (Thomas Gloe and Rainer Bähme 2014) was used. In particular, 4, 666 images belonging to the JPEG scenes (a part of the dataset built to specifically study model-specific JPEG compression algorithms) and 16, 956 images of natural scenes were considered. Images were captured with one

device of each available camera model while iterating over all combinations of compression, image size, and flash settings) Overall, they considered images acquired by 73 devices from 26 camera models of 14 different brands. A part of these authentic images was re-saved with each of the eight investigated image processing software (ExifTool, Gimp, IrfanView, Jhead, cjpeg, Paint.NET, PaintShop, and Photoshop), with all available combinations of JPEG compression settings. Finally, they obtained and analyzed more than 32, 000 images.

The study focuses firstly on analyzing the sequence and occurrence of marker segments in the JPEG data structures. The main findings of the author are that:

1. some optional segments can occur with different combinations;
2. segments can appear multiple times;
3. the sequence of segments is generally not fixed, with the exception of some required combinations;
4. JPEG thumbnails exist in different segments and employ their complete sequence of marker segments;
5. arbitrary data can exist after the marker end of image (EOI) of the main image.

The markers' analysis allows to distinguish between pristine and edited images correctly. Furthermore, none of the investigated software allows to recreate the sequence of markers consistently.

The author also considered the sequence of EXIF data structures that store camera properties and image acquisition settings. Interestingly, they found differences between digital cameras, image processing software, and metadata editors. In particular, the following four characteristics appear to be relevant:

1. the byte order is different between camera models (and the default setting employed by ExifTool);
2. sequences of image file directories (IFDs) and corresponding entries (including tag, data type and often also offsets) appear constant in images acquired with the same model and differ between different sources;
3. some manufacturers use different data types for the same entry;
4. raw values of rational types differ between different models while resulting in the same interpreted values (e.g., 200/10 or 20/1).

These findings allow to conclude that the analysis of EXIF data structures can be useful to discriminate between authentic and edited images.

14.2.4 Automatic Analysis of JPEG Header Information

All previous works analyzed images acquired by digital cameras, whereas today, most of the visual content is generated by smartphones. These two classes of devices are somewhat different. While traditional digital cameras typically have a fixed firmware, modern smartphones keep updating their acquisition software and operating system,

making it harder to achieve a unique characterization of device models based on the image file format.

In Mullan Patrick et al. (2019), the authors performed a large-scale study on Apple smartphones to characterize the source device based on JPEG header information. The results confirm that identifying the hardware is much harder for smartphones than for traditional cameras, while we can identify the operating system version and selected apps.

The authors exploit Flickr images following the rules adopted in Kee Eric (2011). Starting from a crawling of one million images, they filtered the downloaded media to remove edited images, obtaining at the end a dataset of 432, 305 images. All images belong to Apple devices, including all models comprised between iPhone 4 and iPhone X, between iPad Air and iPad Air 2, with operating system versions from iOS 5 to iOS 12.

They consider two families of features: the number of entries in the image file directories (IFDs) and the quantization tables. The selected directories include the standard IFDs "ExifIFD", "IFD0", "IFD1", "GPS", and the special directories "ICC_Profile" and "MakerNotes". The Y and Cb quantization tables are concatenated, and their hash is computed, obtaining one alphanumeric number, called QC-Table. Differently from Kee Eric (2011), the Huffman tables were not considered since, in the collected database, they are identical in the vast majority of cases. For similar reasons, they also discarded the sequence of JPEG syntax elements studied by Thomas Gloe (2012).

First, the authors verified the modification of the header information for Apple smartphones over time. They showed that image metadata change more often than compression matrices, and that different hardware platforms with same operating system exhibit very similar metadata information. These findings highlight that it is more complicated to analyse smartphones than digital cameras.

After that, the authors design an algorithm for determining the smartphone hardware model from the header information in an automatic way. This is the first algorithm designed to perform an automatic source identification on images through file format information. The classification is performed with a random forest, that uses numbers of entries per directory and quantization tables, as described before, as features.

Firstly, a random forest classifier was trained to identify seven different iPhone models marketed in 2017. Secondly, the same algorithm was trained to verify its ability to identify the operating system from header information, considering all versions from iOS 4 to iOS 11. In all the two cases, experiments were performed by considering as input features: (i) the Exif directories only, (ii) the quantization tables only, (iii) metadata and quantization features together. We report the achieved results in Table 14.2. It worth noticing that using only the quantization tables gives the lowest results, whereas the highest accuracy is reached with all the considered features. Moreover, it is easier to discriminate among operating systems than among hardware models.

Table 14.2 Overall accuracy of models and iOS version classification when using EXIF directories (EXIF-D), quantization matrices (QT), or both

	EXIF-D	QT	Both
iPhone Models classification	61.9%	35.4%	65.6%
iOS version Classification	80.4%	33.7%	81.7%

14.2.5 Methods for the Identification of Social Networks

Several works investigated how JPEG file format modifications can be exploited to understand whether a probe image has been exchanged through a social network service. Indeed, a social network usually recompresses, resizes, and alters image metadata.

Castiglione et al. Castiglione et al. (2011) made the first study on image resizing, compression, renaming, and metadata modifications introduced by Facebook, Badoo, and Google+.

Giudice et al. Oliver Giudice et al. (2017) built a dataset of images from different devices, including digital cameras and smartphones with Android and iOS operating systems. Then, they exchanged all the images through ten different social networks.

The considered features are a 44-dimensional vector composed of pixel width and height, an array containing the Exif metadata, the number of JPEG markers, the first 32 coefficients of the luminance quantization table, and the first 8 coefficients of the chrominance quantization table. These features are fed to an automatic detector based on two K-NN classifiers and a decision tree fitted on the built dataset. The method can predict the social network the image belongs to and the client application with an accuracy of 96% and 97.69% respectively.

Summarizing, the authors noticed that modifications left by the process in the JPEG file depend on both the platform (server-side) and the application used for the upload (client-side).

In Phan et al. (2019), the previous method was extended by merging JPEG metadata information (including quantization tables, Huffman coding tables, image size) with content-related features, namely the histograms of DCT coefficients. A CNN is trained on these features to detect an image's multiple exchanges through Facebook, Flickr, and Twitter.

The adopted dataset consists of two parts: R-SMUD (RAISE Social Multiple Up-Download), generated by the RAISE dataset Duc-Tien Dang-Nguyen et al. (2015), containing images shared over the three Social Networks; V-SMUD (VISION Social Multiple Up-Download), obtained from a part of the VISION dataset Dasara Shullani et al. (2017), shared three times via the three social networks, following different testing configurations.

The experiments demonstrated that JPEG metadata's information improves the performance with respect to using content-based information only. When considering

single-shared contents, the accuracy improves from 85.63% to 99.87% on R-SMUD and is in both cases 100.00% on V-SMUD. When considering images shared both once and twice, the accuracy improves from 43.24% to 65.91% on R-SMUD and from 58.82% to 77.12% on V-SMUD. When dealing with three times shared images also, they achieve an accuracy of 36.18% on R-SMUD and 49.72% on V-SMUD.

14.3 Analysis of Video File Formats

Gloe et al. proposed the first more in-depth analysis of video file containers Gloe Thomas et al. (2014). The authors observe that videos from different cameras and phone models expose different container formats and compression. This allows a forensic practitioner to extract this information manually and, based on a comparison with a reference, expose forensic findings. In Jieun Song et al. (2016) the authors studied 296 video files in AVI format acquired with 43 video event data recorders (VEDRs), and their manipulated version with 5 different video editing software tools. All videos were classified according to a sequence of field data structure types. This study showed that the field data structure represents a valid feature set to determine whether video editing software was used to process a probe video. However, these methods require the manual analysis of the video container information, thus needing high effort and high programming skills. Some attempts were proposed to identify the containers' signature of the most relevant brands and social media platforms Carlos Quinto Huamán et al. (2020). Another path tried to automatize the forensic analysis of video file structures by verifying their multimedia stream descriptors with simple binary classifiers through a supervised approach David Güera et al. (2019). However, in Iuliani Massimo (2019), the authors propose a way to formalize the MP4-like (.mp4, .mov, .3gp) video file structure. A probe video is converted in an XML tree-based structure; then, the comparison between the parsed data and a reference dataset addresses both integrity verification and brand classification. A similar approach has been also proposed in Erik Gelbing (2021), but the experiments have been applied only to the scenario of brand classification.

Both Iuliani Massimo (2019) and David Güera et al. (2019) allow the integrity assessment of a digital video with negligible computational costs. A further improvement was finally proposed in Yang Pengpeng et al. (2020), where decision trees were employed to reduce further the computational effort required to a fixed amount of time, independently of the reference dataset's size. These latest works also introduced an open-world scenario where the tested device was not available in the training set. The issue is also addressed in Raquel Ramos López et al. (2020), where information extracted from the video container was used to cluster sets of videos by data source, consisting in brand, model, or social media. In the next sections, we briefly summarize the main works: (i) the first relevant paper on video container Gloe Thomas et al. (2014), (ii) the first video container formalization and usage Iuliani Massimo (2019), (iii) the most recent work for efficient video file format analysis Yang Pengpeng et al. (2020).

14.3.1 Analysis of the Video File Structure

In Gloe Thomas et al. (2014), the authors identify the manufacturer and model-specific video file format characteristics and show how processing software further modifies those features. In particular, they consider 19 digital cameras and 14 mobile phones belonging to different models, and they consider from 3 to 14 videos per device, with all available video quality settings (e.g., frame size and frame rate). In Table 14.3, we report the list of these devices. Most of these digital cameras

Table 14.3 The list of devices analyzed in the study

Make	Model	Container
Digital camera models		
Agfa	DC-504, DC-733s, DC-830i, Sensor530s	AVI
Agfa	Sensor505-X	AVI
Canon	S45,S70, A640, Ixus IIs	AVI
Canon	EOS-7D	MOV
Casio	EX-M2	AVI
Kodak	M1063	MOV
Minolta	DiMAGE Z1	MOV
Nikon	CoolPix S3300	AVI
Pentax	Optio A40	AVI
Pentax	Optio W60	AVI
Praktica	DC2070	MOV
Ricoh	GX100	AVI
Samsung	NV15	AVI
Mobile phone models		
Apple	IPhone 4	MOV
Benq Siemens	S88	3GP
BlackBerry	8310	3GP
Google	Nexus 7	3GP
LG	KU990	3GP, AVI
Motorola	MileStone	3GP
Nokia	6710	3GP, MP4
Nokia	E61i	3GP, MP4
Nokia	E65	3GP, MP4
Nokia	X3-00	3GP
Palm	Pre	MP4
Samsung	GT-5500i	MP4
Samsung	SGH-D600	MP4
Sony Ericsson	K800i	3GP

store videos in AVI format, and some use Apple Quicktime MOV containers and compress video data using motion JPEG (MJPEG), and only three use more efficient codecs (DivX, Xvid, or H.264). All the mobile phones store video data in MOV-based container formats (MOV, 3GP, MP4), except for the LG KU990 camera phone, which supports AVI. On the contrary, the mobile phones adopt H.263, H.264, MPEG-4 or DivX, and not MJPEG compression.

A subset of the camera models has also been used to generate cut short videos, 10 seconds long, through video editing tools (FFmpeg, Avidemux, FreeVideoDub, VirtualDub, Yamb, and Adobe Premiere. Except for Adobe Premiere, all of them have allowed lossless video editing, i.e., the edited files have been saved without re-compressing the original stream. After that, they have analyzed the extracted data. This analysis reveals considerable differences between device models and software vendors, and some general findings are summarized here.

The authors carry out the first analysis by grouping the devices according to the file structure, i.e., AVI or MP4. Original AVI videos do not share the same components. Both the content and the format of specific chunks may vary, depending on the particular Camera model. Their edited versions, with all software tools, even when lossless processing is applied, leave distinct traces in the file structure, which do not match any of the camera's features in the dataset. It is then possible to use these differences to spot if a video file was edited with these tools by comparing a questioned file's file structure with a device's reference container.

Original MP4 videos, due to the by far more complex file format, exhibit an even more considerable degree of source-dependent internal variations than AVI files. However, none of the cut videos produced through the editing tools is compatible with any source device considered in the study. Dealing with original MJPEG-compressed video frames, different camera models adopt different JPEG marker segment sequences when building MJPEG-compressed video frames. Moreover, content-adaptive quantization tables are generally used in a video sequence, such that 2914 unique JPEG luminance quantization tables in this small dataset have been found. On these contents, lossless video editing leaves compression settings unaltered, but they introduce very distinctive artifacts in container files' structure. These findings stated that videos from different digital cameras and mobile phones appear to employ different container formats and compression. It worth noticing that this study is not straightforward since the authors had to write customized file parsers to extract the file format information from the videos. On the other side, this difficulty is also advantageous since we are currently unaware of any publicly available software that consistently allows users to forge such information without advanced programming skills. This fact makes the creation of realistic forgeries undoubtedly a highly non-trivial undertaking and thereby reemphasizes that file characteristics and metadata must not be dismissed as unreliable sources of evidence to address file authentication.

14.3.2 Automated Analysis of mp4-like Videos

The previous method is subjected to two main limitations: Firstly, being manual is time demanding. Secondly, it is prone to error since the forensic practitioner have to subjectively assess the finding's value. In Iuliani Massimo (2019), an automated method for unsupervised analysis of video file containers to assess video integrity and classify the source device brand. The authors developed an MP4 Parser library Apache (2021) that automatically extracted a video container and store it in an XML file. The video container is represented as a labeled tree where internal nodes are labeled by atoms names (e.g.,*moov-2*), and leaves are labeled by field-value attributes (e.g.,**@stuff:MovieBox[]**). To take into account the order of the atoms, each XML-node is identified by a 4-byte code of the corresponding atom along with an index that represents the relative position with respect to the other siblings at a certain level.

Technical Description

A video file container, extracted from a video X, can be represented as an ordered collection of atoms $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$, possibly nested.

We can describe each atom in terms of a set of field-value attributes, as $a_i = \big(\omega_1(\boldsymbol{a}_i), \ldots, \omega_{m_i}(\boldsymbol{a}_i)\big)$.

By combining the two previous descriptions, the video container can be characterized by the set of field-value attributes $X = \big(\omega_1, \ldots, \omega_m\big)$, each with its associated path $p_X(\omega)$, that is the ordered list of atoms to be crossed to reach ω in X starting from the root. As an example, consider the video X, whose fragments are reported in Fig. 14.1. The path to reach the field-value $\omega =$ *@timescale: 1000* in X is the sequence of atoms (*ftyp-1, moov-2, mvhd-1*). In this sense, we state that $p_X(\omega) = p_{X'}(\omega')$ if the same ordered list of atoms is crossed to reach the field-values in the two trees, respectively, which is not the case in the previous example.

In summary, the video container structure is completely described by a list of m field-value attributes $X = \big(\omega_1, \ldots, \omega_m\big)$, and their corresponding paths $p_X(\omega_1), \ldots, p_X(\omega_m)$.

From now on, we will denote with $\mathcal{X} = \{X_1, \ldots, X_N\}$ the world set of digital videos, and with $\mathcal{C} = \{C_1, \ldots, C_s\}$ the set of disjoint possible origins, e.g., device *Huawei* P9, iPhone 6s, and so on.

When a video is processed in any way (with respect to its native form), its integrity is compromised SWGDE-SWGIT (2017).

More generally, given a query video X and a native reference video X' coming from the same supposed device model, their container structure dissimilarities can be exploited to expose integrity violation evidence, as follows.

Given two containers $X = (\omega_1, \ldots, \omega_m)$, $X' = (\omega'_1, \ldots, \omega'_m)$ with the same cardinality[2] m, we define a similarity core function between two field-values as

[2] If the two containers have a different cardinality, we pad the smaller one with empty field-values to obtain the same value m.

$$S(\omega_i, \omega_j^{'}) = \begin{cases} 1 & \text{if } \omega_i = \omega_j^{'} \text{ and } p_X(\omega_i) = p_{X'}(\omega_j^{'}) \\ 0 & \text{otherwise} \end{cases} \tag{14.1}$$

We can easily extend the measure to the comparison between a single field-value $\omega_i \in X$ and the whole $X^{'}$ as

$$\mathbf{1}_{X'}(\omega_i) = \begin{cases} 1 & \text{if } \exists\, \omega_j^{'} \in X^{'} : S(\omega_i, \omega_j^{'}) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{14.2}$$

Then, the dissimilarity between X and $X^{'}$ can be computed as the mismatching percentage of all field-values, i.e.,

$$mm(X, X^{'}) = 1 - \frac{\sum_{i=1}^{m} \mathbf{1}_{X'}(\omega_i)}{m} \tag{14.3}$$

and, to preserve symmetry, the degree of dissimilarity between X and $X^{'}$ can be computed as

$$D(X, X^{'}) = \frac{mm(X, X^{'}) + mm(X^{'}, X)}{2}. \tag{14.4}$$

Based on its definition, $D(X, X^{'}) \in [0, 1]$, $D(X, X^{'}) = 1$ when $X = X^{'}$, and $D(X, X^{'}) = 0$, when they have no elements in common.

Experiments

The method is tested on 31 portable devices from VISION dataset Dasara Shullani et al. (2017) that leads to an available collection of 578 videos in the native format plus their corresponding social versions (YouTube and WhatsApp are considered).

The metric in Eq. (14.4) is applied to determine whether a video's integrity is preserved or violated. The authors consider four different scenarios of integrity violation[3]:

- exchange through *WhatsApp*;
- exchange through *YouTube*;
- cut without re-encoding through *FFmpeg*;
- date editing through *ExifTool*.

For each of the four cases, we consider the set of videos $X_1, \ldots, X_{N_{C_i}}$, available for each device C_i, and we compute the intra-class dissimilarities between two native videos X_i, X_j belonging to the same model $D_{i,j} = D(X_i, X_j), \forall i \neq j$ based on Eq. (14.4). For simplicity, we denote with D_{oo} this set of dissimilarities. Then, we consider the corresponding inter-class dissimilarities $D^t_{i,j} = D(X_i, X^t_j), \forall i \neq j$,

[3] See the paper for the implementation details.

Table 14.4 AUC values computed for all devices for each of the four attacks, and when the native video is compared to videos from other devices. In the case of ExifTool and Other Devices, the achieved min and max AUC values are represented. For all other cases, an AUC equal to 1 is always obtained

Case	WhatsApp	YouTube	FFmpeg	ExifTool
AUC	1.00	1.00	1.00	[0.64 1.00]

between a native video X_i and its corrupted version X_j^t obtained with the tool-t (WhatsApp, YouTube, *ffmpeg*, or *Exiftool*). We denote with D_{oa}^t this set of dissimilarities. By applying this procedure to all the considered devices, we collected 2890 samples for both D_{oo} and any of the four D_{oa}^t.

In most of the performed tests the maximum value achieved for D_{oo} is lower than the minimum value of D_{oa}^t, thus indicating that the two classes can be separated. In Table 14.4 we report the AUCs for each of the considered cases. Noticeably, the integrity violation through *ffmpeg*, YouTube, and WhatsApp, the AUC is 1 for all devices. These values clearly show that the exchange through WhatsApp and YouTube strongly compromises the container structure, and thus it is easily possible to detect the corresponding integrity violation. Interestingly, cutting the video using *ffmpeg*, without any re-encoding, also results in a substantial container modification that we can detect. On the contrary, the date change with *Exiftool*induced on some containers a modification that is comparable with the observed intra-variability of native videos; in these cases, the method can not detect it. Fourteen devices still achieve unitary AUC, eight more devices yield an AUC greater than 0.8, and the remaining nine devices yield AUC below 0.8, with the lowest value of 0.64 obtained for D02 and D20 (*Apple* iPhone 4s and iPad mini).

To summarize, for the video integrity verification tests, the method obtains perfect discrimination for videos altered by social networks or *ffmpeg*, while for *Exiftool*it achieves an AUC greater than 0.82 on 70% of the considered devices. It worth also noticing that analyzing a video query requires less than a second.

More detailed results are reported in the corresponding paper. The authors also show that this container description can be immersed in a likelihood ratio framework to determine which atoms and field-values are highly discriminative for specific classes. In this way, they also show how to classify the brand of the originating device Iuliani Massimo (2019).

14.3.3 Efficient Video Analysis

The method described in Iuliani Massimo (2019), although effective, has a linear computational cost since it requires checking the dissimilarity of the probe video with all available reference containers. As a consequence, increasing the reference dataset size leads to a higher computational effort.

In this section, we summarize the method proposed in Yang Pengpeng et al. (2020) that provides an efficient way to analyze video file containers independently on the reference dataset's size.

The method, called *EVA* from now on, is based on Decision Trees Ross Quinlan (1986), a non-parametric learning method used for classification problems in many signal processing fields.

Their key feature is breaking down a complex decision-making process into a collection of more straightforward decisions.

The process is extremely efficient since a decision can be taken by checking a small number of features.

Furthermore, *EVA* allows both to characterize the identified manipulations and to provide an explanation for the outcome.

The method is also enriched with a likelihood ratio framework designed to automatically clean up the container elements that only contribute to source intra-variability (for the sake of brevity, we do not report these details here).

In short, *EVA* can identify the manipulating software (e.g., *Adobe Premiere*, *ffmpeg*, …) and provide additional information related to the original content history such as the source device operating system.

Technical Description

Similarly to the previous case, we consider a video container X as a labeled tree where internal nodes and leaves correspond to, respectively, atoms and field-value attributes. It can be characterised by the set of symbols $\{s_1, \ldots s_m\}$, where s_i can be: (i) the path from the root to any field (value excluded), also called *field-symbols*; (ii) the path from the root to any field-value (value included), also called *value-symbols*. An example of this representation can be[4]:

s_1 = [`ftyp/@majorBrand`]
s_2 = [`ftyp/@majorBrand/isom`]
…
s_i= [`moov/mvhd/@duration`]
s_{i+1}= [`moov/mvhd/@duration/73432`]
…

Overall, we denote with Ω the set of all unique symbols $s_1, \ldots, s_M$ available in the world set of digital video containers $\mathcal{X} = \{X_1, \ldots, X_N\}$. Similarly, $\mathcal{C} = \{C_1, \ldots, C_s\}$ denotes a set of possible origins (e.g., *Huawei* P9, *Apple* iPhone 6s). Let a container X, the different structure of its symbols $\{s_1, \ldots, s_m\}$ can be exploited to assign the video to a specific class C_u.

For this purpose, binary decision trees Rasoul Safavian and Landgrebe (1991) are employed to build a set of hierarchical decisions. In each internal tree node the input data is tested against a specific condition; the test outcome is used to select a child as the next step in the decision process. Leaf nodes represent decisions taken by the

[4] Note that `@` is used to identify atom parameters, and `root` is used for visualization purposes. Still, it is not part of the container data.

algorithm. More specifically, *EVA* adopts the growing-pruning-based Classification And Regression Trees (CART) Leo Breiman (2017).

Given the size of unique symbols $|\Omega| = M$, a video container X is converted into a vector of integers $X \mapsto (x_1 \dots x_M)$ where x_i is the number of times that s_i occurs into X. This approach is inspired by the bag-of-words representation Hinrich Schütze et al. (2008) which is used to reduce variable-length documents to a fixed-length vectorial representation.

For the sake of example, we consider two classes:

C_u: iOS devices, native videos;
C_v: iOS devices, dates modified through *Exiftool* (see Sect. 14.3.3 for details).

With these settings, the decision tree highlights that the usage of *Exiftool* can be simply decided by looking, for instance, at the presence of `moov/udta/XMP_/@stuff`.

Experiments

Tests were performed on 34 smartphones of 10 different brands. Over a thousand tampered videos were tested, considering both automated (*ffmpeg*, *Exiftool*) and manual editing operations (*Kdenlive*, *Avidemux* and *Adobe Premiere*). The manipulations mainly include cut with and without re-encoding, speed up, and slow motion.

Achieved videos (both native and manipulated) were also exchanged through YouTube, Facebook, Weibo, and TikTok.

All tests were performed by adopting an exhaustive leave-one-out cross-validation strategy: the dataset is partitioned in 34 subsets, each one of them containing pristine, manipulated, and social-exchanged videos belonging to a specific device. In this way, test accuracy collected after each iteration is computed on videos belonging to an unseen device. When compared to state of the art, *EVA* exposes competitive effectiveness at the lowest computational effort (see 14.5).

In comparison with David Güera et al. (2019), it achieves higher accuracy. We can reasonably attribute this performance to their use of a smaller feature space; indeed, only a subset of the available pieces of information are extracted without considering their position within the video container. On the contrary, *EVA* features also include the path from the root to the value, thus providing a higher discriminating power. Indeed, this approach distinguishes between two videos where the same information

Table 14.5 Comparison of our method with the state of the art. Values of accuracy and time are averaged over the 34 folds

	Balanced accuracy	Training time	Test time
Guera et al. David Güera et al. (2019)	0.67	347 s	< 1 s
Iuliani et al. Iuliani Massimo (2019)	0.85	N/A	8 s
EVA	0.98	31 s	< 1 s

is stored in different atoms. When compared with Iuliani Massimo (2019), *EVA* is capable of obtaining better classification performance with a lower computational cost. In Iuliani Massimo (2019), $O(N)$ comparisons are required since all the N reference-set examples must be compared with a tested video; on the contrary, the cost for a decision tree analysis is $O(1)$ since the output is reached in a constant number of steps. Furthermore, *EVA* allows a simple explanation for the outcome. For the sake of example, we report in Fig. 14.2a a sample tree from the integrity verification experiments: the decision is taken by up to four checks, just based on the presence of the symbols `ftyp/@minorVersion = 0`, `uuid/@userType`, `moov/udta/XMP_` and `moov/udta/auth`. Then, a single decision tree can handle both easy- and hard-to-classify cases at the same time.

Manipulation Characterization
EVA is also capable of identifying the manipulating software and the operating system of the originating device. More specifically, three main questions were posed:

A **Software identification:** Is the proposed method capable of identifying the software used to manipulate a video? If yes, is it possible to identify the operating system of the original video?
B **Integrity Verification on Social Media:** Given a video from a social media platform (YouTube, Facebook, TikTok or WeiBo), can we determine whether the original video was pristine or tampered?
C **Blind scenario:** Given a video that may or may not has been exchanged through a social media platform, is it possible to retrieve some information on the video origin?

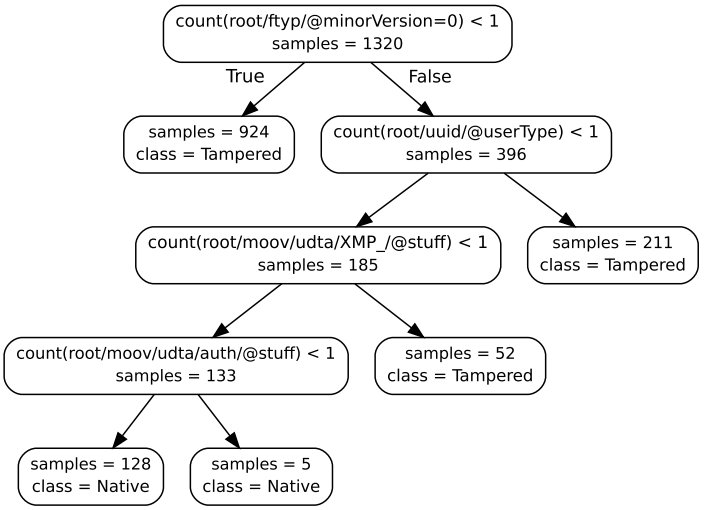
In the **Software Identification**, the experiment comprises the following classes: "native" (136 videos), "*Avidemux*" (136 videos), "*Exiftool*" (136 videos), "*ffmpeg*" (680 videos), "*Kdenlive*" (136 videos), and "*Premiere*" (136 videos). As shown in Table 14.6, *EVA* obtains a balanced global accuracy of 97.6% with slightly lower accuracy in identifying *ffmpeg* with respect to the other tools. These wrong classifications are reasonable because *ffmpeg* library is used by other software and, internally, by Android devices.

The algorithm maintains a high discriminative power even when trained to identify the originating operating system (Android vs. iOS).

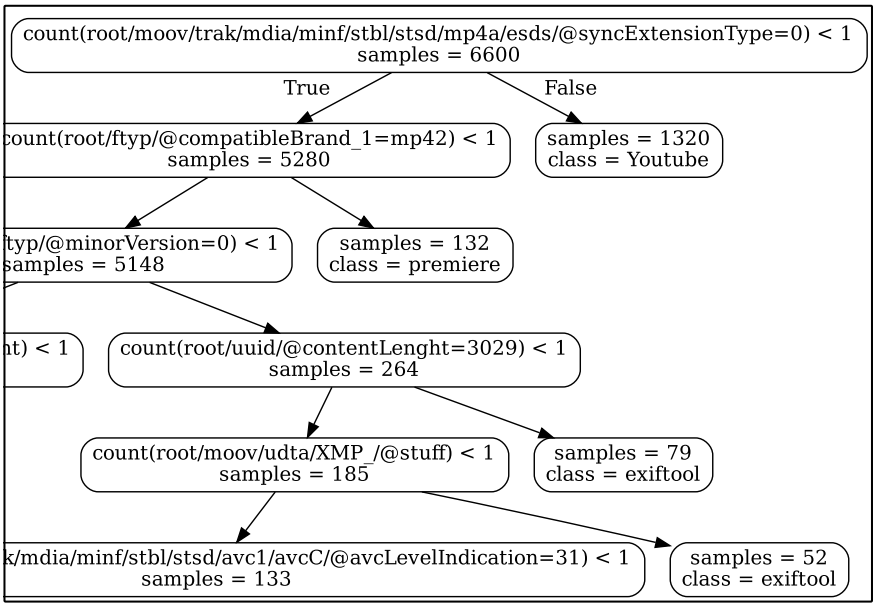
In a few cases, *EVA* wrongly classifies only the source's operating system. This mistake happens explicitly in the case of *Kdenlive* and *Adobe Premiere*. At the same time, these programs are always correctly identified. This result indicates that the container's structure of videos saved by *Kdenlive* and *Adobe Premiere* is probably reconstructed in a software-specific way. More detailed performance is reported in the related paper.

With regard to **Integrity Verification on Social Media**, *EVA* was also tested on YouTube, Facebook, TikTok, and Weibo videos to determine whether they were pristine or manipulated before the upload (Table 14.7).

Results highlight poor performance due to the social media transcoding process that flattens the containers almost independently on the video origin. For example,



(a) Integrity verification classifier.



(b) Detail of a blind scenario classifier.

Fig. 14.2 Pictorial representation of some of the generated decision trees

Table 14.6 Confusion matrix under the software identification scenario for native contents (Na), *Avidemux* (Av), *Exiftool* (Ex), *ffmpeg* (Ff), *Kdenlive* (Kd), *Premiere* (Pr)

	Na	Av	Ex	Ff	Kd	Pr
Na	0.97	–	0.03	–	–	–
Av	–	1.00	–	–	–	–
Ex	0.01	–	0.99	–	–	–
Ff	–	0.01	–	0.90	0.09	–
Kd	-	–	–	–	1.00	–
Pr	–	–	–	–	–	1.00

Table 14.7 Performance achieved for integrity verification on social media contents. We report for each social network the obtained accuracy, true positive rate (TPR), and true negative rate (TNR). All these performance measures are balanced

	Accuracy	TNR	TPR
Facebook	0.76	0.40	0.86
TikTok	0.80	0.51	0.75
Weibo	0.79	0.45	0.82
YouTube	0.60	0.36	0.74

after YouTube transcoding, videos produced by *Avidemux* and by *Exiftool* are shown to have exactly the same container representation. We do not know how the considered platforms process the videos due to the lack of public documentation, but we can assume that uploaded videos undergo custom/multiple processing. Indeed, social media videos need to be viewable on a great range of platforms and thus need to be transcoded to multiple video codecs and adapted for various resolutions and bitrates. Thus, it seems plausible that those operations could discard most of the original container structure.

Eventually, in the *Blind Scenario*, the authors show that we can remove the prior information (whether the video belongs to social media or not) without degrading detection performances.

In summary, *EVA* can be considered an efficient forensic method for checking video integrity. If manipulation is detected, *EVA* can also identify the editing software and, in most cases, the video source device's operating system.

14.4 Concluding Remarks

In this chapter, we described how features extracted from image and video file containers can be exploited for integrity verification. In some cases, these features also provide insights into the probe's previous history. In several cases, they allow the

characterization of the source device's brand or model. Similarly, they allow deriving that the content was exchanged through a particular social network.

This kind of information shows several advantages with respect to content-based traces. Firstly, it is usually much faster to be extracted and analyzed than content-based ones. It also requires an almost fixed computational effort, independently on the media size and resolution (this characteristic is particularly relevant for videos). Secondly, it is also much more challenging to perform anti-forensics operations on these features since the structure of an image, or video file is overly complicated. Thus, it is not straightforward to mimic a native file container. There are, however, several drawbacks to be taken into account. First of all, current editing tools and metadata editors cannot access such low-level information. It is required to design a proper parser to extract the information stored in the file container. Moreover, these features' effectiveness strongly depends on the availability of updated reference datasets of native and manipulated contents. Eventually, these features cannot discriminate the full history of a media since some processing operations, like uploading to a social network, tend to remove the traces left by previous steps. It is also worth mentioning that the forensic analysis of social media content is fragile since the providers' settings keep changing. This fact makes collected data and achieved results outdated in a short time. Furthermore, collecting large datasets can be challenging depending on the social media provider's upload and download protocol. Future work should be devoted to the fusion of content-based and format-based features since it is expected that the exploitation of these two domains can provide a better performance, as witnessed in Phan et al. (2019), where images exchanged on multiple social networks were studied.

References

Apache. Java mp4 parser. *http://www.github.com/sannies/mp4parser*

Apple Computer, Inc. Quicktime file format. 2001

Carlos Quinto Huamán, Ana Lucila Sandoval Orozco, and Luis Javier García Villalba. Authentication and integrity of smartphone videos through multimedia container structure analysis. *Future Generation Computer Systems*, 108:15–33, 2020

A. Castiglione, G. Cattaneo, and A. De Santis. A forensic analysis of images on online social networks. In *2011 Third International Conference on Intelligent Networking and Collaborative Systems*, pages 679–684, 2011

Dasara Shullani, Marco Fontani, Massimo Iuliani, Omar Al Shaya, and Alessandro Piva. VISION: a video and image dataset for source identification. *EURASIP Journal on Information Security*, 2017(1):15, 2017

David Güera, Sriram Baireddy, Paolo Bestagini, Stefano Tubaro, and Edward J Delp. We need no pixels: Video manipulation detection using stream descriptors. In *International Conference on Machine Learning (ICML), Synthetic Realities: Deep Learning for Detecting AudioVisual Fakes Workshop*, 2019

Duc-Tien Dang-Nguyen, Cecilia Pasquini, Valentina Conotter, and Giulia Boato. Raise: A raw images dataset for digital image forensics. In *Proceedings of the 6th ACM multimedia systems conference*, pages 219–224, 2015

Eric Hamilton. Jpeg file interchange format. 2004

Eric Kee, Johnson Micah K, Hany Farid (2011) Digital image authentication from JPEG headers. IEEE Trans. Inf. Forensics Secur. 6(3–2):1066–1075

Erik Gelbing, Leon Würsching, Sascha Zmudzinski, and Martin Steinebach. Video source identification from mp4 data based on field values in atom/box attributes. In *Proceedings of Media Watermarking, Security, and Forensics 2021*, 2021

H. Farid. Digital image ballistics from jpeg quantization: a followup study. *[Technical Report TR2008-638] Department of Computer Science, Dartmouth College, Hanover, NH, USA*, 2008

Hany Farid. Digital image ballistics from jpeg quantization. *[Technical Report TR9-1-2006] Department of Computer Science, Dartmouth College, Hanover, NH, USA*, 2006

Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008

International Telecommunication Union. Advanced video coding for generic audiovisual services h.264. https://www.itu.int/rec/T-REC-H.264/, 2016

International Telecommunication Union. High efficiency video coding. https://www.itu.int/rec/T-REC-H.265, 2018

ISO/IEC 10918-1. Information technology. digital compression and coding of continuous-tone still images. *ITU-T Recommendation T.81*, 1992

ISO/IEC 14496. Information technology. coding of audio-visual objects, part 12: Iso base media file format, 3rd ed. 2008

ISO/IEC 14496. Information technology. coding of audio-visual objects, part 14: Mp4 file format. 2003

JEITA CP-3451. Exchangeable image file format for digital still cameras: Exif version 2.2. 2002

Jesse D. Kornblum. Using jpeg quantization tables to identify imagery processed by software. *Digital Investigation*, 5:S21–S25, 2008. The Proceedings of the Eighth Annual DFRWS Conference

Jieun Song, Kiryong Lee, Wan Yeon Lee, and Heejo Lee. Integrity verification of the ordered data structures in manipulated video content. *Digital Investigation*, 18:1–7, 2016

JPEG: Still image data compression standard. Springer Science & Business Media, 1992

Leo Breiman. *Classification and regression trees*. Routledge, 2017

Massimo Iuliani, Dasara Shullani, Marco Fontani, Saverio Meucci, Alessandro Piva (2019) A video forensic framework for the unsupervised analysis of mp4-like file container. IEEE Trans. Inf. Forensics Secur. 14(3):635–645

Microsoft. Avi riff file. https://docs.microsoft.com/en-us/windows/win32/directshow/avi-riff-file-reference, 1992

Oliver Giudice, Antonino Paratore, Marco Moltisanti, and Sebastiano Battiato. A classification engine for image ballistics of social data. In Sebastiano Battiato, Giovanni Gallo, Raimondo Schettini, and Filippo Stanco, editors, *Image Analysis and Processing - ICIAP 2017*, pages 625–636, Cham, 2017. Springer International Publishing

Patrick Mullan, Christian Riess, Felix Freiling (2019) Forensic source identification using jpeg image headers: The case of smartphones. Digital Investigation 28:S68–S76

Q. Phan, G. Boato, R. Caldelli, and I. Amerini, Tracking multiple image sharing on social networks. In: 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2019), pages 8266–8270, 2019

Pengpeng Yang, Daniele Baracchi, Massimo Iuliani, Dasara Shullani, Rongrong Ni, Yao Zhao, Alessandro Piva (2020) Efficient video integrity analysis through container characterization. IEEE J. Sel. Top. Signal Process. 14(5):947–954

Raquel Ramos López, Elena Almaraz Luengo, Ana Lucila Sandoval Orozco, and Luis Javier García-Villalba. Digital video source identification based on container's structure analysis. *IEEE Access*, 8:36363–36375, 2020

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986

S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991

SWGDE-SWGIT. SWGDE best practices for maintaining the integrity of imagery, Version: 1.0, July 2017

Thomas Gloe. Forensic analysis of ordered data structures on the example of JPEG files. In *2012 IEEE International Workshop on Information Forensics and Security, WIFS 2012, Costa Adeje, Tenerife, Spain, December 2-5, 2012*, pages 139–144. IEEE, 2012

Thomas Gloe, and Rainer Bähme (2010) The Dresden image database for benchmarking digital image forensics. J Digit Forensic Pract 3(2–4):150–159

Thomas Gloe, André Fischer, Matthias Kirchner (2014) Forensic analysis of video file formats. Digit. Investig. 11(1):S68–S76