SIGMATA: Storage Integrity Guaranteeing Mechanism against Tampering Attempts for Video Event Data Recorders

Hyuckmin Kwon, Seulbae Kim, Heejo Lee
Department of Computer Science and Engineering, Korea University
Seoul, Republic of Korea

ABSTRACT

The usage and market size of video event data recorders (VEDRs), also known as car black boxes, are rapidly increasing. Since VEDRs can provide more visual information about car accident situations than any other device that is currently used for accident investigations (e.g., closed-circuit television), the integrity of the VEDR contents is important to any meaningful investigation. Researchers have focused on the file system integrity or photographic approaches to integrity verification. However, unlike other general data, the video data in VEDRs exhibit a unique I/O behavior in that the videos are stored chronologically. In addition, the owners of VEDRs can manipulate unfavorable scenes after accidents to conceal their recorded behavior. Since prior arts do not consider the time relationship between the frames and fail to discover frame-wise forgery, a more detailed integrity assurance is required. In this paper, we focus on the development of a frame-wise forgery detection mechanism that resolves the limitations of previous mechanisms. We introduce SIGMATA, a novel storage integrity guaranteeing mechanism against tampering attempts for VEDRs. We describe its operation, demonstrate its effectiveness for detecting possible frame-wise forgery, and compare it with existing mechanisms. The result shows that the existing mechanisms fail to detect any frame-wise forgery, while our mechanism thoroughly detects every frame-wise forgery. We also evaluate its computational overhead using real VEDR videos. The results show that SIGMATA indeed discovers frame-wise forgery attacks effectively and efficiently, with the encoding overhead less than 1.5 milliseconds per frame.

Keywords: VEDR, Car Black Box, Storage Integrity, Chronological I/O, and Forgery Detection.

1. INTRODUCTION

Currently, the sales and market scale of video event data recorders (VEDRs) are steadily increasing [1]. VEDRs, also known as car black boxes, are devices that are installed in a vehicle to record the view through the windshield of the vehicle while it is being driven (some models continue to record while the vehicle is parked). They also save the recorded video stream to storage as a file. Since a VEDR records the view in front of the vehicle, the video data constitute the most important evidence in the investigation of an accident. Therefore, a method of detecting any tampering with the stored data in the VEDR is essential to the integrity of any investigation.

Since VEDRs incorporate storage for the video files, their integrity has to be treated specially. Most frequently, adversaries try to interfere with the video frames. One may insert, delete, replace, or reorder one or more frames in the original video file in order to fabricate evidence of crimes. Thus, we introduce a concept of "frame-wise integrity" in this paper, which indicates the preservation of the existence, time information, and chronological relationship of all the recorded frames. Studies have been conducted on file system integrity or integrity assurance in general, but studies in which the frame-wise integrity is considered do not exist. Thus, a study that attempts to address frame-wise forgery detection and covers the intra- and inter-file chronological relationship is required.

Our mechanism, SIGMATA, that is, "Storage Integrity Guaranteeing Mechanism against Tampering Attempts," is a robust video forgery detection mechanism that ensures frame-wise integrity against forgery attempts. To detect any frame-wise tampering flawlessly, an information about the chronological order of original frames needs to be securely maintained. Thus, SIGMATA processes each frame and stores the resulting sequence of integrity assurance values (IAVs), which are subsequently used for verifying integrity. During the process, each frame's byte-sequence is augmented by the size of previous frame, and hashed after appending different salts. The salts are generated by applying another hash function to the elements of one-way hash chain, which renders our mechanism resistant to successive exposure of salts. If an adversary tampers with one or multiple frames, SIGMATA produces a different sequence of IAVs. It can detect forgery by comparing the current sequences with the stored IAV sequence. If a salt is discovered, the exposure and resulting damage does not propagate to other frames. A detailed explanation of the system architecture and principles is provided in Section 4.

We evaluated the effectiveness of SIGMATA based on possible frame-wise forgery attack scenarios, which consisted of insertion, deletion, replacement, and reordering attacks. Moreover, we validated that it is nearly impossible for an adversary to bypass our mechanism even if s/he has full knowledge of the internal principle and operation. In addition, through feature comparison with existing mechanisms that handle file system integrity, we validated that only our mechanism can reveal the frame-wise forgery, thus being the best fit for integrity in the VEDR environment. Furthermore, through performance evaluation using real videos from VEDR, we validated the efficiency of SIGMATA with which the encoding time for videos were incremented by average 1.26% per frame.

The contributions of this paper include:

- A concept of frame-wise integrity that is specific to the VEDR file system is proposed for the first time.
- The design of a thorough integrity assurance mechanism for VEDR storage against frame-wise tampering of video files is described.
- The efficacy of the mechanism is validated by comparing it with that of earlier mechanisms that handle only file-level integrity assurance in various attack-suppression scenarios.

The remainder of this paper is organized as follows. In Section 2, related works that have examined data integrity thus far are presented. In Section 3, we define the problems that lie in prior studies, which we address in this study. In Section 4, we propose and explain in detail the mechanism, SIGMATA. In Section 5, we evaluate the efficacy of SIGMATA by using security analysis and its efficiency by using running examples. In Section 6, the issues of the mechanism are discussed. In Section 7, the conclusion is presented.

2. RELATED WORK

In this section, we address prior arts that address data integrity.

File System-based Approach
Tripwire [2] is a file system integrity checker designed to help UNIX system administrators and users to monitor a designated set of files and directories to discover any changes. It builds up a database, the entries of which contain the filename, inode attributes, and signature information of selected files. When it is called to check integrity, it generates a new database of selected files and compares this with the baseline database to determine changes in the files, which are then reported. This approach provides a good guideline for file system integrity checking. I^3FS [3] is an in-kernel integrity checker and integrity detection file system. It detects unauthorized modifications of files by using cryptographic checksums. L. Catuognol et al. suggested a versioning file system [4]. Although the mechanisms presented in [2-4] are widely used for inspecting integrity, they can determine only whether the file has been changed or not, and neglect the detection of inter-frame forgery, such as frame insertion and replacement, which are specific to VEDR. Cao et al. suggested a method for hashing the files in the storage and sending the hashed data to a remote server to check the integrity through hash value comparison [5]. This approach is not applicable for common VEDRs that do not hold network modules. Lee et al. proposed a scheme which exploits residual data in unused slack space of a storage [6].

Photographic Approach
Researchers have investigated many different photographic approaches for detecting a forgery in a single video file. Shanableh suggested an approach that uses machine learning for application in a method for detecting frame deletion [7]. Kancherla et al. presented a forgery detection method for video that uses Markov models [8]. To improve the performance, they applied the Markov models for residual motion, as obtained from the base frame of the video. Dong et al. proposed a mechanism for detecting frame-based video tampering by using a motion-compensated edge artifact (MCEA), derived from double-MPEG compression [9]. Hyun et al. proposed a mechanism to detect arbitrary cropping and partial manipulation by an attacker by using the extracted sensor pattern noise (SPN), which is unique to each surveillance camera [10]. F. Arab et al. suggested a watermarking technique specific to the AVI formatted videos [11]. These approaches can all detect tampering with a video stream within a file, but are not capable of assessing integrity regarding the inter-file relationship.

3. ASSUMPTION AND PROBLEM DEFINITION

In this section, we define the assumptions and the problems addressed in this paper.

Assumption
Unlike general computing devices and environments, a VEDR has a restricted operating environment and allows user access to the physical device. Thus, we need to define the following assumptions in order to design an integrity assurance mechanism for the VEDR environment.

- **Chronological File I/O.** The video files of a VEDR are created and stored in chronological sequence. When the available storage is exhausted, the least recently recorded files are deleted first.
- **Isolated Device.** We assume the VEDRs do not support any networking features. This means that a remote server that the users cannot reach to verify integrity cannot be utilized.
- **Open Access.** The entire body of the VEDR is in the hands of the users who are simultaneously the adversaries. This means that we grant the adversaries full access to our underlying technique.

Problem: Detecting Frame-wise Forgery in a VEDR file
Frame-wise forgery refers to the action of modifying the byte-sequence of video frames or reordering their temporal sequence. There are four types of such forgery: insertion, deletion, replacement, and reordering of frames. The goal of our research is to resolve the problem above, as it critically affects the investigation of video evidence.

4. PROPOSED MECHANISM

In this section, we describe the architecture and operation of SIGMATA, that is, "Storage Integrity Guaranteeing Mechanism against Tampering Attempts", in detail. To detect frame-wise forgery without network connection, we need a part which is in charge of storing the chronological order of frames during the recording of video, which can constitute up to 24 hours a day. The part is called IAV Generator, and is implemented in the recorder. However, the integrity examination occurs sporadically when it is required, e.g., for the investigation of a car accident. Thus, the other part, Integrity Checker, exists independently with the VEDR, and takes advantage of the formerly generated values for such an occasion.

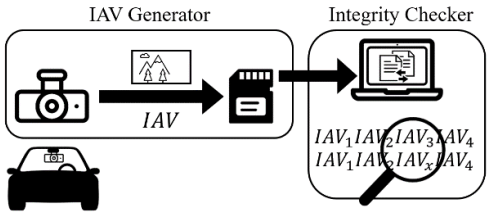

Figure 1. Overall architecture of SIGMATA

Architecture
Figure 1 illustrates the overall architecture of SIGMATA. The assurance value generation part corresponds to the IAV Generator, which transforms the recorded video stream into a sequence of IAVs and saves it in the storage. The integrity verification part corresponds to the Integrity Checker, which performs the actual integrity examination by comparing regenerated IAVs with stored IAVs.

IAV Generator
The IAV Generator produces IAVs from the recorded video stream, and saves the values to storage. It performs the generation while the VEDR is recording the video. The IAV Generator is further broken down into three steps: frame preprocessing, salted hashing, and storage of the computed integrity assurance values. Figure 2(a) describes the IAV Generator and Figure 2(b) shows it pseudocode.

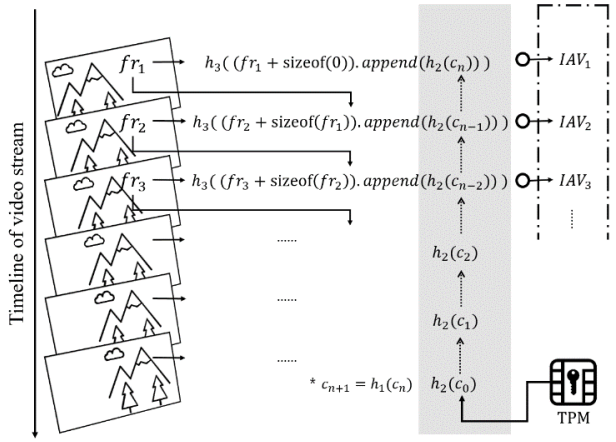

Figure 2(a). Structure of IAV Generator

Input : fr_i: i-th frame of video stream that is being recorded in real-time
h_1, h_2, h_3: Hash functions for one-way hash chain, salt generation, and salted hashing respectively
n: The number of elements of one-way hash chain that is generated each day
c_i: i-th element of the one-way hash chain
$salt_i$: i-th salt used for processing fr_i
Output: IAV_i : i-th integrity assurance value

1 $i = 0$
2 c_0 = GET_FROM_TPM()
3 WHILE(recording of fr_i is not ended) DO
4 $c_{n-i} = {h_1}^{n-i}(c_0)$
5 $salt_i = h_2(c_{n-i})$
6 $IAV_i = h_3\left((fr_i + sizeof(fr_{i-1})).append(salt_i)\right)$
7 STORE(IAV_i)
8 $i = i + 1$

Figure 2(b). Pseudocode of IAV Generator

In the initial step, frame preprocessing, the IAV Generator receives a video frame (fr_i) from the VEDR and adds the size of the previous frame (fr_{i-1}). We call the resulting value an "augmented frame," such that the i-th augmented frame is ($fr_i +$ sizeof(fr_{i-1})).

In the salted hashing step, the IAV Generator first creates a salt, which is appended to the augmented frame, using multiple-key distribution inspired by TESLA [12]. TESLA, a broadcast authentication protocol, generates a chain of keys by repeatedly applying a one-way hash function and reveals the values in the opposite order. Likewise, the Generator generates a one-way hash chain of length n ($c_1, c_2, c_3, \ldots, c_n$) by repeatedly applying hash function $h_1(x)$ to the elements so that the n^{th} element of the chain is a hash of the (n - 1)$^{\text{th}}$ element, i.e., $c_n = h_1(c_{n-1})$. The first element of the chain is securely stored in a storage that an attacker cannot access, e.g., Trusted Platform Module (TPM) [13]. The length n is set to be a sufficient length.

To utilize the chain elements as salts, we apply another hash function $h_2(x)$ to each element. When hashing the frames, the corresponding salt is generated immediately, so that the first frame (fr_1) is encrypted by salt $h_2(c_n)$, the next frame (fr_2) by salt $h_2(c_{n-1})$, and so forth. This "double-sided lock" prevents the adversary from obtaining the rest of the hash chain even when s/he acquires one salt by chance. To elaborate on this, although the chain-generating hash function $h_1(x)$ is exposed to the adversary who identified one of the elements (let us say, c_i) in the hash chain, s/he is not able to generate the prior element (c_{i-1}) because of the irreversibility of the hash function. Moreover, even if an attacker obtains a salt $h_2(c_i)$, of which there is little likelihood, s/he cannot easily determine also the salt for the previous frame $h_2(c_{i+1})$ or the subsequent frame $h_2(c_{i-1})$, since doubly hashed values are not correlated.

As a result of the hashing, each video frame is transformed into an IAV. Finally, in the final phase, the storing step, the consecutive IAVs of the frames are saved in the video storage.

Input : IAV_b: A set of baseline IAVs which has length n
fr_s: A set of submitted frames
IAV_s: A set of IAVs generated by Integrity Checker
Output: is_forged : The value indicating whether a frame is forged

1 IAV_s = IAV_GENERATOR(fr_s)
2 is_forged = FALSE
3 $flag\_deleted$ = FALSE
4 IF length($IAV_b(i)$) ≠ length($IAV_s(i)$) THEN
5 $flag\_deleted$ = TRUE
6 FOR i = 1 TO n
7 IF $IAV_b(i) \neq IAV_s(i)$ THEN
8 is_forged = TRUE
9 IF $flag\_deleted$ = TRUE THEN
10 PRINT("Frame number i was deleted")
11 BREAK
12 ELSE IF $IAV_b(i+1) \neq IAV_s(i+1)$ THEN
13 IF $IAV_b(i+2) \neq IAV_s(i+2)$ THEN
14 PRINT("Frame number i, $i+1$ were rearranged")
15 ELSE THEN
16 PRINT("Frame number i was inserted or replaced")

Figure 3. Pseudocode of Integrity Checker

Integrity Checker

The Integrity Checker performs a comparison of IAV sequences to verify the integrity of the frames on the occasion of an investigation.

When it is called, the Integrity Checker regenerates a sequence of IAVs by processing the video data submitted for investigation through the IAV Generator. Then, it examines the integrity by comparing the new sequence with the baseline sequence that was originally generated and stored. If one of the frames is fabricated, the corresponding IAV value is also changed so that the IAV Checker is able to discover the forgery. Figure 3 is the pseudocode of the IAV Checker.

5. EVALUATION

In this section, we provide several attack-suppression scenarios of frame-wise fabrication to show the effectiveness of our mechanism, and a security analysis, which addresses the possibility that an adversary can bypass our mechanism when s/he is fully conscious of it**.** We also present a feature comparison of our mechanism with general mechanisms for file system integrity or secure file systems. Moreover, we evaluate its performance by comparing processing time with or without SIGMATA in the Raspberry Pi 2 environment.

Frame-wise Forgery Detection

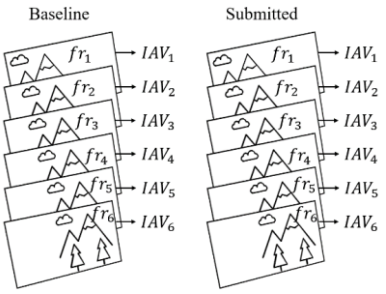

Figure 4. Intact state

Let us assume that the original video consists of six frames, fr_1 to fr_6. The IAV Generator creates a baseline IAV sequence, i.e., $Set\ B = \{IAV_1, IAV_2, IAV_3, IAV_4, IAV_5, IAV_6\}$. If the video is in an intact state, as illustrated in Figure 4, the Integrity Checker obtains the same sequence of IAVs and reports no forgery.

The following subsections describe the rationale behind the detection of each frame-wise attack.

Insertion Attack: As shown in Figure 5(a), a frame (fr_x) is inserted between the second and the third frame. The resulting IAV sequence is $Set\ I = \{IAV_1, IAV_2, IAV_x, IAV'_3, IAV_4, IAV_5, IAV_6\}$. By comparing $Set\ I$ with $Set\ B$, the Checker finds that IAV_x, a previously unseen value, is inserted and IAV_3 is missing in $Set\ I$. Although fr_3 is not modified, we can observe that IAV_3 is changed to IAV'_3, since the size of fr_x is added to fr_3 before hashing. Then, it concludes that an insertion attack has been perpetrated in between fr_2 and fr_3.

Deletion Attack: As shown in Figure 5(b), fr_3 is deleted from the video. The resulting IAV sequence is $Set\ D = \{IAV_1, IAV_2, IAV'_4, IAV_5, IAV_6\}$. By comparing $Set\ D$ with $Set\ B$, the Checker finds that IAV_3 is missing in $Set\ D$. Although fr_4 is not modified, we can observe that IAV_4 is changed to IAV'_4, since the size of fr_2 is added to fr_4 before hashing. Then, the Checker concludes that a deletion attack has been perpetrated in between fr_2 and fr_4, and that the deleted frame is fr_3.

Replacement Attack: As shown in Figure 5(c), fr_3 is replaced with fr_x. The resulting IAV sequence is $Set\ RP = \{IAV_1, IAV_2, IAV_x, IAV'_4, IAV_5, IAV_6\}$. By comparing $Set\ RP$ with $Set\ B$, the Checker finds that IAV_3 and IAV_4 are missing, and that the values match from IAV_5 to the end. Then, it concludes that a replacement attack has been perpetrated in between fr_2 and fr_4. If fr_4 is also modified, IAV_5 should be changed, too.

Reordering Attack: As shown in Figure 5(d), the order of frames is changed. fr_4 precedes fr_3, and then, fr_5 follows. The resulting IAV sequence is $Set\ RO = \{IAV_1, IAV_2, IAV'_4, IAV'_3, IAV'_5, IAV_6\}$. The Checker finds that there is a forgery in between fr_3 and fr_4. The discovery of a reordering attack requires a supplementary inspection to distinguish it from a replacement attack. The Checker uses the fact that IAV'_5 is generated from hashing one of the frames in the forged section and adding the size of another frame in the same section, if it is a replacement attack. In the former example, it checks whether the hash of (fr_3 + sizeof(fr_4)) is equal to IAV'_5. If so, the attack is identified as a reordering attack that swapped fr_3 and fr_4. If not, the attack is a replacement attack that replaced fr_3 and fr_4 with other arbitrary frames, respectively.

Security Analysis

Here, we analyze the security of our mechanism, assuming the adversary has a thorough knowledge of the mechanism. Integrity assurance mechanism, the core of which consists of salted hashing, can be threatened by hash collision since an adversary can neutralize the Checker by deliberately taking advantage of a hash collision to generate the same IAV as the baseline. We logically assess the validity and likelihood of such attack.

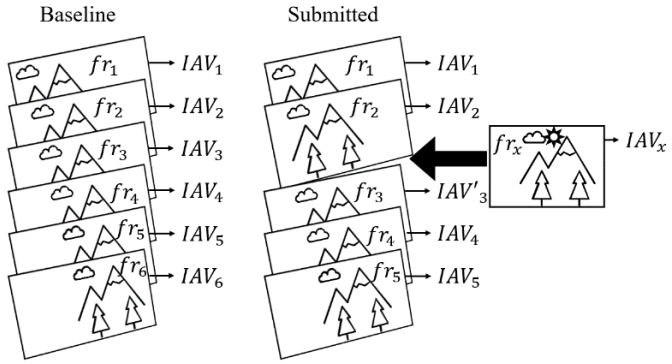

Figure 5(a). Insertion Attack

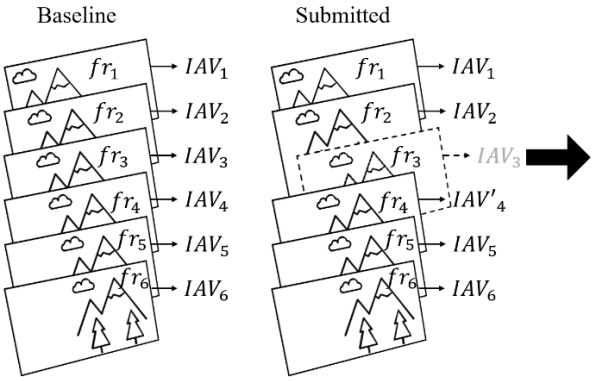

Figure 5(b). Deletion attack

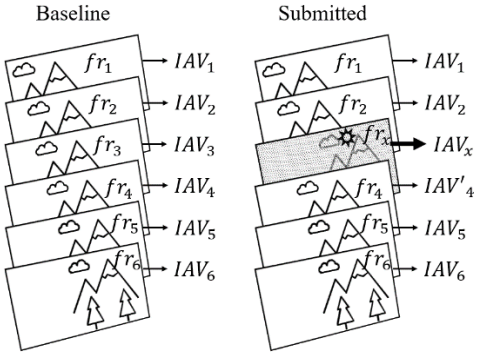

Figure 5(c). Replacement attack

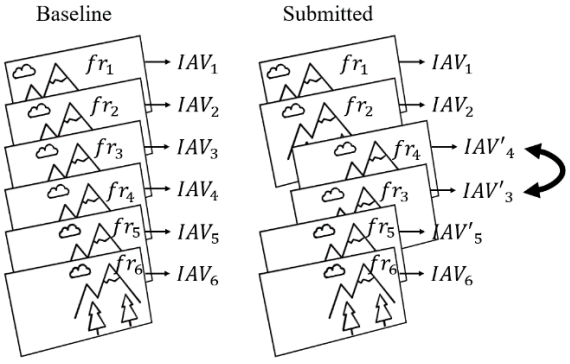

Figure 5(d). Reordering attack

Table 1. Feature Comparison with Other Mechanisms

Feature	NCryptFS [14]	Cao et al. [5]	ICAR, Jerzy et al. [15]	SIGMATA
Detection of frame-wise insertion	No	No	No	Yes
Detection of frame-wise deletion	No	No	No	Yes
Detection of frame-wise replacement	No	No	No	Yes
Detection of frame-wise reordering	No	No	No	Yes
Data recovery	No	No	Yes	No
Storage reusability	Yes	Yes	No	Yes
Network connection	No	Yes	No	No
Implementation layer	Kernel	Application (server-client)	Kernel	Application (Codec)

Generation of Fake IAV: Let us assume the adversary attempts to attack the frame hashing step of the IAV Generator. If s/he determines the byte-stream that causes a hash collision with the original frame, s/he can generate an identical IAV as the baseline. In this case, s/he must satisfy the following three constraints. First, s/he must find a value that causes a hash collision. In addition, the forged frame must be of the same size as the original frame in order not to corrupt the next IAV. Furthermore, the frame that the value represents must be visually valid. However, the intersection of the value set that leads to a hash collision, the value set the length of which is the same as the original frame, and the set that produces a visually complete video frame has not thus far been reported. Thus, we claim that such an attack is impractical.

Feature Comparison

Table 1 shows a comparison of the features.

Frame-wise Tampering: SIGMATA detects every frame-wise tampering attack: insertion, deletion, replacement and reordering. Other mechanisms, i.e., NCryptFS, ICAR, and Cao et al.'s, address only file integrity, and thus, fail to discover specific frame forgery. They also cannot reveal the time and order information of the frames.

Data Recovery and Storage Reusability: SIGMATA, NCryptFS, and Cao et al.'s mechanisms do not provide a data recovery feature. ICAR is able to recover the corrupted data, because a copy of the original data is made at a read-only device. Consequentially, ICAR's storage is not reusable.

Network Connection: SIGMATA, NCryptFS, and ICAR can operate without network connection. However, Cao et al.'s mechanism requires network connection.

Implementation Layer: The implementation of SIGMATA is simpler than that of the other mechanisms, since it does not require kernel-level modification.

Performance

To evaluate the performance of SIGMATA, we compared the encoding time of a raw video stream without SIGMATA with that with SIGMATA. We used three raw video streams recorded by a VEDR (with the H.264 Codec) at a resolution of 1280 × 720 pixels and 30 fps. The videos were 60 seconds, 120 seconds, and 180 seconds long, respectively. MD5, RIPEMD-128 and SHA-1 hash functions were used for h_1, h_2, h_3, respectively.

Assuming a VEDR as a low-end computing machine, we conducted the experiment in Raspberry Pi 2, which has a 900 MHz quad-core ARM Cortex-A7 CPU, 1 GB RAM, and a Micro SD slot, powered by a 5 V micro USB port.

A video file, which recorded the view ahead of a vehicle when it was being driven along a highway, was taken from a VEDR's storage and sliced into three pieces that differ in length. Each piece is named Video 1, Video 2, and Video 3. Figure 6 shows a frame of Video 1, which portrays the driving environment.

Figure 6. A frame of recorded video taken from a VEDR

First, we decoded the videos to get the raw video stream in YUV format. Thereafter, we encoded each raw video twice, once by the unmodified FFmpeg [16], and once by the modified FFmpeg in which SIGMATA was implemented. We used 30 fps, 4:2:0 subsampling, and ultrafast preset of FFmpeg for encoding.

Table 2. Time Comparison for Videos of Various Lengths

Video	Video 1		Video 2		Video 3	
No. of frames	1,800		3,600		5,400	
Frames per second	30		30		30	
Length (sec)	60		120		180	
SIGMATA applied	No	Yes	No	Yes	No	Yes
Encoding time (sec)	149.30	150.33	293.39	297.84	428.58	436.69
Avg. encoding time / frame	0.0829	0.0835	0.0815	0.0827	0.0794	0.0807

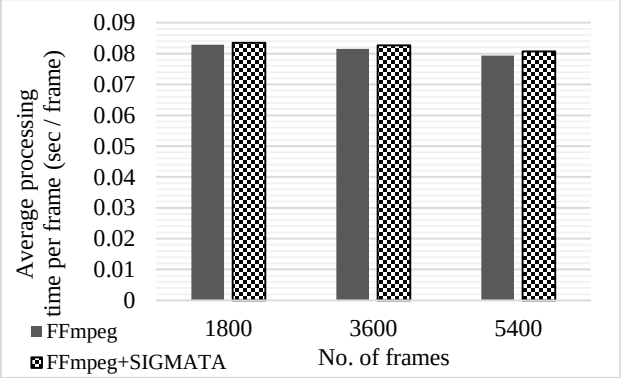

Figure 7. Comparison of the average encoding time per frame

As seen in Table 2, encoding takes less than 0.0015 more seconds per frame using SIGMATA. The critical factors that cause this difference are the salt generation, together with three hash functions applied to each frame. Considering that SIGMATA introduces an average computational overhead of 1.26% for each frame, which is relatively insignificant, SIGMATA is applicable in a real-time scenario.

7. DISCUSSION

In this section, we discuss a few issues of SIGMATA.

Forgery of the first frame

As shown in Figure 2(a), the first frame of the video stream is directly hashed without adding the size of the previous frame, since such a frame does not exist. This may amplify the likelihood of forgery since the size constraint is not necessarily considered during fabrication. However, considering that the first frame occupies only a small portion, 0.033 sec, of the entire video stream spanning 24 hours, this weakness is negligible.

The use of a user-inaccessible storage

In our mechanism, we assume the existence of a secure storage, such as TPM, which is not accessible by the users. We claim that general VEDRs are ready to utilize such hardware, since the ARMv6 architecture, which has supported TrustZone since 2001, is one of the most widespread architectures for embedded processors. For the devices that have no such hardware, there are commercial TPM chips designed for embedded devices such as Atmel AT97SC3203S. T. Winkler et al. make use of the Atmel TPM for their embedded smart cameras [17]. According to T. Winkler, TPM chips are sold at reasonable prices and readily available.

8. CONCLUSION

In this paper, we proposed a novel concept of frame-wise forgery in VEDR storage and a mechanism to assure its integrity. The mechanism resolves several problems, including the detection of insertion, deletion, replacement, and reordering of frames. We verified the utility of our mechanism by investigating attack scenarios and conducting a security analysis of the possibility of bypassing SIGMATA. The results show that SIGMATA is robust against frame-wise forgery attacks and that there is only a slight chance that adversaries can circumvent SIGMATA. Furthermore, we evaluated its performance under Raspberry Pi 2 environment, by running SIGMATA on the videos recorded by VEDRs. The results show that SIGMATA yields near-zero overhead, which means it is applicable to the real-time scenario.

ACKNOWLEDGEMENT

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

REFERENCES

[1] Grand View Research, “North America car DVR market analysis by product (single channel, dual channel) and segment forecasts to 2016”, http://www.grandviewresearch.com/industry-analysis/car-dvr-market, 2014.

[2] G.H. Kim, and E.H. Spafford, “The design and implementation of tripwire: A file system integrity checker”, Proceedings of the 2nd ACM Conference on Computer and Communications Security, 1994.

[3] S. Patil, A. Kashyap, G. Sivathanu, and E. Zadok, “I^{3}FS: An in-kernel integrity checker and intrusion detection file system”, LISA, Vol. 4, 2004, pp. 67-78.

[4] L. Catuognol, H. Lohr, M. Winandy, and A. Sadeghi, “A trusted versioning file system for passive mobile storage devices”, Journal of Network and Computer Applications, 2014, pp. 65-75.

[5] D. Cao, and B. Yang, “Design and implementation for MD5-based data integrity checking system”, The 2nd IEEE International Conference on Information Management and Engineering (ICIME), 2010.

[6] S. Lee, J. Song, W. Lee, and H. Lee. "Integrity verification scheme of video contents in surveillance cameras for digital forensic investigations." IEICE TRANSACTIONS on Information and Systems 98.1 (2015): 95-97

[7] T. Shanableh, “Detection of frame deletion for digital video forensics”, Digital Investigation, Vol. 4, No. 10, 2013, pp. 350-360.

[8] K. Kancherla, and S. Mukkamala. “Novel blind video forgery detection using Markov models on motion residue”, Intelligent Information and Database Systems, 2012, pp. 308-315.

[9] Q. Dong, G Yang, and N. Zhu. “A MCEA based passive forensics scheme for detecting frame-based video tampering” Digital Investigation, Vol. 2, No. 9, 2012, pp. 151-159.

[10] D.K. Hyun, M.J. Lee, S.J. Ryu, H.Y. Lee, and H.K. Lee, “Forgery detection for surveillance video”, The Era of Interactive Media, 2013, pp. 25-36.

[11] F. Arab, S. Abdullah, S. Hashim, A. Manaf, and M. Zamani, “A robust video watermarking technique for the tamper detection of surveillance systems”, Multimedia Tools and Applications, 2015, pp. 1-31.

[12] A. Perrig, R. Canetti, J.D. Tygar, and D. Song, “The TESLA broadcast authentication protocol”, RSA CryptoBytes, 2005.

[13] T. Morris, “Trusted platform module”, Encyclopedia of Cryptography and Security, 2011, pp. 1308-1310.

[14] C.P. Wright, M.C. Martino, and E. Zadok, “NCryptfs: A secure and convenient cryptographic file system”, USENIX Annual Technical Conference, 2003.

[15] J. Kaczmarek, and M. Wrobel, “Modern approaches to file system integrity checking”, 1st International Conference on Information Technology, 2008.

[16] F. Bellard, “FFmpeg multimedia system”, http://www.ffmpeg.org/, 2005.

[17] T. Winkler, and B. Rinner, “Securing embedded smart cameras with trusted computing”, EURASIP Journal on Wireless Communications and Networking, 2011.