Efficient Video Integrity Analysis Through Container Characterization

Pengpeng Yang, *Student Member, IEEE*, Daniele Baracchi, Massimo Iuliani, Dasara Shullani, Rongrong Ni, Yao Zhao, *Senior Member, IEEE*, and Alessandro Piva, *Fellow, IEEE*

***Abstract*—Most video forensic techniques look for traces within the data stream that are, however, mostly ineffective when dealing with strongly compressed or low resolution videos. Recent research highlighted that useful forensic traces are also left in the video container structure, thus offering the opportunity to understand the life-cycle of a video file without looking at the media stream itself. In this article we introduce a container-based method to identify the software used to perform a video manipulation and, in most cases, the operating system of the source device. As opposed to the state of the art, the proposed method is both efficient and effective and can also provide a simple explanation for its decisions. This is achieved by using a decision-tree-based classifier applied to a vectorial representation of the video container structure. We conducted an extensive validation on a dataset of 7000 video files including both software manipulated contents (*ffmpeg*, *Exiftool*, *Adobe Premiere*, *Avidemux*, and *Kdenlive*), and videos exchanged through social media platforms (Facebook, TikTok, Weibo and YouTube). This dataset has been made available to the research community. The proposed method achieves an accuracy of 97.6% in distinguishing pristine from tampered videos and classifying the editing software, even when the video is cut without re-encoding or when it is downscaled to the size of a thumbnail. Furthermore, it is capable of correctly identifying the operating system of the source device for most of the tampered videos.**

***Index Terms*—Video forensics, video container, social media, integrity, authentication, video tampering, decision trees, machine learning.**

I. Introduction

DIGITAL videos are becoming more and more relevant in the communication among users and in providing information. Recent statistics show that the current global average of video consumption per day stands at 84 minutes and it is expected to increase and hit 100 minutes per day by 2021.[1] Therefore, it is not surprising that digital videos are often involved in investigations and other forensic analysis. At the same time, video editing programs, both open source (e.g. *ffmpeg*) and commercial (e.g. *Adobe Premiere*), allow users to easily cut and manipulate videos to create fake contents.

Video Forensics develops algorithms for assessing video integrity and authenticity by looking at the digital traces left during the video life-cycle [1]. Most of the existing video forensic techniques verify the authenticity of a video file by investigating the presence of inconsistencies in pixel statistics. For example, double encoding or manipulation can be detected by analyzing prediction residuals [2] or macroblock types [3]–[5]. Similarly, traces of frame rate up-conversion can be used to prove malicious video processing [6], [7]. Recent works have also successfully employed deep-learning techniques to detect video forgeries [8]. Jamimamul *et al.* [9] focused on interframe video forgery detection by designing a 3D Convolutional neural network. Verde *et al.* [10] introduced a CNN-based approach to detect and localize splicing manipulation by learning video codec traces.

A major drawback of most of those techniques is their high computational cost; furthermore, strong compressions and downsampling often hide forensic traces, thus severely restricting the number of scenarios where those methods can be employed.

Recently, a new research branch highlighted that video integrity.[2] can be determined using information hidden in the whole video file and not just in the video stream [12]. Video files, in fact, are written to disk using a specific structure called container, comprising multiple streams (video, audio, subtitles) and metadata, which are exploited by decoding software to correctly reproduce the video. Guera *et al.* [13] showed how to identify forged videos without looking at the pixel space. To do so, they extracted high level features (multimedia stream descriptors)

Manuscript received December 2, 2019; revised March 26, 2020 and June 10, 2020; accepted June 10, 2020. Date of publication July 8, 2020; date of current version August 24, 2020. This work was supported in part by the National Key Research and Development of China under Grant 2016YFB0800404, in part by National NSF of China under Grants U1936212 and 61672090, in part by AFRL, in part by DARPA under Grant FA8750-16-2-0188, and in part by the Italian Ministry of Education, Universities and Research MIUR under Grant 2017Z595XS. The guest editor coordinating the review of this manuscript and approving it for publication was Mr. Anderson de Rezende Rocha. *(Corresponding author: Rongrong Ni; Alessandro Piva.)*

Pengpeng Yang, Rongrong Ni, and Yao Zhao are with the Institute of Information Science, Beijing Jiaotong University, Beijing 100044, China, and also with the Beijing Key Laboratory of Advanced Information Science and Network Technology, Beijing Jiaotong University, Beijing 100044, China (e-mail: 14120339@bjtu.edu.cn; rrni@bjtu.edu.cn; yzhao@bjtu.edu.cn).

Daniele Baracchi and Dasara Shullani are with the Department of Information Engineering, University of Florence, 50139 Florence, Italy (e-mail: daniele.baracchi@unifi.it; dasara.shullani@unifi.it).

Massimo Iuliani and Alessandro Piva are with the Department of Information Engineering, University of Florence, 50139 Florence, Italy, and also with the FORLAB, Multimedia Forensics Laboratory, PIN Scrl, 59100 Prato, Italy (e-mail: massimo.iuliani@unifi.it; alessandro.piva@unifi.it).

Digital Object Identifier 10.1109/JSTSP.2020.3008088

[1][Online]. Available: https://www.oberlo.com/blog/video-marketing-statistics, Accessed on March 2020.

[2]Note that integrity and authenticity are different concepts. Integrity is proved when the imagery is complete and unaltered, from the time of acquisition or generation through the life of the imagery; indeed, content authentication is used to determine whether the visual content depicted in imagery is a true and accurate representation of subjects and events. More details can be found on the Best Practices for Image Authentication of the Scientific Working Group on Digital Evidences [11].

related to video coding using *ffprobe*. However, the tree-shaped container structure is not taken into account, thus discarding most of the overall available information. Iuliani *et al.* [14] highlighted that video integrity can be assessed by looking at the video file container structure, since any post-processing operation alter the content and the position of some atoms and field-values. This approach turned out to be also promising in classifying the source brand of native videos.

However, [14] merely detects a loss of integrity, without providing a human-interpretable explanation of the reasoning behind its decisions. Furthermore, the method has a linear computational cost since it requires to check the dissimilarity of the probe video with all available reference containers. As a consequence, an increase of the reference dataset size leads to a higher computational effort. Furthermore, both [13], [14] do not provide any characterization of manipulated videos, nor any explainability of the achieved outcome.

In this article we introduce an efficient method for the analysis of video file containers that allows both to characterize identified manipulations and to provide an explanation for the outcome. The proposed approach is based on Decision Trees [15], a non-parametric learning method used for classification problems in many signal processing fields. Their key feature is the ability to break down a complex decision-making process into a collection of simpler decisions. We enriched the tool with a likelihood ratio framework designed to automatically clean up the container elements that only contribute to source intra-variability.

With respect to the state of the art, the proposed method, simply called *EVA* from now on, offers new forensic opportunities, such as: identifying the manipulating software (e.g. *Adobe Premiere*, *ffmpeg*, ...); providing additional information related to the original content history, such as the source device operating system.

The process is extremely efficient since a decision can be taken by checking the presence of a small number of features, independently on the video length or size. Furthermore, *EVA* can provide a simple explanation for the process leading to an outcome, since container symbols used to take a decision can be inspected. To the best of our knowledge, this is the first video forensic method with all these desirable traits. Experiments have been performed using videos produced by 34 modern smartphones from some of the most popular brands on the market, e.g. *Apple*, *Samsung*, *LG*, *Huawei*. Tampered contents were generated using both automated processing and manual user operations. This approach allowed us to build a sizeable, realistic dataset. Manipulations include contents generated using *Exiftool*, *ffmpeg*, *Adobe Premiere*, *Avidemux* and *Kdenlive*. Eventually, we investigated whether a container-based approach is effective when dealing with videos exchanged through Facebook, TikTok, Weibo, and YouTube. Overall, the experimental validation involved seven thousands videos. This article is organised as follows: Section II describes the video container standard; Section III introduces the mathematical tools to represent and analyse the video file container; Section IV and V are devoted to the experimental validation of the proposed techniques; finally, Section VI draws some final remarks and outlines future works.

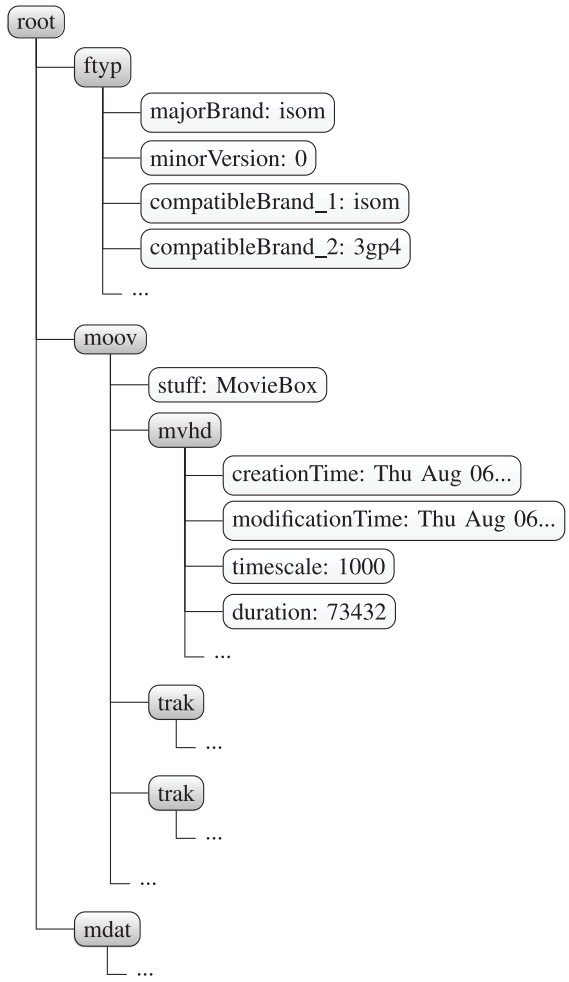


Fig. 1. Pictorial representation of an MP4-like video container structure.

II. Video File Format

Most smartphones and compact cameras output videos in `mp4`, `mov`, or `3gp` format. These video packaging refer to the same standard, ISO/IEC 14496 Part 12 [16], that defines the main features of *MP4* [17] and *MOV* [18] containers while leaving a wide margin for those who implement it. In Fig. 1 we provide an example of an MP4-like container, a tree-like structure describing the video file with respect to three aspects: how the bytes are organized (physical aspect); how the audio/video streams are synchronized (temporal aspect); and how the latter two aspects are linked (logical aspect).

Each node (*atom*) is identified by a unique 4-byte code. It consists of a header which describes its role in the container and possibly some associated data. The first atom to appear in a container has to be `ftyp`,[3] since it defines the best usage and compatibility of the video content. The video structural information is separated from the data itself, indeed the first one is stored in the movie atom (`moov`) and the second one in the media data atom (`mdat`). The `moov` atom links the logical and timing relationships of the video-samples, and provides pointers to their `mdat` location. It is worth noting that the `moov`

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

sub-tree can contain one or more `trak` atoms, depending on the number of streams present in a video (i.e. visual-stream and/or audio-stream).

III. Proposed Approach

We can represent a video container as a labelled tree where internal nodes and leaves correspond to, respectively, atoms and field-value attributes. A video container X can be characterised by the set of symbols $\{s_1, \dots s_m\}$, 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 $s_1, \dots, s_M$ available in the world set of digital video containers $\mathcal{X} = \{X_1, \dots, X_N\}$. Similarly, $\mathcal{C} = \{C_1, \dots, C_s\}$ denotes a set of possible origins (e.g., *Huawei* P9, *Apple* iPhone 6 s). Given a container X, the different structure of its symbols $\{s_1, \dots, s_m\}$ can be exploited to assign the video to a specific class C_u.

For this purpose binary decision trees [19] 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 algorithm. An example is reported in Fig. 3. More specifically, in our approach we adopted the growing-pruning-based Classification And Regression Trees (CART) [20].

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 [21] used to reduce variable-length documents to a fixed-length vectorial representation.

Note that X contains several symbols that are not representative of any class, thus contributing to class intra-variability only (e.g. information related to video length, acquisition date and time). This information is expected to introduce noise in the decision process and it should be possibly removed. Thus, we pre-filtered the data in Ω by using the likelihood ratio framework. Given two classes $C_u, C_v, u \neq v$, and a symbol s_i, the log-likelihood ratio (LLR)

$$\log L_{u,v}(s_i) = \log \frac{P(s_i|C_u)}{P(s_i|C_v)} \tag{1}$$

is computed by approximating the conditional probabilities

$$P(s_i|C_u) = W_{C_u}(s_i)$$
$$P(s_i|C_v) = W_{C_v}(s_i),$$

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

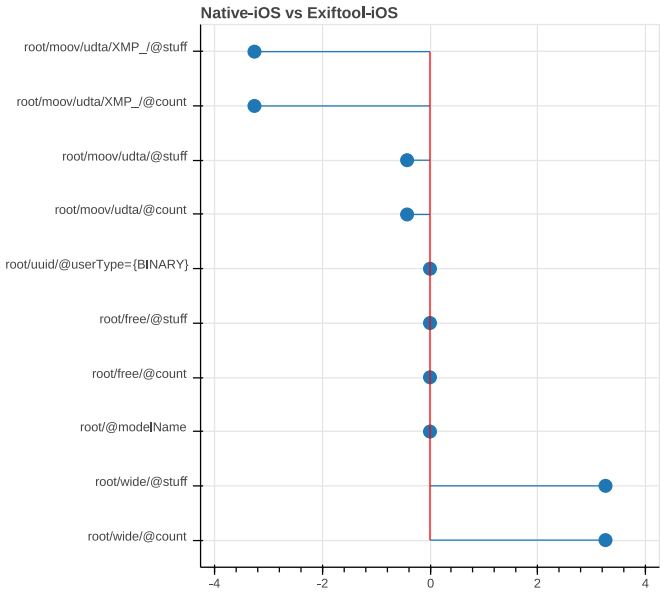


Fig. 2. LLRs of symbols obtained when comparing native videos with ones altered through *Exiftool*. Values far from 0 are automatically included in the analysis. Values close to 0 in all compared classes are excluded from the analysis. Note that @ is used to identify a tree leaf.

where $W_{C_u}(s_i)$ and $W_{C_v}(s_i)$ are the frequencies of s_i in C_u and C_v respectively.[5] The symbol s_i is preserved only if $\exists\, u, v,\ u \neq v : \log L_{u,v}(s_i) > \tau$, with τ a threshold, otherwise it is considered useless and then removed from Ω. It should be noted that using the likelihood ratio we can possibly keep a *field-symbol* while discarding its corresponding *value-symbol*, or vice-versa. In this way we can automatically understand whether the value or the field are relevant for the classification. As an example, we consider two classes:

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

In Fig. 2 some achieved LLRs are reported. The symbols `moov/udta/XMP_/@stuff`, `moov/udta/XMP_/@count`, `wide/@stuff`, `wide/@count` are clearly relevant in identifying this kind of operation on devices equipped with iOS. On the other hand, symbols like `free/@stuff` will be possibly filtered since their LLR is close to zero. In this case, the manipulation only affects a small set of symbols. Indeed, the decision tree can detect such a processing in a single step, by looking, for instance, at the presence of `moov/udta/XMP_/@stuff`, as shown in Fig. 3.

IV. Integrity Verification

The first relevant experimental question is whether the proposed approach is capable of distinguishing between pristine and tampered videos. To answer that we created a new collection of videos, starting from VISION [22], a publicly available dataset

[5]We avoid null frequencies by adding one to both the numerator and the denominator.

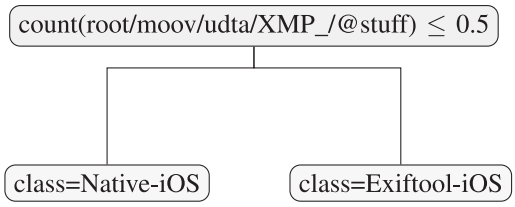


Fig. 3. Decision tree applied to distinguish iOS native videos from iOS videos with modified dates through *Exiftool*. The decision is easily explainable since it is taken by simply looking at the presence of the symbol `moov/udta/XMP_/@stuff`.

that includes native videos from 35 smartphones of 10 different brands. As it would have not been feasible to perform the editing operations, upload, and download of all the videos in VISION, we selected 4 videos for each device, thus obtaining a total of 140 pristine videos. Then, we created 1260 (140×9 editing operations) tampered videos, both automatically generated with *ffmpeg* and *Exiftool*, and manually created through *Kdenlive*, *Avidemux* and *Adobe Premiere*. More specifically:

- **cut with re-encoding:** each video was cut through *ffmpeg* and re-encoded;[6]
- **cut without re-encoding:** each video was cut through *ffmpeg* by copying the audio and video coding parameters[7] to minimize the traces left by the operation;
- **speed up:** each video was speeded up[8] through *ffmpeg*;
- **slow down:** each video was cut through *ffmpeg* and slowed down;[9]
- **cut + downscale:** each video was cut through *ffmpeg* and downscaled[10] to the resolution of 320×240;
- **cut-kd:** each video was manually cut through *Kdenlive* (v17.12.3) by keeping 5 to 7 seconds and then the video was saved with the *MP4 - the dominating format(H264/AAC)* setting;
- **cut-av:** each video was manually cut through *Avidemux* (v2.7.4) by keeping 5 to 7 seconds and then the video was saved as *copy* and *MP4 Muxer* settings;
- **cut-ap:** each video was manually cut through *Adobe Premiere* Pro CC 2019 by keeping 5 to 7 seconds and by saving as H.264 with medium bitrate setting;
- **date change:** each video was manually processed through *Exiftool* (v11.37) to change the date information within the metadata.[11]

We considered *ffmpeg*, *Exiftool*, *Avidemux* and *Kdenlive* for two main reasons:

1) some of them can forge videos in automated way, thus allowing us to create a dataset of tampered videos large enough to obtain statistically significant results;
2) they allow even a novice to create persuasive forged videos, for instance by cutting specific frames, slowing down or speeding up the streams.

Indeed, some real-world forged videos involved such operations. The White House suspended access to CNN's Jim Acosta, after he refused to give up the microphone while asking a question about the Russia investigation at a news conference with President Trump. However, the video reporting the event was possibly speeded up.[12] Another example is a viral clip of Nancy Pelosi that has been edited to give the impression that the Democratic House speaker was drunk or unwell.[13] We also considered videos manually forged with *Adobe Premiere* a proficient video editing tool that can be used by an expert to produce fake contents.

Furthermore, all the produced contents (140 pristine videos and 1260 tampered ones) were exchanged through different social media platforms:

YouTube videos: manual upload on YouTube and automated download through *youtube-dl*[14];

Facebook videos: manual upload and download from Facebook with the 'SD' setting.

Tiktok videos: manual upload and download from TikTok 10.0.0 via a HUAWEI Mate 30 Pro 5 G device with the system of EMUI 10.0.0, Android 10. Several accounts were used to overcome the uploading limitation.

Weibo videos: manual upload to Weibo and automated download using Flvcd.0.4.8.1 (http://www.flvcd.com).

The new dataset thus consists of 7000 videos (from now on *EVA-7 K* Dataset[15]).

The container structure, described in Section II, is extracted from each video by means of the MP4 Parser library [23]. Note that, due to how the dataset was built, some *value-symbols* are always present in some classes even if they are not relevant for their identification. For instance, all the cut videos have the same duration even if this is not, per se, relevant for identifying the editing. As this could lead to artificially higher performance, we manually removed the *value-symbols* associated to the following fields: `@author`, `@count`, `@creationTime`, `@depth`, `@duration`, `@entryCount`, `@entryCount`, `@flags`, `@gpscoords`, `@matrix`, `@modelName`, `@modificationTime`, `@name`, `@sampleCount`, `@segmentDuration`, `@size`, `@stuff`, `@timescale`, `@version`, `@width`, `@height`, `@language`.

It should also be noted that VISION is composed by several iOS/Android devices and a single Windows phone. We removed this latter device (`D17`) from our tests since it is not a representative sample for Windows Phone devices. For this reason, our approach aims to distinguish between iOS and Android

[6]The operation is performed with *ffmpeg* version 3.4.6 through the command `ffmpeg -i $file -ss 00:00:03 -t 00:00:05 -vcodec libx264 -acodec copy $name`

[7]The operation is performed with the command `ffmpeg -i $file -ss 00:00:03 -t 00:00:05 -c copy $name`

[8]The operation is performed through the command `ffmpeg -i $file -vf "setpts=0.25*PTS" $name` for all the other devices.

[9]The operation is performed with the command `ffmpeg -i $file -ss 00:00:03 -t 00:00:15 -vf "setpts=4*PTS" $name`

[10]The operation is performed on with the command `ffmpeg -i $file -ss 00:00:03 -t 00:00:15 -vf scale=320:240 $name`

[11]The operation is performed with the command `exiftool "-AllDates=1986:11:05 12:00:00" $videos`

[12][Online]. Available: https://bit.ly/2vniNi5

[13][Online]. Available: https://bit.ly/2Vx2BGj

[14][Online]. Available: through the command line `youtube-dl -f mp4 -o "%(title)s.%(ext)s" "videos_list_link"`

[15]*EVA-7 K* is available for download from our research group site [Online]. Available: https://lesc.dinfo.unifi.it/en/datasets.

TABLE I
BALANCED ACCURACIES OBTAINED IN THE BASIC SCENARIO FOR EACH DEVICE

Device	Balanced Accuracy	Device	Balanced Accuracy	Device	Balanced Accuracy
D01	1.00	D13	1.00	D26	1.00
D02	1.00	D14	1.00	D27	1.00
D03	1.00	D15	1.00	D28	1.00
D04	0.99	D16	1.00	D29	1.00
D05	1.00	D18	1.00	D30	1.00
D06	1.00	D19	1.00	D31	1.00
D07	1.00	D20	1.00	D32	1.00
D08	1.00	D21	1.00	D33	1.00
D09	1.00	D22	1.00	D34	1.00
D10	1.00	D23	1.00	D35	1.00
D11	1.00	D24	1.00		
D12	0.50	D25	1.00		

TABLE II
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. [13]	0.67	347 s	$<$ 1 s
Iuliani et al. [14]	0.85	N/A	8 s
EVA	0.98	31 s	$<$ 1 s

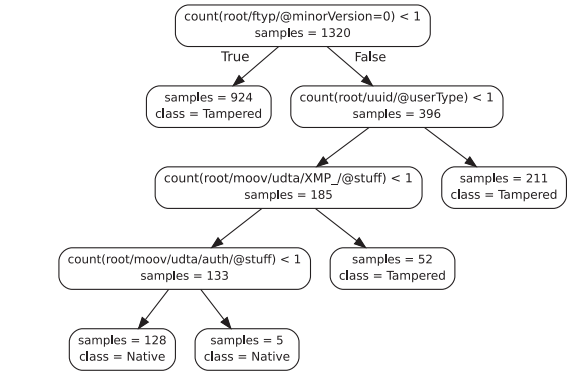
videos only. This is a negligible limitation given that Windows Phone devices represent less than 0.3% of the mobile devices market [24].

In order to estimate the real-world performance of the proposed method we adopted an exhaustive leave-one-out cross-validation strategy. We partitioned our dataset in 34 subsets, each one of them containing pristine, manipulated, and social-exchanged videos belonging to a specific device. We performed each of the experiments hereby described 34 times, each time keeping one of the subsets out as test set, and using the remaining 33 for training our model. In this way, test accuracies collected after each iteration are computed on videos belonging to an unseen device. We reported the mean accuracies obtained among all the iterations as confusion matrices. During the training we assigned to each class a weight inversely proportional to the class frequency. We used the decision trees algorithms included in *scikit-learn* [25], a freely available Python toolkit for machine learning.

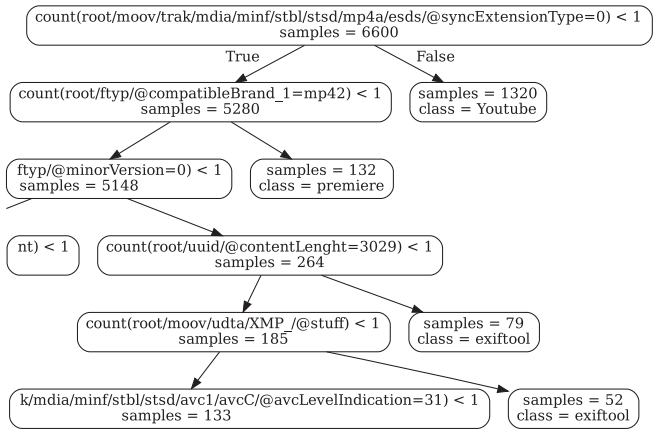
We trained our method to distinguish between the two classes "Pristine" (containing 136 videos) and "Tampered" (containing 1224 videos). We obtained a global balanced accuracy of 98.5%, failing only for videos produced by `D12` (see Table I). The low accuracy obtained on such a device is reasonably due to the fact that it is the sole Sony smartphone in our dataset.

As a consequence of our strict leave-one-device-out strategy, we have no videos belonging to a Sony device in our training set when `D12` is tested. Thus, our algorithm cannot learn the features needed to correctly classify those videos. This limitation does not always apply as different camera models can exhibit very similar containers. In such a case, a native video can be correctly classified even if the specific originating device is unavailable in the training set. This is the case of the LG D290 (`D04`) that reaches an accuracy of 0.99.

We also compared our method with two recently proposed algorithms for video integrity [13], [14]. In Table II we report



(a) Integrity verification classifier.



(b) Detail of a blind scenario classifier.

Fig. 4. Pictorial representation of some of the generated decision trees.

TABLE III
CONFUSION MATRIX FOR THE SOFTWARE IDENTIFICATION SCENARIO

	Native	***Avidemux***	***Exiftool***	***ffmpeg***	***Kdenlive***	***Premiere***
Native	0.97	-	0.03	-	-	-
Avidemux	-	1.00	-	-	-	-
Exiftool	0.01	-	0.99	-	-	-
ffmpeg	-	0.01	-	0.90	0.09	-
Kdenlive	-	-	-	-	1.00	-
Premiere	-	-	-	-	-	1.00

the mean global accuracy and the average runtime per fold for the proposed approach and for those two methods.

A. Discussion

EVA provides several improvements with respect to the state of the art. In comparison with [13] we achieve a higher accuracy. This can be reasonably attributed 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 stronger discriminating power. Indeed, this approach allows to distinguish between two videos where the same information is stored in different atoms. When compared with [14], *EVA* is capable of obtaining better classification performance with a lower computational cost. In [14] $O(N)$ comparisons are required since all the N reference-set examples must be compared

TABLE IV
CONFUSION MATRIX FOR THE SOFTWARE IDENTIFICATION SCENARIO WHEN THE OS IS TAKEN INTO ACCOUNT

		Native		***Avidemux***		***Exiftool***		***ffmpeg***		***Kdenlive***		***Premiere***	
		Android	iOS	Android	iOS	Android	iOS	Android	iOS	Android	iOS	Android	iOS
Native	Android	0.95	-	-	-	0.05	-	-	-	-	-	-	-
	iOS	-	1.00	-	-	-	-	-	-	-	-	-	-
Avidemux	Android	-	-	0.95	0.05	-	-	-	-	-	-	-	-
	iOS	-	-	-	1.00	-	-	-	-	-	-	-	-
Exiftool	Android	0.01	-	-	-	0.99	-	-	-	-	-	-	-
	iOS	-	-	-	-	-	1.00	-	-	-	-	-	-
ffmpeg	Android	0.01	-	0.01	0.05	-	-	0.75	-	-	0.15	0.04	-
	iOS	-	-	-	-	-	-	-	1.00	-	-	-	-
Kdenlive	Android	-	-	-	-	-	-	-	-	0.75	0.25	-	-
	iOS	-	-	-	-	-	-	-	-	0.38	0.62	-	-
Premiere	Android	-	-	-	-	-	-	-	-	-	-	0.79	0.21
	iOS	-	-	-	-	-	-	-	-	-	-	0.37	0.63

TABLE V
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

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. 4(a) a sample tree from the integrity verification experiment: 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`. We also report in Fig. 4(b) a tree from the blind scenario experiment: in this case the tree needs to check the absence of just one atom to classify a YouTube video; at the same time a series of more complex checks are used to assign a video to other classes. This shows how a single decision tree can handle both easy- and hard-to-classify cases at the same time. Neither [14] nor [13] provide an equivalent feature. Moreover, *EVA* is equipped with a formal likelihood ratio framework that can estimate the relevance of symbols for specific tasks. This framework has been used to automatically remove symbols that only contribute to class intra-variability.

V. MANIPULATION CHARACTERIZATION

We also performed a set of experiments designed to show that the proposed method, as opposed to the state of the art, is also capable of identifying the manipulating software and the operating system of the originating device. More specifically, we tried to answer the following questions:

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 have been exchanged through a social media platform, is it possible to retrieve some information on the video origin?

A. Software Identification

In this scenario we only analyze videos that either are native, or that have undergone a manipulation. This time, however, we trained our algorithm to classify which software has been used to tamper the video, if any. Our classes are thus: "native" (136 videos), "*Avidemux*" (136 videos), "*Exiftool*" (136 videos), "*ffmpeg*" (680 videos), "*Kdenlive*" (136 videos), and "*Premiere*" (136 videos).

In this experiment *EVA* obtained a global balanced accuracy of 97.6%; the detailed results reported in Table III show that the algorithm achieved a slightly lower accuracy in identifying *ffmpeg* with respect to the other tools. This is reasonably due to the fact that *ffmpeg* library is used by other software and, internally, by Android devices.

We also trained our algorithm to classify both the editing software used to tamper the video, if any, and the operating system of the device originally used for the acquisition. The classes for this scenario are: "Android-native" (84 videos), "iOS-native" (52 videos), "Android-avidemux" (84 videos), "iOS-avidemux" (52 videos), "Android-exiftool" (84 videos), "iOS-exiftool" (52 videos), "Android-ffmpeg" (420 videos), "iOS-ffmpeg" (260 videos), "Android-kdenlive" (84 videos), "iOS-kdenlive" (52 videos), "Android-premiere" (84 videos), and "iOS-premiere" (52 videos).

A summary of the results obtained by this experiment is reported in Table IV. Our approach maintains good performance in correctly identifying the editing software. We notice, however, that the operating system used for videos manipulated with *Kdenlive* or with *Adobe Premiere* is often misclassified. At the same time, both those programs are always identified correctly. This indicates that the container's structure of videos saved by *Kdenlive* and *Adobe Premiere* is probably reconstructed in a software-specific way.

TABLE VI
CONFUSION MATRIX FOR THE BLIND SCENARIO

		Native		***Avidemux***		***Exiftool***		***ffmpeg***		***Kdenlive***		***Premiere***		**Facebook**	**TikTok**	**Weibo**	**YouTube**
		Android	iOS	Android	iOS	Android	iOS	Android	iOS	Android	iOS	Android	iOS				
Native	Android	1.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	iOS	-	1.00	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Avidemux	Android	-	-	0.95	-	-	-	-	-	-	-	-	-	-	-	0.05	-
	iOS	-	-	-	1.00	-	-	-	-	-	-	-	-	-	-	-	-
Exiftool	Android	0.01	-	-	-	0.99	-	-	-	-	-	-	-	-	-	-	-
	iOS	-	-	-	-	-	1.00	-	-	-	-	-	-	-	-	-	-
ffmpeg	Android	0.05	-	0.01	-	-	-	0.75	-	-	0.15	-	-	-	-	0.05	-
	iOS	-	-	-	-	-	-	-	1.00	-	-	-	-	-	-	-	-
Kdenlive	Android	-	-	-	-	-	-	-	-	0.75	0.25	-	-	-	-	-	-
	iOS	-	-	-	-	-	-	-	-	0.38	0.62	-	-	-	-	-	-
Premiere	Android	-	-	-	-	-	-	-	-	-	-	0.80	0.20	-	-	-	-
	iOS	-	-	-	-	-	-	-	-	-	-	0.37	0.63	-	-	-	-
Facebook		-	-	-	-	-	-	-	-	-	-	-	-	1.00	-	-	-
TikTok		-	-	-	-	-	-	-	-	-	-	-	-	-	1.00	-	-
Weibo		-	-	-	-	-	-	-	-	-	-	-	-	-	-	1.00	-
YouTube		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1.00

B. Integrity Verification on Social Media

In this scenario we tested YouTube, Facebook, TikTok and Weibo videos to determine whether they were pristine or manipulated prior the upload.

A summary of the results obtained by our method is reported in Table V. We achieved global balanced accuracies of 0.76, 0.80, 0.79, and 0.60 on Facebook, TikTok, Weibo, and Youtube, respectively. Such results are characterised by low true negative rates, and thus it cannot be considered effective in this scenario, as many tampered videos are incorrectly classified as pristine.

The poor performance are mainly due to the social media transcoding process that flattens the containers almost independently on the video origin. As an example, after YouTube transcoding, videos produced by *Avidemux* and by *Exiftool* have exactly the same container representation. We do not know how the videos are processed by the considered platforms 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 multiple resolutions and bitrates. Thus, it seems plausible that those operations could discard most of the original container structure.

C. Blind Scenario

In this scenario we considered videos that may or may not have been exchanged through a social media platform and we would like to extract the most complete information possible. We used all the videos in our dataset and we trained our classifier to distinguish (i) whether the video was downloaded from a social media platform; (ii) whether the video was tampered and, if so, which software was used; (iii) whether the original video belonged to an Android or iOS device.

A summary of the results obtained by our method is reported in Table VI. Even without any prior knowledge of the video origin, we are still able to distinguish between native and tampered videos. Our method is also able of correctly identifying videos belonging to YouTube, Facebook, TikTok and Weibo, even though in those cases it is not possible to make further claims on the video authenticity. In most cases we are also able to correctly classify the operating system of the source device.

VI. CONCLUSION

In this article we proposed an efficient forensic method for checking video integrity. If a manipulation is detected, the proposed method allows to identify the editing software and, in most cases, whether the original video belonged to an Android or iOS device.

This is achieved by exploiting a decision tree classifier applied to a vector based representation of the video container structure, enriched with the likelihood ratio framework that is employed to automatically remove container elements that only contribute to source intra-variability. The proposed method, in case of tampered videos, is able to characterise the software that performed the manipulation with an accuracy of 97.6%, even when the video is cut without re-encoding. Except for manipulations performed with *Adobe Premiere* and *Kdenlive*, the proposed method correctly determines the operating system of the video source device.

As opposed to the state of the art, the proposed method is extremely efficient and can provide a simple explanation for its decisions. A new experimental dataset of 7000 videos was also created and shared with the research community, including contents generated with five editing tools (*ffmpeg*, *Exiftool*, *Adobe Premiere*, *Avidemux*, and *Kdenlive*) and four social media platforms (Facebook, TikTok, Weibo and Youtube). The current limitation of the method is that a container-based approach can identify whether the video belongs to a social medial platform like YouTube, Facebook, TikTok or Weibo, but it cannot be effectively applied on such contents for authenticity assessment, since the transcoding operation wipes out most of the forensic traces from the video container. Future works will aim to improve our tool by adding handcrafted features to improve the performance on social media contents.

ACKNOWLEDGMENT

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as having to do with the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory and the Defense Government. Pengpeng Yang would like to

acknowledge the China Scholarship Council, State Scholarship Fund, that supports his joint Ph.D program.

REFERENCES

[1] S. Milani *et al.*, "An overview on video forensics," *APSIPA Trans. Signal Inf. Process.*, vol. 1, pp. 1229–1233, 2012.

[2] T. Shanableh, "Detection of frame deletion for digital video forensics," *Digit. Investigation*, vol. 10, no. 4, pp. 350–360, 2013.

[3] D. Vázquez-Padın, M. Fontani, T. Bianchi, P. Comesaña, A. Piva, and M. Barni, "Detection of video double encoding with GOP size estimation," in *Proc. IEEE Int. Workshop Inf. Forensics Secur.*, 2012, pp. 151–156.

[4] A. Gironi, M. Fontani, T. Bianchi, A. Piva, and M. Barni, "A video forensic technique for detecting frame deletion and insertion," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2014, pp. 6226–6230.

[5] D. Vázquez-Padín, M. Fontani, D. Shullani, F. Pérez-González, A. Piva, and M. Barni, "Video integrity verification and GOP size estimation via generalized variation of prediction footprint," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 1815–1830, Nov. 2019.

[6] X. Ding, Y. Gaobo, R. Li, L. Zhang, Y. Li, and X. Sun, "Identification of motion-compensated frame rate up-conversion based on residual signal," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 7, pp. 1497–1512, Jul. 2017.

[7] M. Xia, G. Yang, L. Li, R. Li, and X. Sun, "Detecting video frame rate up-conversion based on frame-level analysis of average texture variation," *Multimedia Tools Appl.*, vol. 76, no. 6, pp. 8399–8421, 2017.

[8] D. D'Avino, D. Cozzolino, G. Poggi, and L. Verdoliva, "Autoencoder with recurrent neural networks for video forgery detection," *Electron. Imag.*, vol. 2017, no. 7, pp. 92–99, 2017.

[9] J. Bakas and R. Naskar, "A digital forensic technique for inter–frame video forgery detection based on 3D CNN," in *Proc. Int. Conf. Inf. Syst. Secur.*, 2018, pp. 304–317.

[10] S. Verde, L. Bondi, P. Bestagini, S. Milani, G. Calvagno, and S. Tubaro, "Video codec forensics based on convolutional neural networks," in *Proc. 25th IEEE Int. Conf. Image Process.*, 2018, pp. 530–534.

[11] S. W. G. on Digital Evidence, "SWGDE Best Practices for Image Authentication," 2018. [Online]. Available: https://www.swgde.org/documents/, Accessed on: Nov. 12, 2019.

[12] T. Gloe, A. Fischer, and M. Kirchner, "Forensic analysis of video file formats," *Digit. Investigation*, vol. 11, pp. S68–S76, 2014.

[13] D. Güera, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp, "We need no pixels: Video manipulation detection using stream descriptors," in *Proc. Int. Conf. Mach. Learn., Synthetic Realities: Deep Learn. Detecting AudioVisual Fakes Workshop*, 2019.

[14] M. Iuliani, D. Shullani, M. Fontani, S. Meucci, and A. Piva, "A video forensic framework for the unsupervised analysis of MP4-like file container," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 3, pp. 635–645, Mar. 2018.

[15] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[16] *Information Technology—Coding of Audio-Visual Objects, Part 12: Iso Base Media File Format, 3rd ed*, ISO/IEC 14496, ISO, Geneva, Switzerland 2008.

[17] *Information Technology—Coding of Audio-Visual Objects, Part 14: MP4 File Format*, ISO/IEC 14496, ISO, Geneva, Switzerland, 2003.

[18] *Quicktime File Format*, Apple Computer, Inc., Cupertino, CA, USA, 2001.

[19] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 3, pp. 660–674, May/Jun. 1991.

[20] L. Breiman, *Classification and Regression Trees*. Evanston, IL, USA: Routledge, 2017.

[21] H. Schütze, C. D. Manning, and P. Raghavan, "Introduction to information retrieval," in *Proc. Int. Commun. Assoc. Comput. Mach. Conf.*, 2008, vol. 39.

[22] D. Shullani, M. Fontani, M. Iuliani, O. Al Shaya, and A. Piva, "VISION: A video and image dataset for source identification," *EURASIP J. Inf. Secur.*, vol. 2017, no. 15, 2017, doi: 10.1186/s13635-017-0067-2.

[23] Apache, "Java mp4 parser," [Online]. Available: http://www.github.com/sannies/mp4parser, Accessed on: Jul. 2020.

[24] "Statcounter: Globalstats 1999-2020," [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide, Accessed on: Jul. 2020.

[25] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.