Vronicle: Verifiable Provenance for Videos from Mobile Devices

Yuxin (Myles) Liu[*]
University of California, Irvine
Irvine, California, USA
yuxin.liu@uci.edu

Yoshimichi Nakatsuka[*]
University of California, Irvine
Irvine, California, USA
nakatsuy@uci.edu

Ardalan Amiri Sani
University of California, Irvine
Irvine, California, USA
ardalan@uci.edu

Sharad Agarwal
Microsoft
Redmond, Washington, USA
Sharad.Agarwal@microsoft.com

Gene Tsudik
University of California, Irvine
Irvine, California, USA
gene.tsudik@uci.edu

ABSTRACT

Demonstrating veracity of videos is a longstanding problem that has recently become more urgent and acute. It is extremely hard to accurately detect manipulated videos using content analysis, especially in the face of subtle, yet effective, manipulations, such as frame rate changes or skin tone adjustments.

In this paper, we present Vronicle, a method for generating provenance information for videos captured by mobile devices and using that information to verify authenticity of videos. A key feature of Vronicle is the use of Trusted Execution Environments (TEEs) for video capture and post-processing. This aids in constructing fine-grained provenance information that allows the consumer to verify various aspects of the video, thereby defeating numerous fake-video creation methods. Another important feature is the use of fixed-function post-processing units that facilitate verification of provenance information. These units can be deployed in any TEE, either in the mobile device that captures the video or in powerful servers.

We present a prototype of Vronicle, which uses ARM TrustZone and Intel SGX for on-device and server-side post-processing, respectively. Moreover, we introduce two methods (and prototype the latter) for secure video capture on mobile devices: one using ARM TrustZone, and another using Google SafetyNet, providing a trade-off between security and immediate deployment. Our evaluation demonstrates that: (1) Vronicle's performance is well-suited for non-real-time use-cases, and (2) offloading post-processing significantly improves Vronicle's performance, matching that of uploading videos to YouTube.

CCS CONCEPTS

• Security and privacy → Systems security; Mobile platform security; Trusted computing; • Information systems → Data provenance; • Computing methodologies → Image and video acquisition; • Computer systems organization → Cloud computing.

[*]Equal contribution

KEYWORDS

Video provenance, Deepfakes

ACM Reference Format:
Yuxin (Myles) Liu, Yoshimichi Nakatsuka, Ardalan Amiri Sani, Sharad Agarwal, and Gene Tsudik. 2022. Vronicle: Verifiable Provenance for Videos from Mobile Devices. In *The 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22), June 25-July 1, 2022, Portland, OR, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3498361.3538943

1 INTRODUCTION

An increasing number of mobile devices, such as smartphones, tablets, body-worn cameras, and smart glasses, incorporate cameras. They allow users to record videos at any time, which facilitates a wide variety of applications, most notably security-critical applications where videos are used as evidence, or the videos themselves are sensitive. Consider the following examples: (*i*) citizen journalists recording footage of an important event (e.g., a protest), or conducting an interview using smartphones, (*ii*) law enforcement using body-worn cameras to record their interactions with citizens, (*iii*) courts using videos as evidence, (*iv*) electronic legal contract-signing platforms using videos from smartphones to identify signing users [49], (*v*) e-voting (e.g., shareholder or political) platforms using videos from smartphones to identify voters [16], or (*vi*) video-conferencing conducted on smartphones or tablets in government, military, or corporate meetings. Indeed, recent events have further highlighted the importance of these applications [18, 57].

Videos have been used as evidence in legal settings for a long time, since faking them was generally believed to be nearly impossible. However, there has been an increasingly concerning trend of fraudulent videos, so-called *deepfakes*, whereby an attacker either *manipulates* an existing video or *spoofs* one. One recent example is a fake video of the Chief-of-Staff for the prominent Russian political activist (Alexei Navalny) in a Zoom call with Dutch parliamentarians [19]. Another example is a video of the Speaker of the US House of Representatives, Nancy Pelosi, which was doctored to show her struggling with speech [34, 48]. Yet another recent example is a fake video used by a parent to disadvantage cheerleading rivals of her daughter [31]. Unfortunately, such fakes are widely shared on a variety of forums, intermingled with genuine videos, making it hard for consumers to distinguish fact from fiction. The root of the problem is lack of concrete means to ascertain a given video's credibility.

Broadly, there are two approaches to combatting fake videos. One way is to detect them through content analysis [40, 43, 44, 46, 53, 66] and identifying anomalies, e.g., unnatural human eye blinking. Although this approach does not impose any requirements for producing a video, its accuracy is low and will likely get worse due to rapid advances in the deepfake technology [20, 60]. The second approach involves embedding *provenance information* into a video. This is information about the source of the video, i.e., the camera that originated it, and the sequence of all filters applied during post-processing. Provenance information allows a video consumer to check whether it was generated by an authentic camera and processed by a set of acceptable and genuine filters.

In this paper, we focus on the latter since it has the potential to effectively mitigate fake videos by providing hard-to-fabricate, detailed provenance info. Several provenance techniques have been recently proposed. They fall into two categories: the first provides *fine-grained provenance info* though its performance is poor. For example, PhotoProof [50] uses zero knowledge proofs to authenticate transformations applied to an image. However, it takes several minutes to generate a proof of transformation for a single 128×128 image, which is unacceptable for videos, especially, those with higher resolutions found on modern mobile devices. The second category trades off better performance for *coarse-grained* provenance info. For example, AMP [29] allows video producers to sign their content, enabling consumers to verify the identity of the video producer. This provenance info, while useful, requires the consumer to fully trust the video producer since there is neither a proof of the originating camera, nor of how video post-processing was performed prior to signing. Furthermore, this approach is even less effective for videos from mobile devices since it only provides a proof of the originator, and not of content authenticity.

Motivated by the need to improve upon prior techniques, we introduce Vronicle (Video Chronicles), a method for generating provenance info for videos captured by mobile devices and using this info to verify authenticity of videos. There are three major components in the Vronicle architecture: (*i*) mobile device with a camera (camera device for short), (*ii*) video post-processor, and (*iii*) video consumer. The camera device is responsible for capturing the video and generating provenance info about the device. The video post-processor receives the video from the camera device, applies the requested filters, and appends cryptographic proofs for each filter to the provenance info. The video consumer, having received the video and its provenance info, checks whether the video was captured by an authentic camera device, and verifies applied filters along with their parameters and the order in which they were applied.

One key contribution of Vronicle is the use of Trusted Execution Environments (TEEs) for video capture and post-processing as well as for constructing *fine-grained provenance info* that allows the consumer to verify various aspects of the video, thereby defeating many fake-video creation methods.

Another key contribution of Vronicle is how it constructs the post-processor using *fixed function post-processing units*, where each unit can be deployed in a separate TEE and is responsible for applying one filter to the video (or for encoding or decoding the video). The main benefit of this approach is that it simplifies verification of provenance info. Moreover, these units can be deployed in different TEEs. For example, they can be deployed in the camera device itself when offline (on-device) processing is needed. Alternatively, they can be offloaded to powerful servers to achieve *high(er) post-processing performance.*

We present a comprehensive prototype of Vronicle, which uses ARM TrustZone and Intel SGX for on-device and offloaded post-processing, respectively. We also introduce two methods (and prototype the latter) for secure video capture on a mobile device: one that uses ARM TrustZone for enhanced security, and the second that leverages (TrustZone-hardened) Google SafetyNet for immediate deployment. [1]

We conduct a comprehensive evaluation of Vronicle. First, we provide a detailed security analysis and show how Vronicle's provenance info mitigates various attacks, including attacks on the platforms (camera or post-processor) and the video. Second, via performance experiments we demonstrate that, while both on-device and offloaded post-processing offer acceptable performance for non-real-time use cases, the latter outperforms the former significantly. Finally, we show that the performance of offloaded post-processing is on par with uploading videos to YouTube. Specifically, it takes about 44 seconds to upload and post-process (with 6 filters) a 10-second 720p video in Vronicle. Meanwhile, it takes an average of 31 (and maximum of 45) seconds to upload the same video to YouTube, which (we suspect) also performs some post-processing, such as transcoding and copyright checks.

2 BACKGROUND

2.1 Trusted Execution Environment

A Trusted Execution Environment (TEE) is a primitive that protects security-critical code and data from untrusted components. A typical TEE provides the following features:

Isolated Execution: A TEE creates an execution environment isolated from the rest of the system, including the OS and hypervisor. Code running inside a TEE is protected against tampering during execution. Also, data within a TEE are only available to the code running inside the same TEE.

Remote Attestation: A TEE can, upon demand, attest itself by creating a cryptographic proof of the platform authenticity and of the integrity of the code running inside it. Any (local or remote) party can verify this proof to determine whether it can trust that TEE and the code running within it.

Examples of popular TEEs are *Intel Software Guard Extensions (SGX)* [23, 36, 47] and ARM TrustZone [2, 3]. SGX, used in x86 machines, provides *enclaves* to host security-critical code. It supports attestation, enabling a remote party to verify an enclave's identity and confirm that it is a genuine SGX enclave. An enclave's identity is a unique identity called `MRENCLAVE`, which is a cryptographic hash of the code that is loaded into the enclave and other configuration details.

TrustZone provides *a secure world*, which uses a secure OS to host security-critical code in *Trusted Applications (TAs)*. TrustZone does not universally support remote attestation. However, the secure

[1]We open source the prototype for the benefit of users and researchers and provide a video demo showing Vronicle's workflow at: https://trusslab.github.io/vronicle/

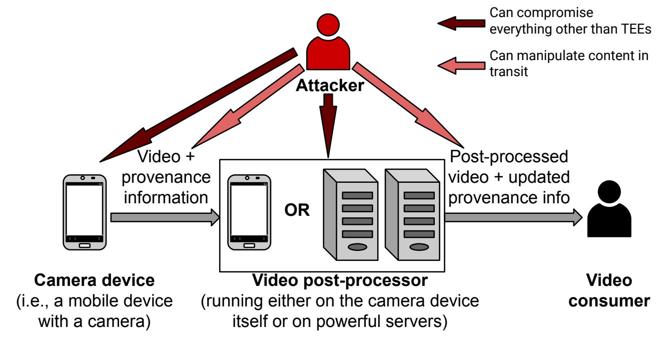

Can compromise everything other than TEEs
Can manipulate content in transit
Attacker
Video + provenance information
OR
Post-processed video + updated provenance info
Camera device (i.e., a mobile device with a camera)
Video post-processor (running either on the camera device itself or on powerful servers)
Video consumer

Figure 1: ***Overview of Vronicle.***

world OS can implement remote attestation, as in Kinibi OS in Samsung devices [25].

However, use of ARM TrustZone to deploy security-critical code faces practical challenges as mobile device vendors control and heavily restrict the set of TAs. Therefore, deploying TAs in commercial devices is not straightforward. An alternative for deploying such code in Android is SafetyNet [7], which helps developers gain trust in users' devices and it is used in security-critical applications such as finance [39]. SafetyNet provides an attestation API [14], which returns a cryptographically-signed attestation report to determine the integrity of the device, its system software, and the requesting app. Recent realizations of SafetyNet use TrustZone to harden its security [62].

2.2 Video Post-Processing

Most videos undergo some post-processing. Modern mobile devices use an Image Signal Processor (ISP) to perform *simple* post-processing tasks, such as demosaicing and white-balancing. Popular video editing software applications support many post-processing filters, such as repairing, enhancing, beautifying, special effects, image scaling, deringing, denoising, deflicking, brightness adjustment, saturation adjustment, contrast adjustment, sharpening, whitening, and lens flare [21]. Filters are employed for a variety of reasons, such as undoing artifacts introduced by the camera hardware in order to make the final video look more "natural" or blurring a face for privacy.

3 VRONICLE DESIGN OVERVIEW

Figure 1 shows the high-level overview of Vronicle. The *camera device* (i.e., a mobile device with a camera such as a smartphone) generates a video. It sends the video and some provenance info to the *video post-processor*. The post-processor applies *a sequence of filters* to the video and updates the provenance info accordingly. Finally, the *video consumer* receives the post-processed video and uses the provenance info to authenticate the camera device that captured the video and the exact sequence of filters applied during post-processing. The *attacker may attempt to manipulate the video on the camera device, on the video post-processor, or in transit. However, the attacker cannot compromise the TEEs used on the camera device and the post-processor.*

The post-processor can either run in the camera device (when on-device, local post-processing is desired) or be offloaded to the cloud (when achieving high performance is critical). We use the TrustZone secure world for on-device post-processing and SGX enclaves for offloaded post-processing.

In addition, we introduce two realizations of the camera device: one where the camera application runs within the TrustZone secure world to securely capture and sign the video (i.e., *TrustZone-camera*), and another, where the camera application uses Google SafetyNet to attest the integrity of the camera device (i.e., *SafetyNet-camera*). The former provides enhanced security and the latter is immediately deployable. We present a prototype of the latter.

4 TEE-BASED PROVENANCE INFO

At the core of Vronicle is the concept of *provenance info*. Its purpose is to allow the consumer to authenticate: (1) the actual camera device that captured the video, and (2) the exact sequence of filters applied during post-processing. Vronicle needs to enable the consumer to verify that the provenance info is authentic, and has not been tampered with.

Our key idea to achieve these goals is to *leverage TEEs both in the camera device to capture the video and in the post-processor to process the video.* The isolated execution environment of TEEs helps secure the execution of video capture and post-processing. Moreover, the attestation feature of TEEs enables the consumer to authenticate the code running in the TEEs and verify the authenticity of provenance info.

Next, we sketch out how our TEE-based provenance info is generated, verified, and used.

4.1 Generating Provenance Info

For the sake of simplicity, for now we consider a single video frame. The camera device uses a TEE to securely capture a frame (F). It then sends the frame along with some provenance info PI to the post-processor.

The provenance info is as follows: PI = {$\mathrm{AR}_{\mathrm{TEE\_cam}}$, F_metadata, $\mathrm{Sig}_{\mathrm{cam}}$}. The first element of PI ($\mathrm{AR}_{\mathrm{TEE\_cam}}$) is the attestation report from the TEE used to capture the frame. It has two key components: (1) identity of the TEE (i.e., an identification of the code running in the TEE, and (2) a certificate for a private key provisioned to (and only to) the TEE. Moreover, the attestation report is signed by the TEE vendor (e.g., Google in the case of SafetyNet). The identity of the TEE helps authenticate it (and hence the camera device) to the consumer.

The second element of PI (F_metadata) is all the important information about the frame needed for inspection by the consumer, e.g., time of capture. The final element ($\mathrm{Sig}_{\mathrm{cam}}$) is a digital signature computed over F and F_metadata using the aforementioned private key. This signature helps protect (and prove to the consumer) the integrity and authenticity of the frame and its info.

Upon receiving the data (i.e., {F, PI}) from the camera device, the post-processor applies a sequence of filters and generates the post-processed frame, F′. It also extends PI to generate PI′, which is as follows: PI′ = {PI, $\mathrm{AR}_{\mathrm{TEE\_pp}}$, filter_metadata, $\mathrm{Sig}_{\mathrm{pp}}$}.

The first element (PI) is included here without modifications. The second element ($\mathrm{AR}_{\mathrm{TEE\_pp}}$) is the attestation report from the

TEE used to execute the post-processing. Similar to the report for the camera device TEE, the report here has two key components: (1) identity of the TEE, and (2) a certificate for a private key provisioned to (and only to) the TEE. Also similarly, the attestation report is signed by the TEE vendor (e.g., Intel in the case of SGX). The identity of the TEE helps authenticate to the consumer the code used for post-processing.

The third element (`filter_metadata`) is all the important information about applied filters, e.g., order of applied filters. The final element ($\mathtt{Sig}_{\mathtt{pp}}$) is a digital signature computed over `PI`, `F′`, and `filter_metadata` using the aforementioned private key. This signature helps protect (and prove to the consumer) the integrity and authenticity of the original provenance info sent from the camera device and included in the extended provenance info, the processed frame, and info about the filters.

4.2 Verifying Provenance Info

The consumer receives the following information: {`F′`, `PI′`}. First, the consumer verifies the attestation reports in `PI′` (e.g., that they are signed by the expected vendors). Second, it inspects and verifies the identities of TEEs included in the attestation reports (e.g., that a legitimate filter is applied). The consumer's goal is to ensure that the right TEEs (i.e., the right TEE platforms and the right code within the TEEs) were used for both capture and post-processing. Third, it validates the certificates in the attestation reports.[2] Fourth, it checks the signature from the post-processor ($\mathtt{Sig}_{\mathtt{pp}}$). However, the consumer does not (and indeed cannot) check the signature from the camera device ($\mathtt{Sig}_{\mathtt{cam}}$) since it does not have access to the original frame `F`. We will address this issue in §6.1. Finally, the consumer checks the info regarding the frame sent from the camera device and the info about the sequence of filters sent from the post-processor, and decides whether to use the post-processed frame or not.

4.3 Supporting Videos

Videos are encoded to reduce size, e.g., using the H.264 encoding. This poses two challenges: (1) image filters typically operate on raw frames, and (2) encoded formats typically include audio. To address this, we include two additional components to the post-processor: decoder and encoder. The former receives the video from the camera device, decodes it and forwards the video frame by frame to image filters. The decoder also extracts the audio from the video. All post-processed frames as well as the audio are then given to the encoder, which encodes them into a video that is ready for the consumer. Figure 2 shows this design. Note that it is possible to perform post-processing on the audio as well, although we do not currently support that in our prototype.

4.4 Provenance Info for Videos

In §4.1, we discussed the components of the provenance info for a single frame. The provenance info for a video includes additional information, shown in Table 1. It includes information about the video, such as the video ID as well as the time of video capture, provided by the camera device. Two additional pieces of info are useful, although our current prototype does not support them: location and depth map. The former is useful when it is important to know *where* content was originally captured. The latter is important to defeat an attack where a video is captured of a deepfake displayed on a screen (§8.1).

[2]This includes multiple checks, including purely cryptographic validity, expiration, revocation, and validation of higher-level PKI certificates.

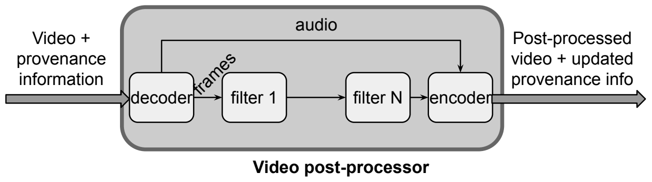

Video + provenance information
audio
frames
decoder
filter 1
filter N
encoder
Post-processed video + updated provenance info
Video post-processor

Figure 2: *High-level view of the post-processor.*

The provenance info also includes information about the video segment. We assume a video is broken into multiple segments and uploaded for post-processing segment by segment. We use segments to help process a video in smaller chunks. In fact, when referring to "a video sent from the camera device," we actually mean "a video segment." Provenance info further includes the list of filters, remote attestation reports of TEEs, e.g., TrustZone TAs, SGX enclaves, or SafetyNet apps.

Table 1 specifies the size of various components of the provenance info, which can be used to calculate the overall size of the provenance info for a video. As an example, the size of the provenance info of a 10-second 1080p video captured in 30 FPS and post-processed with 3 1-parameter filters is 11,194 bytes (80 bytes for Video Info, 12 bytes for Segment Info, 204 bytes for Filter Info, 88 bytes for Encoder/Decoder Info, and 10810 bytes for Camera Device Info).

Table 1 implies the use of different TEEs for filters, decoder, and encoder (as opposed to one TEE for the whole post-processor). We will discuss this design decision in (§6).

5 THREAT/ADVERSARY MODEL

Since Vronicle's primary goal is for consumers to gain trust in videos using verifiable video provenance, we discuss the threat model from the consumer's perspective.

We consider two types of adversaries. The first adversary tries to generate a fake video or tamper with a video produced by Vronicle. The second adversary attacks the camera device or the post-processor. Examples of this adversary are the owner/operator of the camera device, or the operator of the data center where offloaded post-processing happens. Both types aim to produce tampered or spoofed videos that appear genuine to the consumer.

We assume that the first adversary type can create arbitrary videos and provenance info, or can make arbitrary changes to a video produced by Vronicle, e.g., insert frames, drop frames, crop frames, or change the frame rate.

We assume that the second adversary type can control everything outside the TEEs in the camera device and the post-processor including the OS, applications, and network. More specifically, in the case of TrustZone (either for the camera device or post-processor), we assume that TrustZone's hardware and firmware, the secure world OS, the TAs used by Vronicle, and the remote attestation report from the secure world OS are trusted. In the case

Provenance Category	Provenance Component Name	Size (typical / bytes)	Usage
Video Info	Video ID	44	Prevents swapping of frames from different videos
	Timestamp	10	Prevents manipulation of date and time of video
	Location (not prototyped)	18	Helps verify where the video was captured
	Depth map (not prototyped)	Varying	Defeats video of video attack
	Dimensions	8	Prevents manipulation of frame sizes
Segment Info	Segment ID	3	Prevents swapping of frames from different segments
	Total number of segments	3	Prevents dropping of segments
	Frame rate	3	Prevents frame rate attacks (Necessary to encode video correctly)
	Total number of frame(s)	3	Prevents dropping of frames
Filter Info	Number of filter(s) (F)	3	Prevents applying a wrong number of filters
	Ordered list of applied filters	15 × # of F	Prevents applying a wrong order of filters
	Identity of applied filter(s)	44 × # of F	Prevents applying wrong filters
	Parameters (P) of applied filter(s)	8 × # of P	Prevents using wrong filter parameters
Encoder/ Decoder Info	Identity of encoder and decoder	88	Prevents tampering with the decoder or encoder
	Frame tag (i.e., a frame ID) (Added by decoder/removed by encoder)	3	Prevents reordering of frames
Camera Device Info	Identity of the camera device TEE	10810	Prevents incorrect or malicious camera devices

Table 1: ***Components of provenance information in Vronicle. We exclude the digital signatures and certificates from the table for brevity. The data sizes for identities assume SGX enclaves for filters, decoder, and encoder and assume SafetyNet for the camera device.***

of SGX (used for post-processing), we assume SGX hardware and firmware, the enclave codes used by Vronicle, and SGX remote attestation report are trusted. In case of TrustZone-hardened SafetyNet (used for the camera device), in addition to TrustZone, we assume that the SafetyNet's realization on the device (which checks the health of the device, OS, and the requesting app) as well as its remote attestation report are trusted.

Out-of-Scope attacks include: side-channel attacks, Return-oriented Programming (RoP) as well as other code-reuse attacks, physical attacks, and attacks on cryptographic primitives. We also consider Denial-of-Service (DoS) attacks to be irrelevant to this paper, since they have nothing to do with the adversary's goal of producing fake videos. Finally, since Vronicle is primarily concerned with veracity of videos (i.e., their authenticity and integrity), confidentiality and secure distribution of videos (i.e., digital rights management) are out of scope.

6 POST-PROCESSING UNITS

A simple design for the video post-processor is to deploy it within one TEE (e.g., one TrustZone TA or one SGX enclave), which hosts all three components: decoder, filter(s), and encoder. However, this presents two important challenges:

First, it is hard for the consumer to verify that the correct filter binaries are deployed in the TEE. This is because of many possible filter combinations. Since TEE remote attestation measures the code loaded at initialization time, we cannot allow dynamic code loading (otherwise the attacker can run arbitrary code in the TEE without being detected). Therefore, a different TEE binary needs to be generated given a desired set of filters. This would complicate verification, since the consumer would need to re-create the binary and compare its measurement with that in the remote attestation

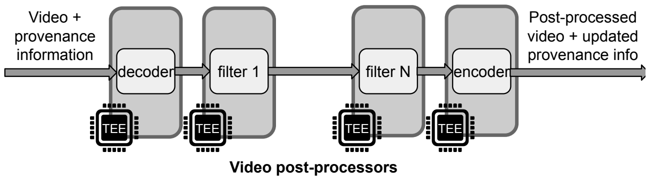

Video + provenance information
decoder
filter 1
filter N
encoder
Post-processed video + updated provenance info
TEE
TEE
TEE
TEE
Video post-processors

Figure 3: ***Fixed-function post-processing units for video post-processing.***

report. Second, it requires generating the TEE binary upon receiving the post-processing request, thus increasing start-up latency.

To address these challenges, we construct the video post-processor using *fixed-function post-processing units*. Each unit is responsible for one filter, or for decoding/encoding the video. The units can then be hosted in separate TEEs (e.g., TrustZone TAs or SGX enclaves). They can post-process the video, each appending its own information to the provenance info. Figure 3 shows how Vronicle uses fixed-function post-processing units. First, the video sent from the camera device is delivered to the decoder TEE, i.e., the TEE hosting the decoder unit. Decoded frames are then forwarded to the sequence of appropriate filter TEEs, i.e., TEEs hosting filter units. The output of the last filter is sent to the encoder TEE, which generates the final post-processed video.

This design addresses the aforementioned challenges: (1) it simplifies verification of remote attestation reports. The consumer only needs to keep a table of measurements for all common filters and decoders/encoders. It can then use this table to check TEE measurements in the report. (2) It relieves the post-processor from having to generate custom TEE binaries for each video, which reduces the overall post-processing latency.

In addition, this design provides two additional benefit. First, fixed-function units can be deployed in different TEEs, either on the device when local processing is needed or in powerful servers when performance is critical. We have ported the same set of post-processing units to execute within TrustZone TAs or Intel SGX enclaves. This design further allows for a combination of these two approaches (i.e., executing some of the filters locally and offloading the rest), although we do not explore this hybrid approach in this paper. Second, in the case of SGX-based post-processing, it enables scaling out the execution of the post-processor to multiple servers, which can help with performance under heavy load. We do not explore this aspect of Vronicle either.

Nonetheless, despite its benefits, our use of fixed-function post-processing units raises some challenges that we address next.

6.1 Minimizing Provenance Info

§3 postponed the discussion of how the consumer can check the signature sent by the camera device in order to verify the integrity of the original video and its info. Doing so requires providing the original video to the consumer, which can double the bandwidth needed to retrieve a video. The use of fixed-function post-processing units further exacerbates this problem, since the consumer needs to check the signatures generated by all units, which would require access to the intermediate, partially-processed videos sent from one unit to another.

We address this problem using *chain verification*, i.e., we rely on each unit to verify integrity of its input. For example, assuming two units, A and B:

(1) A receives the video and its info from the camera device and verifies the signature of these data.
(2) B verifies A's signature on the intermediate video generated (as well as other info) by A.
(3) The consumer verifies B's signature on the final video (as well as other info).

This frees the consumer from checking integrity of data passed from the camera device to the first unit and between units, hence minimizing required bandwidth. One question remains: how can the consumer ensure that the units do perform the aforementioned checks? The consumer can do so since it authenticates the code of each unit (i.e., the code running in the corresponding TEE) via its remote attestation report.

6.2 Flow of Provenance Across Units

While the consumer can establish trust in the TEEs hosting the post-processing units via remote attestation reports, it cannot trust anything outside the TEEs. Moreover, the fixed-function post-processing paradigm requires the video to be split to frames, which go through filter TEEs independently, before getting merged back into the final video in the encoder. This complicates consumer verification of: (1) all frames in the input video being post-processed and included in the final video, (2) preservation of frame ordering in the final video, and (3) completeness and accuracy of the provenance info of the post-processed video. To address this, we carefully

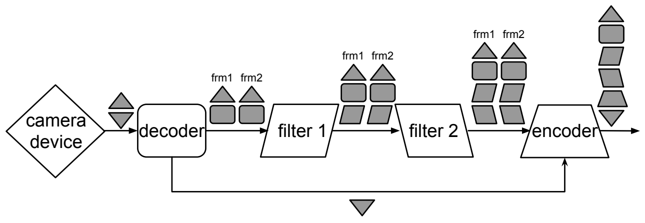


Figure 4: *Flow of provenance info between fixed-function post-processing units.*

design the flow of provenance info between units, as discussed next and illustrated in Figure 4 (for a video with two frames).

Provenance splitting. The decoder receives the video and its provenance info. After decoding the video segment, it sends a part of the video provenance info (along with some new info) with each of the frame to filter TEEs. The part of the provenance info that is passed to the filter TEEs is needed to securely filter and encode the video segment in the encoder. It includes the video ID and segment ID, which help the encoder ensure that it is encoding the frames of the same video segment, as well as the total number of frames, which is used by filter TEEs for sanity checking. Other components of the provenance info can then be sent directly to the encoder. This is especially important for large components of the provenance info, mainly the attestation reports, since otherwise sending them alongside each frame would adversely affect performance. Also, the decoder adds a frame-specific tag (*frame id*) to each frame's provenance. This helps the encoder to encode the frames in the correct order.

Provenance appending. After processing a frame, a filter TEE appends its identity (e.g., SGX enclave *MRENCLAVE* value) to the frame provenance info. This helps the consumer of the video authenticate the filter. Moreover, the decoder and encoder TEEs also add their own identity to provenance info. This is needed by the consumer to verify that the video was correctly decoded and subsequently encoded.

Provenance merging. The encoder receives the post-processed frames along with their provenance info. It verifies the provenance info of each frame by ensuring that their common parts (all except *frame ID*) match. It also checks the *frame ID* and compares it with the total number of frames for correct and complete ordering. It then merges frame provenance into the finalized provenance info for the encoded video, which also includes the provenance info directly sent from the decoder.

7 IMPLEMENTATION

7.1 Video Post-Processor

Vronicle is carefully designed for easy integration into different TEEs. We build an ARM TrustZone-based prototype using OP-TEE to prove the feasibility of secure post-processing in mobile devices and an Intel SGX-based prototype, which we use for offloading post-processing to powerful servers. They both have four components: H.264-based decoder and encoder, filters, and scheduler, where the decoder, encoder, and filters each have a trusted part (which runs inside the TEE) and an untrusted part (which runs outside the TEE).

On the other hand, the scheduler is completely untrusted. Its main role is to manage the execution of other components.

We build a filter framework that allows popular open-source filters to be easily integrated into Vronicle. To demonstrate this, we integrate 7 popular filters: *Blur*, *Sharpen*, *Brightness*, *Grayscale*, *Auto de-noise*, *Auto white balance*, and *Frame erase*.

The frame erase filter needs some discussion. The goal of this filter is to remove a frame from the video, e.g., for privacy reasons. A typical implementation of this filter would drop the frame from the video. In this case, to prevent frame dropping attacks by adversaries, adequate provenance info needs to be added to securely identify the frames dropped by the filter. While possible, this solution complicates the design of the encoder that needs to check presence of all frames and integrity of frame ordering in the video. Therefore, we implement this filter using an alternative, simpler method. Our frame erase filter just zeros out all the pixels of the frame instead. This way, the erased frame still exists in the video (hence making the checks easier in the encoder), yet contains no information at all. Moreover, the consumer can decide what to do with erased frames. For example, it can drop them when viewing the video or can even show them to the user so that the user is aware of dropped frames.

ARM TrustZone vs. Intel SGX. Trusted parts of our units in ARM TrustZone-based post-processor run as TAs in OP-TEE, where each TA is signed and protected by OP-TEE. Note that TAs can be deployed and updated only by the device vendor. As mentioned in §2.1, OP-TEE does not support remote attestation. Therefore, we emulate it by generating a key-pair (and the corresponding certificate) for each TA.

Intel SGX-based post-processor uses enclaves to host the trusted parts. Remote attestation is used to generate each trusted part (enclave)'s key-pair and its certificate. We use remote attestation leveraging Intel Attestation Service (IAS). We also integrate an open-source demultiplexer in the decoder enclave and a multiplexer in the encoder enclave to make our system capable of directly processing an audio-included MP4 container.

Another key difference between the two TEEs is that at the time of implementation, OP-TEE's TAs only support C, whereas SGX's enclaves support both C and C++, resulting in better filter compatibility and optimization in the Intel SGX-based post-processor.

ARM TrustZone Post-Processor API. In our ARM TrustZone based post-processor prototype, each TA can be called using some API (using Android NDK) from an Android application. The android application here is in fact the scheduler that controls the whole flow of data and component executions.

7.2 Camera Device

We report a prototype of the SafetyNet-camera. More specifically, we build an Android application for recording and uploading videos. The application also allows the user to choose the set of filters to be applied.

SafetyNet attestation presents a particular challenge that we address. SafetyNet attests that the OS and the app are not compromised at the time of the check. This creates an opportunity for a TOCTOU attack, whereby the attacker compromises the device (e.g., roots it) **after the check and before** using the camera app. To mitigate this, we deploy a two-report scheme: *(i)* the camera app generates a fresh key-pair and uses the hash of the public key as a nonce to conduct the first round of SafetyNet attestation; *(ii)* the user records a video using the camera app; *(iii)* the app again uses the hash of the public key as a nonce to conduct another round of SafetyNet attestation; *(iv)* the app uses the private key to sign the video as well as its provenance info and then erases that key. Besides preventing TOCTOU attacks, this scheme uses a new key-pair and new attestation reports for each video, thus preventing reuse, which would otherwise raise privacy concerns. Also, note that the SafetyNet attestation report includes a timestamp, which prevents the reuse of the same report. Finally, this scheme binds the public key generated for the video to the app that generated it via the SafetyNet report.

7.3 TrustZone-based Camera

We have not prototyped the TrustZone-camera. The straightforward approach to securely use the camera in TrustZone is to port its device driver to the secure world. This approach would, however, increase the TCB. An enforcement layer can be introduced between hardware and platform software to enable end-to-end secure applications while giving users fine-grained control over their devices [42]. Our own work, Tabellion [49], tries to address this problem by utilizing the hypervisor to secure the memory buffer containing the photo captured by the camera. However, it only supports photos, and not videos. We are also aware of industrial effort to securely take verifiable photos using TrustZone [15].

7.4 Consumer

We build a consumer-facing video application. After downloading the video, the application validates the attestation reports with the corresponding vendors (e.g., Intel in the case of IAS issued certificate) and verifies the signature of the processed video and provenance info using the authenticated certificates. It then checks that the provenance info contains the identity (e.g., *MRENCLAVE* value) of the pre-approved post-processing units: decoder, filters, and encoder as well as that of the camera device TEE (e.g., SafetyNet attestation reports). It then displays the verification status to the user. After successful verification, it plays the video (using the open source VLC video player [1]).

8 SECURITY ANALYSIS

This section discusses a range of potential attacks on Vronicle. We assume the attacker can attempt to manipulate the video on the *camera device*, on the *video post-processor*, and in transit. However, the attacker cannot compromise the TEEs used on the camera device and the post-processor.

8.1 Video Attacks

Note that in the following attacks, we assume that attacker cannot manipulate or spoof the provenance since it would be easily detected.

Frame Modification/Spoofing. An attacker may try to change or spoof the content of some of the frames. This would be easily detected since the provenance info includes a signature computed over the video frames.

Frame Deletion. An attacker may attempt to remove some frames from a video segment. However, since the provenance info includes the *total number of frames in a video segment* (Table 1: Segment Info), the encoder will not produce the final video.

Frame Substitution. An attacker may try to replace some frames in a video segment, e.g., using frames from a different video. Because *video ID* (Table 1: Video Info) in the provenance info would not match that in substituted frames[3], this attack is easily detected.

Frame Cropping. An attacker may try to prevent the consumer from viewing a certain part of the frame by cropping it and falsely declaring the frame as having smaller dimensions. Since provenance info includes frame dimensions (Table 1: Video Info), this manipulation is detected.

Video Segment Omission. An attacker may try to delete an entire video segment. Once again, since the provenance info includes the total number of video segments and segment ID (Table 1: Segment Info), this attack is trivially detected.

Video Segment Substitution. An attacker may try to substitute an entire segment. As the provenance info includes a segment ID (Table 1: Segment Info), this attack is easily detected.

Frame Rate Manipulation. This is the attack used to create a deepfake video of the US Speaker of the House Nancy Pelosi [48], in which the frame rate of the video segment (where she was talking) was reduced. Vronicle detects this attack because the frame rate (Table 1: Segment Info) is part of the provenance info.

Using Undesired Filters. An attacker who controls the untrusted components of the post-processor may try to use filters that are not included in provenance info. Recall that every filter TEE appends its identity to the provenance info (Table 1: Filter Info), thus defeating this attack.

Wrong-Order Filtering. Applying the same set of filters in a different order usually yields a different outcome. An attacker may try to apply filters in an order different from that declared in provenance info. Note that the video producer requests the order of filters. However, we are not concerned with what the producer requested, rather with the provenance info to correctly capture the order that was applied. Since provenance info (covered by the signature) includes the exact order in which filters were applied (Table 1: Filter Info), wrong-order filtering is easily detected.

Video of video. An attacker may use Vronicle's camera device to record a video of a deepfake video displayed on a screen. As mentioned in §4.4, it is possible use the depth map info to defeat this attack. That is, the camera device can record the depth map (Table 1: Video Info), at the same time that it records the video, and include that in the provenance info. The consumer can then detect if the content is recorded off of a flat (or even curved) display. The depth map can be recorded using hardware solutions (e.g., the TrueDepth camera or dual cameras, which are already available in many mobile devices and can record the distance of pixels to the camera [24]), or using software-only solutions [30].

8.2 Platform Attacks

As can be seen, the provenance info protects against various attacks on the video. Therefore, the attacker may try to attack Vronicle's platforms (camera device and post-processor) in the hope of tampering with the provenance info. Next, we discuss the techniques such an attacker may attempt and Vronicle's protection against them.

To compromise the camera device, the attacker may consider to: (1) modify the camera app, (2) root the device, (3) install a custom OS image, (4) install a rootkit, (5) try to bypass the attestation framework, e.g., SafetyNet attestation, (6) exploit a kernel vulnerability, (7) mount a code reuse attack on the kernel, (8) mount a physical attack to extract keys, (9) use a side-channel attack to extract keys, (10) exploit a vulnerability in the TEE, or (11) mount a code reuse attack on the TEE. Our SafetyNet-camera protects against attacks (1)–(5). Our TrustZone-camera further protects against attacks (6) and (7). (8)–(11) are out of scope of our threat model.

To compromise the local post-processor running in TrustZone secure world, the attacker may use the same set of aforementioned attacks. And similarly to TrustZone-camera, the TrustZone-based post-processor protects against attacks (1)–(7) (and the rest are out of scope).

To compromise the SGX-based post-processor, the attacker may: (1) modify the enclave code, (2) try to bypass the enclave attestation framework, (3) exploit the enclave code vulnerabilities, (4) mount a code reuse attack on the enclave code [26, 41], or (5) use a side-channel attack to extract the enclave keys. Our post-processor prototype protects against (1) and (2), while (3)–(5) are out of the scope of our threat model.

Moreover, out-of-scope attacks can be addressed orthogonally. For example, if future SGX versions eliminate currently present side-channels [28, 38, 55], Vronicle would be automatically hardened against attack (5) on the post-processor. Also, vulnerabilities in the TEE code can be addressed by using memory-safety vulnerability checking tools (e.g., TEEREX [27]) or by using an SDK written in a secure programming language (e.g., Rust [63]).

9 PERFORMANCE EVALUATION

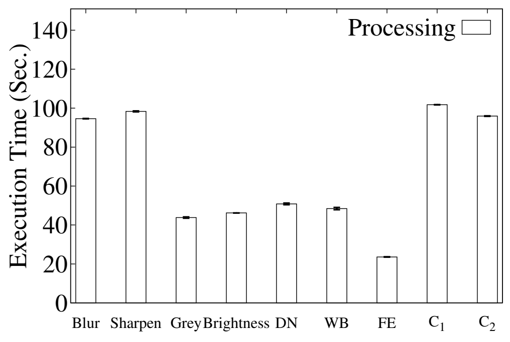
9.1 Setup

ARM TrustZone-based post-processor. We implement Vronicle's ARM TrustZone-based post-processor on a Hikey 620 Development Board. It is equipped with a HiSilicon Kirin 620 SoC, which has 8 ARM Cortex-A53 cores running at 1.2 GHz with 2GB of RAM, and runs OP-TEE 3.8.0 and AOSP 9(P) with a modified Linux kernel 4.14. ARM TrustZone in our configuration can use up to 120MB of memory.

Intel SGX-based post-processor. We realize Vronicle's Intel SGX-based post-processor on a Microsoft Azure Confidential Computing VM. It has a 3.7GHz Intel Xeon E-2288G 8-core processor with 32GB of RAM, and runs Ubuntu 18.04 LTS OS with the Azure Linux kernel 5.4.0. Intel SGX in our configuration can use up to 128MB of memory.

Consumer and camera device. We instantiate the consumer on a machine with a 3.4GHz Intel Core i7-3770 4-core processor and 16GB of RAM with wired Internet connection. This machine runs Ubuntu 20.04 LTS OS with Linux kernel 5.4.0. The camera device is a Samsung Galaxy S20+ smartphone, equipped with a Qualcomm Snapdragon 865 processor and 12GB RAM with 5GHz

[3]Video ID is the hash of the public key provisioned to the camera device TEE, including the modulo and public exponent.



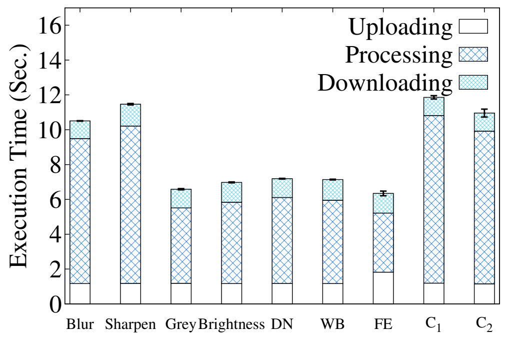


Figure 5: ***Post-processing execution time with different filters in the ARM TrustZone-based post-processor (Left) and the Intel SGX-based post-processor (Right).***

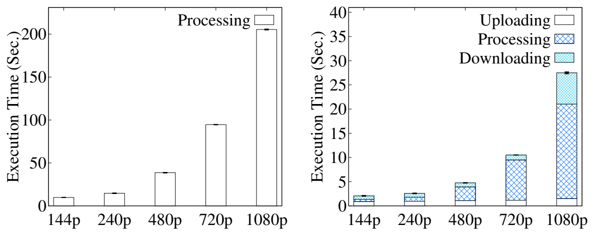


Figure 6: ***Post-processing execution time for videos with different resolutions in the ARM TrustZone-based post-processor (Left) and the Intel SGX-based post-processor (Right).***

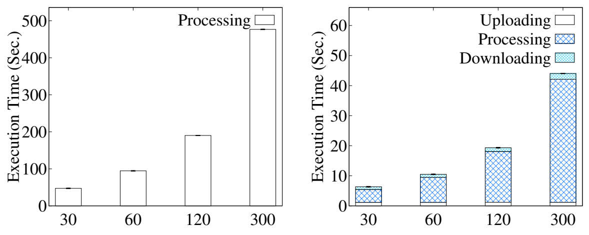


Figure 7: ***Post-processing execution time for videos with a different number of frames in the ARM TrustZone-based post-processor (Left) and the Intel SGX-based post-processor (Right).***

Wi-Fi Internet connection. It runs Android 11 OS, and supports the TrustZone-hardened SafetyNet Attestation API.

Configurations. For reproducibility of experiments, we use a pre-recorded 10-second 1080p video with 334 frames in our camera device. We resize the footage into the following resolutions: 176 × 144 (144p), 320 × 240 (240p), 640 × 480 (480p), 1280 × 720 (720p), and 1920 × 1080 (1080p). As default, we use a 2-second video segment (60 frames) with a 720p resolution, and the blur filter. The 7 filters we use are identical on both prototypes, where the only difference is that the white balance filter is able to use optimized libraries in Intel SGX-based prototype, which is because Intel SGX supports C++ in addition to C. We run each experiment at least 3 times.

The default setting for each filter is as follows: convolution matrix of 7 × 7 for both the blur and sharpen filters and 0.2 decrease for the brightness filter. All other filters (gray, de-noise, white balance, and frame erase) have no configuration parameters. For the TrustZone-based post-processor, we measure its total processing time, which is from when the decoder TA starts initializing to when the encoder TA outputs the result. As the Intel SGX-based post-processor is a server-based prototype, we partition measured latency into three parts: uploading, processing, and downloading. Uploading represents the time from when the scheduler receives a connection from the camera device to when the scheduler finishes receiving all files. A small portion of enclave preparation is also counted in this since we utilize multiple threads. Processing represents the time from when the scheduler finishes receiving all files to when the encoder finishes encoding the last frame. Downloading represents the time from when the encoder receives a connection from the consumer to when the encoder gets the confirmation from the consumer that it has successfully received all files.

9.2 Evaluation Results

Post-processing execution time. Figure 5 shows the execution time of Vronicle for applying various filters to a 2-second video in both platforms. We show the results for applying a single filter (the first 7 bars) and for three-filter combinations (C1 is the sharpen filter followed by the white balance and de-noise filters, and C2 is the blur filter followed by the brightness and gray filters). Results show that Vronicle achieves a decent performance in both setups: it takes about 23.6 and 7.3 seconds to process the video with the most lightweight filter with local and offloaded processing, respectively, and takes about 101.8 and 13.1 seconds for a heavyweight filter combination (C1). In contrast, as we show in §11, prior systems take several orders of magnitude longer to achieve the same. Moreover, these results show that offloading post-processing to powerful servers significantly improves the performance.

We also measure the execution time for different frame sizes or different numbers of frames. Figure 6 and Figure 7 show the results, which show that execution time increases linearly with the increase in either frame size (in terms of the number of frame pixels) or the number of frames in both prototypes.

The overhead of design decisions in Vronicle. Two important design choices in Vronicle are: hosting the post-processor in TEEs and using fixed-function post-processing units. We now evaluate the overhead of each choice using our SGX-based post-processor. To do this, we introduce three baseline implementations of the video post-processor. Baseline1 does away with the fixed-function units and executes all the post-processor components in one SGX enclave. Baseline2 does away with the use of enclaves (and digital signatures) and uses trusting OS processes to host different components of the post-processor. Baseline3 does away with both design decisions and executes all post-processor components in one OS process. We show the results for two filter combinations: one with 2 filters (blur and sharpen) and one with 6 filters (all our filters excluding the frame erase filter). For a fair comparison, we use the same number of threads for filters in all prototypes and use separate threads for decoding and encoding. We use 2 and 6 threads for filters for 2-filter and 6-filter experiments, respectively.

The results are shown in Figure 8. The comparison of Vronicle vs. Baseline1 shows that the use of fixed-function units adds 14.3% and 118.9% performance overhead in the 2-filter and 6-filter configurations, respectively. The major reason for the high performance overhead is due to thread idling in Vronicle. More specifically, in Vronicle, each filter runs as an individual thread, while in Baseline1, all filters run together in one thread (though there are multiple such threads). This means that Baseline1 can make sure that all threads are making use of as much computing resource as possible, whereas, Vronicle may have some threads idling at times. The comparison of Vronicle vs. Baseline2 shows that the use of SGX enclaves adds 44.3% and 62.8% performance overhead in the 2-filter and 6-filter configurations, respectively. Finally, the comparison of Vronicle vs. Baseline3 shows that both design decisions together add 54.7% and 223.4% performance overhead in the 2-filter and 6-filter configurations, respectively. Given the benefits of these design decisions (i.e., the ability to provide verifiable provenance, ease of verification by the consumer, and ease of portability), we believe that their overheads are acceptable.

Finally, note that Baseline1 and Baseline3 require some time to generate the post-processor binary, since it changes depending on the choice of filters. We measure the compilation time from source to binary (assuming that the time to generate the source from existing filters is negligible): 6.5 and 4.1 seconds for Baseline1 and Baseline3, respectively. Note that this overhead is not shown in Figure 8.

Comparison with YouTube. To get a sense of usability of Vronicle's performance, we compare it with YouTube. More specifically, we upload a 10-second video to YouTube and measure the time it takes for the video to be uploaded and processed by YouTube. We suspect the processing includes transcoding the video, as well as several checks, such as copyright checks. We use the same Galaxy S20+ smartphone under the same 5GHz Wi-Fi network for uploading the video. This takes an average of about 31 seconds (with a maximum of 45 seconds and a minimum of 20 seconds). We then upload and post-process the same 10-second video with 6 filters (all our filters excluding the frame erase filter) in Vronicle. This takes an average of about 44 seconds (with a standard deviation of 0.14 seconds). Therefore, we believe Vronicle's performance is good and usable for non-real-time use cases.

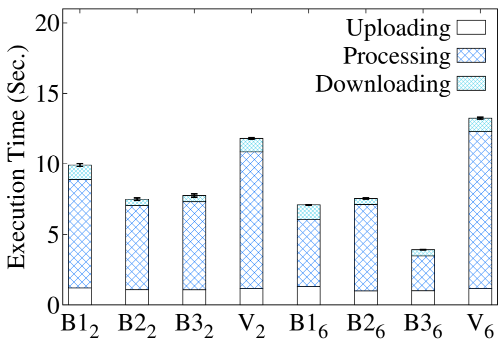


Figure 8: *Post-processing execution time with different design decisions. B1, B2, B3, and V refer to Basline1, Baseline2, Baseline3, and Vronicle. The last number in the x-axis labels shows the configuration. 2 is the 2-filter configuration and 6 is the 6-filter configuration.*

Consumer. We measure the verification time by the consumer. Results show that the consumer takes on average 3.4 seconds (with standard deviation of 0.0012 seconds) to verify the provenance information when SGX and Android SafetyNet are used. This shows that verification is fast enough for consumers not to experience much playback latency.

Power consumption of on-device post-processing. We use a Galaxy S20+ camera device for this experiment. We do not use the HiKey board since it cannot represent the power consumption of a real mobile device. Since we cannot program the TrustZone secure world in a commodity phone, we leverage *Android Native Development Kit* (NDK) [10] to port the Blur filter to run in an Android app and measure the power consumption of executing this filter in the device. We measure the energy consumption of applying this filter to five randomly-generated 2-second video segments using *Batterystats* and *Battery Historian* [6]. Our overall results show that the average energy consumption is 86.77 (standard deviation of 2.59) milliwatt-hours, which is 0.67% of this smartphone's battery capacity assuming the voltage to be 3.7 V.

10 DISCUSSIONS & FUTURE WORK

10.1 Performance Optimizations

Our prototype achieves decent performance for non-real-time use-cases, where the captured video is not needed by the consumer immediately. However, it does not support real-time use-cases, notably video conferencing. We are considering solutions to support real-time videos. One solution is parallelizing the post-processing using multiple instances of the same filter TEEs. We can further make use of distributed computing to split work between multiple machines. Another is to leverage accelerators to increase the performance of our filters. However, the latter would require TEE-enabled accelerators [51, 61], which are not currently available for SGX enclaves.

	Granularity of Provenance	Performance (sec)	Videos?	Method
Alethia [17]	Coarse-grained	No measurements	Yes	Digital Signature
PhotoProof [50]	Fine-grained	673.5 (128x128)	No	Zero Knowledge Proof
YouProve [33]	Coarse-grained	28.0 (JPEG, 1296x972)	No	Fidelity Analysis
FrameProv [22]	Fine-grained	No prototype	Yes	Digital Signature
AMP [29]	Coarse-grained	0.08 ~0.24	Yes	Fragile Watermarking
Vronicle	Fine-grained	4 (30 frames, 1280x720) ~40 (300 frames)	Yes	TEE

Table 2: ***Comparison with related work.***

10.2 Third-Party Verifiers

In Vronicle, video consumers use provenance info to decide whether to trust a video before watching it. However, doing so is not trivial as it requires performing several checks and reasoning about filters and their parameters. While savvy users can do this on their own, we envision a third-party verifier that takes the provenance info, rates the trustworthiness of the video, and makes the result available to end-users. This is similar to how rating agencies in the financial markets rate various financial assets. While more experienced investors can directly study each financial asset to make a decision on their own, most investors simply trust the ratings by well-known and reputable rating agencies. We leave it to future work to develop an algorithm to rate the trustworthiness of a video processed by a sequence of filters.

10.3 Camera Device Privacy

Vronicle relies on the camera device TEE to generate signatures over the captured video and its provenance info using a private key. Usage of this private key might raise concerns regarding camera device privacy since videos signed with the same private key can be linked. Vronicle's SafetyNet-camera prototype addresses this by generating a fresh key-pair for each video, as discussed in §7.2.

We however note that the ability to identify camera devices might be beneficial in certain use-cases. For example, in the case of a video used as evidence in the court, the complete identity of the device that captured the video might be needed. In some other cases, however, it might be adequate to specify the device model, e.g., iPhone 12. We believe that existing privacy solutions, such as k-anonymity [58], can be integrated in our solution.

11 RELATED WORK

Secure Video Provenance. Several video provenance methods have been proposed in the past [17, 22, 29, 32, 35, 56], including watermarking, hash chaining, use of digital signatures, and use of blockchain. Table 2 summarizes key related work and how they compare against Vronicle. Granularity of provenance is defined as "fine" when the system provides information of how the video/photo was edited, and "coarse" otherwise. Below, we describe the key related works.

Some works provide provenance for a single photo only. PhotoProof [50] uses zero knowledge proofs to cryptographically prove that the photo has been edited correctly without revealing the original photo. However, the latency of generating and proving the proof is orders of magnitude slower compared to Vronicle. YouProve [33] analyzes a photo to understand where it was edited.

Some works extend photo provenance to provide provenance for a video. One example is Aletheia [17], a tool that allows users to protect the provenance of their videos by signing them. The main drawback of this system is that it requires consumers to trust the signer without the ability to verify the filters applied to the video. Another example is FrameProv [22], which provides provenance for a video of raw frames using a hashchain of frames. However, it requires trusting the system that conducts video processing and encoding. Several related works have proposed a video provenance system that uses blockchain to prove and assess the provenance of the video [11, 15, 35]. All of these systems suffer from high latency because of their dependency on blockchain. AMP [29] utilizes the Confidential Computing Framework [12] instead of blockchain to build a fast and reliable public ledger that keeps track of published media. AMP uses watermarking methods to embed provenance information into the media and therefore does not allow editing of the videos, as it breaks the watermarked information. The drawback of the system is that it requires trusting the publisher.

Media provenance is gaining attraction not only in academia but also in the industry. An early attempt on trustworthy photo capture in industrial setting was Canon's Original Data Security Kit [4] to prove image originality. However, later on it was proven that there was a flaw in the system [5]. More recently, startup companies such as Truepic [15] and Serelay [9] aim to provide verifiable provenance for photos using TEEs (but not videos). Moreover, we see their trusted cameras as complementary to our system, as it will provide users with a wide variety of trusted camera devices to choose from. In addition, iPhone uses a physically isolated Secure Enclave Processor to securely control the camera [8]. While not the case today, the same design can be used to provide secure provenance as well.

Both Truepic and Serelay follow the Content Authenticity Initiative (CAI) [13] standard. CAI is an organization that aims to establish a standard in media provenance. CAI's vision aligns with Vronicle: providing *fine-grained* provenance information at a *"reasonable" performance*. We believe Vronicle is a step towards achieving CAI's requirements for videos.

Secure Camera and Sensors. Some works utilize Trusted Platform Module (TPM) to implement secure cameras. TrustCAM [65] uses a non-migratable key protected by a TPM on a surveillance camera to sign frames before they are streamed. TrustEYE.M4 [64] goes a step further and applies similar techniques to the image sensing unit, which can effectively shrink the Trusted Computing Base (TCB). Saroiu et al. use Intel Trusted eXecution Technology (TXT) to authenticate different raw sensors' data such as camera and GPS's readings [45, 54]. Their work cannot, however, provide fine-grained provenance for videos, especially when offloaded.

Secure Video Processing using GPU TEEs. We envision utilizing GPU TEEs in Vronicle in the future to provide more complex filters such as edge detection and object recognition. There has been several works to realize TEEs for GPUs, notably Graviton [61] (which is a proprietary emulator) and Telekine [37]. Visor [52] is a secure, privacy-preserving video analytics platform that is built on top of Graviton. It provides confidentiality to the video analytics pipeline and protects the video data from attacks including side-channel attacks. In contrast, in Vronicle, we are not concerned with confidentiality of the video.

Motion-based Video Timestamps Attestation C-14 [59] assures the earliest timestamp when an untrusted drone records a video. It provides a random challenge containing a sequence of motions for the drone to complete, where the footage is then used to verify that the drone has successfully finished all required motions.

12 CONCLUSIONS

We envision a world where consumers can easily verify authenticity and integrity of videos using trustworthy provenance information. Current video veracity techniques are either too coarse-grained in provenance (i.e., prove little to the consumer), or offer poor performance – a matter of minutes for a single small frame.

Vronicle provides precise provenance information on both the original video recording as well as every step of post-processing. Vronicle defends against attackers that control all network communication as well as all software state outside TEEs. One of the prototypes described in this paper can be immediately deployed and used. Our evaluation shows that (1) Vronicle's performance is good and usable for non-real-time use cases and (2) offloading Vronicle's post-processing improves its performance significantly.

ACKNOWLEDGMENTS

The work of UCI authors was supported in part by the NSA Award #H98230-20-1-0345 and UCI ICS Exploration Research Award. The work of Gene Tsudik and Yoshimichi Natakatsuka was also supported in part by NSF Awards SATC-1956393 and CICI-1840197. The authors thank the paper shepherd, Steven Ko, and the anonymous reviewers for their insightful comments.

REFERENCES

[1] 2001. VLC. https://www.videolan.org/vlc/.
[2] 2004. ARM Security Technology, Building a Secure System using TrustZone Technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.
[3] 2004. TrustZone: Integrated Hardware and Software Security: Enabling Trusted Computing in Embedded Systems. In *ARM White Paper*.
[4] 2010. Canon Original Data Security Kit. http://www.canon.co.jp/imaging/osk/osk-e3/index.html.
[5] 2010. Canon Original Data Security System Compromised: ElcomSoft Discovers Vulnerability. https://www.elcomsoft.com/PR/canon_101130_en.pdf.
[6] 2015. Profile battery usage with Batterystats and Battery Historian. https://developer.android.com/topic/performance/power/setup-battery-historian.
[7] 2017. Protect against security threats with SafetyNet. https://developer.android.com/training/safetynet.
[8] 2017. Security certifications for the Secure Enclave Processor. https://support.apple.com/en-gb/guide/sccc/sccca7433eb89/web.
[9] 2017. Serelay: Trusted Media Capture. https://www.serelay.com/.
[10] 2018. Android NDK. https://developer.android.com/ndk/index.html.
[11] 2019. Amber: Instilling trust into video. https://app.ambervideo.co/.
[12] 2019. CCF: A framework for building confidential verifiable replicated services. https://github.com/microsoft/CCF/blob/master/CCF-TECHNICAL-REPORT.pdf.
[13] 2019. Content Authenticity Initiative. https://contentauthenticity.org/.
[14] 2019. Google SafetyNet Attestation API. https://developer.android.com/training/safetynet/attestation.
[15] 2019. Truepic: Photo and video verification you can trust. https://truepic.com/.
[16] 2019. Votz. https://voatz.com/.
[17] 2020. aletheia: Fight fake news by signing your files. https://danielquinn.github.io/aletheia/.
[18] 2020. Impact of COVID-19 on the Video Conferencing Market, 2020 - ResearchAndMarkets.com. https://www.businesswire.com/news/home/20200416005739/en/Impact-COVID-19-Video-Conferencing-Market-2020--.
[19] 2021. Dutch MPs in video conference with deep fake imitation of Navalny's Chief of Staff. https://nltimes.nl/2021/04/24/dutch-mps-video-conference-deep-fake-imitation-navalnys-chief-staff.
[20] 2021. Video: No, Tom Cruise isn't on TikTok. It's a deepfake. https://www.cnn.com/videos/business/2021/03/02/tom-cruise-tiktok-deepfake-orig.cnn-business.
[21] Adobe. 2021. Video effects and transitions in Premiere Pro. https://helpx.adobe.com/lu_en/premiere-pro/user-guide.html/lu_en/premiere-pro/using/video-effects-transitions.ug.html.
[22] M. Ahmed-Rengers. 2019. FrameProv: Towards End-to-End Video Provenance. In *Proc. ACM New Security Paradigms Workshop (NSPW)*.
[23] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proc. ACM International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*.
[24] Apple Developer Article. 2017. Capturing Photos with Depth. https://developer.apple.com/documentation/avfoundation/cameras_and_media_capture/capturing_photos_with_depth.
[25] N. Asokan and A. Paverd. 2021. Remote Attestation. Building trust in things you can't see. Tutorial.
[26] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A. Sadeghi. 2018. The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel SGX. In *Proc. USENIX Security Symposium.*
[27] T. Cloosters, M. Rodler, and L. Davi. 2020. TeeRex: Discovery and Exploitation of Memory Corruption Vulnerabilities in SGX Enclaves. In *Proc. USENIX Security Symposium.*
[28] V. Costan, I. Lebedev, and S. Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *Proc. USENIX Security Symposium.*
[29] P. England, H. S. Malvar, E. Horvitz, J. W. Stokes, C. Fournet, R. Burke-Aguero, A. Chamayou, S. Clebsch, M. Costa, J. Deutscher, S. Erfani, M. Gaylor, A. Jenks, K. Kane, E. Redmiles, A. Shamis, I. Sharma, S. Wenker, and A. Zaman. 2020. AMP: Authentication of Media via Provenance. (2020). arXiv:2001.07886
[30] H. Farrukh, R. M. Aburas, S. Cao, and H. Wang. 2020. FaceRevelio: A Face Liveness Detection System for Smartphones with a Single Front Camera. In *Proc. ACM MobiCom.*
[31] J. Fingas. 2021. Woman allegedly made deepfakes to kick rivals off daughter's cheerleading squad. https://www.engadget.com/woman-creates-deepfakes-for-cheerleader-daughter-215024991.html.
[32] A. Gehani and U. Lindqvist. 2007. VEIL: A System for Certifying Video Provenance. In *Proc. IEEE International Symposium on Multimedia (ISM).*
[33] P. Gilbert, J. Jung, K. Lee, H. Qin, D. Sharkey, A. Sheth, and L. P. Cox. 2011. YouProve: Authenticity and Fidelity in Mobile Sensing. In *Proc. ACM SenSys.*
[34] D. Harwell. 2019. Faked Pelosi videos, slowed to make her appear drunk, spread across social media. https://www.washingtonpost.com/technology/2019/05/23/faked-pelosi-videos-slowed-make-her-appear-drunk-spread-across-social-media/.
[35] H. R. Hasan and K. Salah. 2019. Combating Deepfake Videos Using Blockchain and Smart Contracts. *IEEE Access* 7 (2019), 41596–41606. https://doi.org/10.1109/ACCESS.2019.2905689
[36] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo. 2013. Using Innovative Instructions to Create Trustworthy Software Solutions. (2013).
[37] T. Hunt, Z. Jia, V. Miller, A. Szekely, Y. Hu, C. J. Rossbach, and E. Witchel. 2020. Telekine: Secure Computing with Cloud GPUs. In *Proc. USENIX NSDI.*
[38] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. 2018. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM Transactions on Computer Systems (TOCS)* (2018).
[39] M. Ibrahim, A. Imran, and A. Bianchi. 2021. SafetyNOT: On the usage of the SafetyNet Attestation API in Android. In *Proc. ACM MobiSys.*
[40] P. Korshunov and S. Marcel. 2018. DeepFakes: a New Threat to Face Recognition? Assessment and Detection. (2018). arXiv:1812.08685
[41] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang. 2017. Hacking in Darkness: Return-Oriented Programming Against Secure Enclaves. In *Proc. USENIX Security Symposium.*
[42] M. Lentz. 2020. Assurance and Control over Sensitive Data on Personal Devices. https://https://doi.org/10.13016/nx2k-waen.

[43] Y. Li, M. Chang, and S. Lyu. 2018. Ictu Oculi: Exposing AI Created Fake Videos by Detecting Eye Blinking. In *Proc. IEEE International Workshop on Information Forensics and Security (WIFS).*

[44] Y. Li and S. Lyu. 2019. Exposing Deepfake Videos by Detecting Face Warping Artifacts. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).*

[45] H. Liu, S. Saroiu, A. Wolman, and H. Raj. 2012. Software Abstractions for Trusted Sensors. In *Proc. ACM MobiSys.*

[46] S. McCloskey and M. Albright. 2018. Detecting GAN-generated Imagery using Color Cues. (2018). arXiv:1812.08247

[47] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. 2013. Innovative Instructions and Software Model for Isolated Execution.. In *Proc. Inter. Workshop on Hardware and Architectural Support for Security and Privacy (HASP).*

[48] S. Mervosh. 2019. Distorted Videos of Nancy Pelosi Spread on Facebook and Twitter, Helped by Trump. https://www.nytimes.com/2019/05/24/us/politics/pelosi-doctored-video.html.

[49] S. Mirzamohammadi, Y. Liu, T. A. Huang, A. Amiri Sani, S. Agarwal, and S. E. Kim. 2020. Tabellion: System Support for Secure Legal Contracts. In *Proc. ACM MobiSys.*

[50] A. Naveh and E. Tromer. 2016. PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations. In *Proc. IEEE Symposium on Security and Privacy (S&P).*

[51] H. Park and F. X. Lin. 2022. TinyStack: A Minimal GPU Stack for Client ML. In *Proc. ACM ASPLOS.*

[52] R. Poddar, G. Ananthanarayanan, S. Setty, S. Volos, and R. A. Popa. 2020. Visor: Privacy-Preserving Video Analytics as a Cloud Service. In *Proc. USENIX Security Symposium.*

[53] A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner. 2019. Faceforensics++: Learning to Detect Manipulated Facial Images. In *Proc. IEEE International Conference on Computer Vision.*

[54] S. Saroiu and A. Wolman. 2010. I Am a Sensor, and I Approve This Message. In *Proc. ACM Workshop on Mobile Computing Systems & Applications (HotMobile).*

[55] M. Shih, S. Lee, T. Kim, and M. Peinado. 2017. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs. In *Proc. NDSS Symposium.* https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/t-sgx-eradicating-controlled-channel-attacks-against-enclave-programs/

[56] M. Sorell. 2012. Video Provenance by Motion Vector Analysis: A Feasibility Study. In *Proc. IEEE International Symposium on Communications, Control and Signal Processing (ISCCSP).*

[57] M. H. Stanzione. 2020. 'Wet' Ink Signatures Requirements May Fade After Coronavirus. Bloomberg Law, The United States Law Week.

[58] L. Sweeney. 2002. *k*-ANONYMITY: A MODEL FOR PROTECTING PRIVACY. *Int. Journal on Uncertainty, Fuzziness and Knowledge-based Systems* (2002).

[59] Z. Tang, F. Delattre, P. Bideau, M. D. Corner, and E. Learned-Miller. 2020. C-14: Assured Timestamps for Drone Videos. In *ACM MobiCom.*

[60] R. Toews. 2020. Deepfakes Are Going To Wreak Havoc On Society. We Are Not Prepared. https://www.forbes.com/sites/robtoews/2020/05/25/deepfakes-are-going-to-wreak-havoc-on-society-we-are-not-prepared/.

[61] S. Volos, K. Vaswani, and R. Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *Proc. USENIX OSDI.*

[62] M. Vonau. 2020. Latest SafetyNet improvements threaten to finally kill Magisk Hide. https://www.androidpolice.com/2020/03/11/safetynet-improvements-kill-magisk-hide/.

[63] H. Wang, P. Wang, Y. Ding, M. Sun, Y. Jing, R. Duan, L. Li, Y. Zhang, T. Wei, and Z. Lin. 2019. Towards Memory Safe Enclave Programming with Rust-sgx. In *Proc. ACM CCS.*

[64] T. Winkler, A. Erdélyi, and B. Rinner. 2014. TrustEYE.M4: Protecting the Sensor – not the Camera. In *Proc. IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS).*

[65] T. Winkler and B. Rinner. 2010. TrustCAM: Security and Privacy-Protection for an Embedded Smart Camera Based on Trusted Computing. In *Proc. IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS).*

[66] X. Yang, Y. Li, and S. Lyu. 2019. Exposing Deep Fakes using Inconsistent Head Poses. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).*