UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Restoring Trust in Visual Media Using Provenance

Permalink

https://escholarship.org/uc/item/8d988619

ISBN

9798247985563

Author

Liu, Yuxin

Publication Date

2026-06-09

Copyright Information

This work is made available under the terms of a Creative Commons Attribution-NonCommercial-NoDerivatives License, available at https://creativecommons.org/licenses/by-nc-nd/4.0/

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Restoring Trust in Visual Media Using Provenance

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Yuxin “Myles” Liu

Dissertation Committee:
Professor Ardalan Amiri Sani, Chair
Professor Gene Tsudik
Professor Habiba Farrukh
Dr. Sharad Agarwal

2026

Portion of Chapter 1 © 2022 ACM
Portion of Chapter 1 © 2024 ACM
Portion of Chapter 1 © 2025 USENIX
Portion of Chapter 2 © 2022 ACM
Portion of Chapter 3 © 2024 ACM
Portion of Chapter 4 © 2025 USENIX
Portion of Chapter 5 © 2022 ACM
Portion of Chapter 5 © 2024 ACM
Portion of Chapter 5 © 2025 USENIX
Portion of Appendix A © 2025 USENIX
All other materials © 2026 Yuxin “Myles” Liu

DEDICATION

I dedicate this dissertation to my parents, who love and support me unconditionally, and to my two lovely cats.

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES viii

LIST OF ALGORITHMS ix

ACKNOWLEDGMENTS x

VITA xii

ABSTRACT OF THE DISSERTATION xiv

1 Introduction 1

2 Vronicle: Verifiable Provenance for Videos from Mobile Devices 4
- 2.1 Introduction . . . 4
- 2.2 Background . . . 7
  - 2.2.1 Trusted Execution Environment . . . 7
  - 2.2.2 Video Post-Processing . . . 8
- 2.3 Vronicle Design Overview . . . 9
- 2.4 TEE-based Provenance Info . . . 10
  - 2.4.1 Generating Provenance Info . . . 10
  - 2.4.2 Verifying Provenance Info . . . 12
  - 2.4.3 Supporting Videos . . . 12
  - 2.4.4 Provenance Info for Videos . . . 13
- 2.5 Threat/Adversary Model . . . 14
- 2.6 Post-Processing Units . . . 15
  - 2.6.1 Minimizing Provenance Info . . . 17
  - 2.6.2 Flow of Provenance Across Units . . . 18
- 2.7 Implementation . . . 19
  - 2.7.1 Video Post-Processor . . . 19
  - 2.7.2 Camera Device . . . 21
  - 2.7.3 TrustZone-based Camera . . . 22
  - 2.7.4 Consumer . . . 22
- 2.8 Security Analysis . . . 22
  - 2.8.1 Video Attacks . . . 23

2.8.2 Platform Attacks . . . 24
2.9 Performance Evaluation . . . 26
2.9.1 Setup . . . 26
2.9.2 Evaluation Results . . . 27
2.10 Discussions & Future Work . . . 31
2.10.1 Performance Optimizations . . . 31
2.10.2 Third-Party Verifiers . . . 32
2.10.3 Camera Device Privacy . . . 32
2.11 Conclusions . . . 33

3 ProvCam: A Camera Module with Self-Contained TCB for Producing Verifiable Videos 34
3.1 Introduction . . . 34
3.2 Threat Model & Motivation . . . 37
3.2.1 Threat Model . . . 37
3.2.2 Current Techniques . . . 38
3.3 Background . . . 40
3.4 Design . . . 41
3.5 Hardware Design Challenges . . . 44
3.5.1 The Use of Untrusted Main Memory . . . 44
3.5.2 Asynchronous Hashing . . . 46
3.5.3 Video Frame-Based Hashing . . . 47
3.6 Software Stack Design Challenges . . . 48
3.6.1 Replay-Based Bare-Bones Driver . . . 48
3.6.2 Double Replay for a Backward-Compatible Software Stack . . . 52
3.6.3 Trusted Capture Session . . . 52
3.7 Prototype . . . 54
3.8 Security Analysis . . . 54
3.9 Prototype Evaluation . . . 55
3.9.1 Hardware Cost . . . 56
3.9.2 Software TCB . . . 59
3.9.3 Performance . . . 59
3.9.4 Energy Consumption . . . 60
3.9.5 Programming Effort . . . 61
3.10 Discussions & Future Work . . . 61
3.10.1 Trusted and Normal Videos . . . 61
3.10.2 Privacy and Identity . . . 62
3.10.3 Key Revocation . . . 62
3.11 Conclusions . . . 63

4 Scoop: Mitigation of Recapture Attacks on Provenance-Based Media Authentication 64
4.1 Introduction . . . 64
4.2 Recapture Attack . . . 68
4.2.1 Categorizing Recapture Attacks . . . 69

4.2.2 Terminology . . . 70
4.2.3 User Study . . . 71
4.3 Threat Model . . . 73
4.4 Overview . . . 74
4.5 Depth Estimation Preliminaries . . . 77
4.5.1 ToF Depth Sensor . . . 77
4.5.2 Learning-based Monocular Depth Estimation . . . 78
4.6 Design . . . 78
4.7 Depth Maps Comparison . . . 80
4.7.1 Regional Segmentation . . . 80
4.7.2 Regional Comparison . . . 82
4.8 Dataset . . . 83
4.8.1 Collection of Our Dataset . . . 86
4.9 Prototype & Evaluation . . . 86
4.9.1 Effectiveness . . . 88
4.9.2 Overheads . . . 91
4.10 Additional Considerations . . . 93
4.10.1 3D Display-Based Attacks . . . 93
4.10.2 Binocular Vision . . . 94
4.10.3 Forward Looking Infrared (FLIR) . . . 94
4.10.4 Videos . . . 94
4.10.5 Post-Processing . . . 95
4.11 Conclusions . . . 96

5 Related Work **97**
5.1 Secure Media Attestation . . . 97
5.1.1 Secure Video Provenance . . . 97
5.1.2 Secure Camera and Sensors . . . 98
5.1.3 Secure Video Processing using GPU TEEs . . . 99
5.1.4 Motion-based Video Timestamps Attestation . . . 99
5.2 Securing Sensing and I/O . . . 99
5.2.1 Replay-based I/O . . . 99
5.3 Media Content Authentication . . . 100
5.3.1 Display Screen Recapture Detection . . . 100
5.3.2 Face anti-spoofing . . . 100
5.3.3 Object anti-spoofing . . . 101

Bibliography **102**

A Scoop: Mitigation of Recapture Attacks on Provenance-Based Media Authentication **115**

LIST OF FIGURES

2.1 Overview of Vronicle. 9
2.2 High-level view of the post-processor. 13
2.3 Fixed-function post-processing units for video post-processing. 16
2.4 Flow of provenance info between fixed-function post-processing units. . . . 18
2.5 Post-processing execution time with different filters in the ARM TrustZone-based post-processor (Left) and the Intel SGX-based post-processor (Right). 26
2.6 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). 28
2.7 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). 29
2.8 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. 30

3.1 Comparison between two current approaches and ProvCam's TCBs. Detailed flow inside ProvCam camera module is shown in Figure 3.3. 41
3.2 3D-stacked secure packaging of ProvCam's camera module. 42
3.3 Comparisons between a typical video capture pipeline and ProvCam's secure pipelines. 43
3.4 Comparison between how ISP and encoder write/read raw frames and how the secure reorder buffer addresses it. 45
3.5 How frame-based hashing works in ProvCam. 49
3.6 An example of mismatched I/O read of a FIFO's status register. 51
3.7 Performance evaluations of ProvCam. The histograms show the average. The error bars show the standard deviation. 58

4.1 Demonstration of effectiveness of recapture attacks. One of these photos is real (i.e., captured by a camera in a real environment) and one is a recapture attack (i.e., captured by a camera pointed at a screen showing a modified photo). Even though we have used a mid-range TV, detecting the attack is very challenging (especially if the user is not suspecting an attack). Section A.1 mentions which one of these photos is real. 65
4.2 An example of a recapture attack on a provenance-based secure camera. . . . 67

4.3 Illustration of full-recapture (a) and partial-recapture (b) attacks. 70
4.4 These two example photos, even if display mediums (i.e., a monitor and a TV) are presented in them, do not constitute an attack since the presence of the medium can be recognized by the viewer. 70
4.5 Terminology of recaptured visual content. Note that Recapture is the largest set, with Misleading Recapture as a subset of it, followed by Malicious Recapture as a subset of Misleading Recapture. 71
4.6 Illustration of a learning-based depth estimation model got tricked by a photo displayed on a TV, while a ToF sensor captured depth map correctly represents the depth of the real scene. 76
4.7 Scoop viewer's workflow. 77
4.8 Scoop viewer's techniques. In each technique, we try to match the region extracted from the ground truth point cloud with its corresponding (pixel-level correspondence) region in the learning-based point cloud. Only if it cannot be matched, we move to the next technique, until we run out of techniques, where we would flag the region as suspicious. 80
4.9 Example from our dataset showing different display mediums used. 83
4.10 Demonstration of Scoop viewer prototype's workflow with a sample photo containing a life size cardboard cutout of a celebrity. 86
4.11 Results of Scoop's prototype under different display mediums. 90
4.12 Results of Scoop's prototype under different benign content categories. . . . 90
4.13 Overall effectiveness of Scoop's prototype. 90
4.14 Both performance and storage overheads of Scoop's prototype. 91

LIST OF TABLES

2.1 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. 13
2.2 *Comparison with related work.* 31

3.1 Extra hardware cost in both hardware prototypes. 57
3.2 Comparisons of the amount of transistors needed for different resolutions between secure buffer and reorder buffer for untrusted memory. 57
3.3 Comparison with related work. Note that TEE + others means the system TCB includes not only TEE, but also other (potentially less trustworthy) components 61

LIST OF ALGORITHMS

1 Scoop's depth map comparison algorithm 79

ACKNOWLEDGMENTS

I would like to express my profound gratitude to my advisor, Professor Ardalan Amiri Sani. He introduced me to the world of research and has been a steadfast guide through the challenging yet exhilarating pathway of both my undergraduate and doctoral journeys. From him, I learned not only the mechanics of conducting research but also the mindset required to excel. I deeply admire his unwavering dedication to integrity and high-quality scholarship. Without his mentorship, I could never have imagined devoting myself so fully to this field.

I am equally grateful for the invaluable guidance of Professor Gene Tsudik and Dr. Sharad Agarwal throughout my Ph.D. studies. Their tremendous support and wisdom made navigating the intersection of academic and industrial research a much smoother and more rewarding experience.

I extend my sincere thanks to my entire dissertation committee—Professor Ardalan Amiri Sani, Professor Gene Tsudik, Professor Habiba Farrukh, and Dr. Sharad Agarwal, for their time, insightful feedback, and encouragement.

To all the collaborators I have had the privilege of working with: thank you. It has been a true honor to build and explore alongside you, and your contributions have made this research journey unforgettable.

I would also like to thank my labmates and friends at UCI. Your camaraderie has made my time working in the lab truly delightful.

Similarly, I wish to express my gratitude to the colleagues I worked alongside during my time at Bosch Research; you made my industrial research experience both highly productive and deeply enjoyable.

To my wonderful friends outside of UCI, thank you for providing balance and making my life beyond research so vibrant and relaxing.

Most importantly, I am deeply grateful to my parents and my two lovely cats for their unconditional love, patience, and support throughout this endeavor.

Finally, my research was graciously supported by NSA Awards (#H98230-20-1-0345 and #H98230-22-1-0308), NSF Awards (#1763172, #1953932, and #2247880) and UCI ICS Exploration Research Awards.

The text of Chapters 1, 2, and 5 of this dissertation is an adaptation of the material as it appears in "Vronicle: verifiable provenance for videos from mobile devices" [123] In Proceedings of the ACM MobiSys 2022, used with permission from ACM. The coauthors listed in this publication are Yuxin (Myles) Liu, Yoshimichi Nakatsuka, Ardalan Amiri Sani, Sharad Agarwal, Gene Tsudik. Ardalan Amiri Sani directed and supervised research which forms the basis for the dissertation.

The text of Chapters 1, 3, and 5 of this dissertation is an adaptation of the material as it appears in "ProvCam: A Camera Module with Self-Contained TCB for Producing Verifiable Videos" [124] In Proceedings of the ACM MobiCom 2024, used with permission from ACM. The coauthors listed in this publication are Yuxin (Myles) Liu, Zhihao Yao, Mingyi Chen, Ardalan Amiri Sani, Sharad Agarwal, Gene Tsudik. Ardalan Amiri Sani directed and supervised research which forms the basis for the dissertation.

The text of Chapters 1, 4, 5, and Appendix A of this dissertation is an adaptation of the material as it appears in "Scoop: Mitigation of Recapture Attacks on Provenance-Based Media Authentication" [125] In Proceedings of the USENIX Security 2025, used with permission from USENIX. The coauthors listed in this publication are Yuxin (Myles) Liu, Habiba Farrukh, Ardalan Amiri Sani, Sharad Agarwal, Gene Tsudik. Ardalan Amiri Sani directed and supervised research which forms the basis for the dissertation.

VITA

Yuxin "Myles" Liu

EDUCATION

Doctor of Philosophy in Computer Science, **2026**
University of California, Irvine *Irvine, California*

Master of Science in Computer Science, **2026**
University of California, Irvine *Irvine, California*

Bachelor of Science with *honor* in Computer Science, **2020**
University of California, Irvine *Irvine, California*

RESEARCH EXPERIENCE

Graduate Research Assistant, **2020-2026**
University of California, Irvine *Irvine, California*

Automotive Security Intern, **2025-2026**
Bosch Research *Pittsburgh, Pennsylvania*

Security Research Intern, **2024**
Bosch Research *Sunnyvale, California*

Undergraduate Research Assistant, **2018-2020**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant, **2021-2023**
University of California, Irvine *Irvine, California*

Lab Tutor, **2017**
University of California, Irvine *Irvine, California*

REFEREED CONFERENCE PUBLICATIONS

Scoop: Mitigation of Recapture Attacks on Provenance-Based Media Authentication, **August, 2025**
USENIX Security

ProvCam: A Camera Module with Self-Contained TCB for Producing Verifiable Videos, **November, 2024**
ACM MobiCom

Vronicle: Verifiable Provenance for Videos from Mobile Devices, **June, 2022**
ACM MobiSys

Tabellion: Secure Legal Contracts on Mobile Devices, **June, 2020**
ACM MobiSys

ABSTRACT OF THE DISSERTATION

Restoring Trust in Visual Media Using Provenance

By

Yuxin “Myles” Liu

Doctor of Philosophy in Computer Science

University of California, Irvine, 2026

Professor Ardalan Amiri Sani, Chair

Visual media plays a central role in journalism, public safety, legal evidence, and everyday communication. Yet advances in deepfakes, generative AI, and media editing have eroded trust in what cameras record. This dissertation argues that restoring trust in visual content requires moving beyond post-hoc detection toward preventative, system-level authenticity guarantees that span capture, processing, and consumption. It presents three systems that address complementary weaknesses in modern visual media pipelines.

First, Vronicle provides fine-grained provenance for videos captured on mobile devices. By combining trusted execution environments (TEEs) with cryptographic provenance and fixed-function post-processing units, it enables a consumer to verify both the source camera and the exact sequence of post-processing filters applied to a video. Second, ProvCam strengthens the root of trust (RoT) at capture time. It introduces a camera module with a small self-contained trusted computing base (TCB) that protects video capture, image processing, encoding, and signing against software and hardware tampering, thereby reducing reliance on large, attack-prone device stacks. Third, Scoop exposes a fundamental blind spot in provenance-based media authentication: recapture attacks, where fake content is displayed on a screen or printed surface and then captured by a secure camera. Scoop detects misleading recaptures by comparing depth sensed by time-of-flight hardware with perceived depth inferred from RGB images.

Together, these systems show that trustworthy visual media requires both strong provenance guarantees and defenses against physical-world attacks that exploit the camera itself. The dissertation demonstrates that these goals are practical on commodity mobile devices and prototype hardware, and that they can substantially improve the authenticity, integrity, and credibility of photos and videos in a world increasingly shaped by synthetic media.

Chapter 1

Introduction

Visual media has become one of the primary ways people understand the world. Cameras embedded in smartphones, tablets, wearable devices, and other everyday systems allow nearly anyone to capture and share photos and videos instantly. This scale is enormous: billions of photos and hundreds of thousands of hours of video are shared online every day [160]. At the same time, visual content increasingly influences consequential decisions in journalism, legal proceedings, public safety, and civic discourse. A substantial fraction of the public now gets news from YouTube and social media platforms [154, 73, 72]. As society relies more heavily on visual evidence, the question of whether a photo or video can be trusted becomes more important.

The question regarding trust in visual media has become much harder to answer. Advances in photo and video editing, deepfakes, and generative AI now make it possible to create or alter highly realistic visual content at low cost and at scale [170, 46, 53]. These techniques can fabricate events that never happened, subtly manipulate genuine footage, or reframe real scenes in misleading ways. Public examples, such as manipulated videos of political figures, have shown how quickly such content can spread and how difficult it is for viewers to assess credibility once a clip is circulating online [29, 1, 148]. The result is not only the direct harm caused by fake media, but also a broader erosion of trust in authentic visual evidence.

Broadly speaking, two technical directions have emerged for addressing this problem. The first is post-hoc detection: analyzing content for artifacts, inconsistencies, or other signatures of manipulation [119, 143]. While this approach is attractive because it can be applied after content is produced, it is fundamentally reactive and faces an ongoing arms race against increasingly capable generative models. The second direction is prevention through provenance: attaching cryptographically verifiable information to visual content so that a consumer can authenticate its origin and understand how it was processed [18, 25, 123, 124, 20]. Provenance-based approaches are especially promising because they aim to provide hard-to-fabricate evidence about the source and history of media, rather than probabilistic judgments based on content alone.

This dissertation takes a preventative stance and studies how to restore trust in visual media through careful system design to reimagine existing visual media pipelines. Its central thesis is that trustworthy visual content requires end-to-end authenticity guarantees across the entire media pipeline: the point of capture, the stages of post-processing, and the eventual consumption context. Securing only one stage is insufficient. A provenance system is weakened if the camera RoT is too large and attack-prone. A secure camera is not enough if subsequent post-processing cannot be authenticated. And even strong provenance can be undermined if an adversary can mount convincing physical-world attacks, such as displaying fake content on a screen and recapturing it with a secure camera. To address these gaps, this dissertation develops three complementary systems: Vronicle, ProvCam, and Scoop.

Vronicle addresses the problem of trustworthy post-capture processing for mobile videos [123]. It provides fine-grained provenance that allows a consumer to verify both the originating camera and the exact sequence of post-processing filters applied to a video. Its design combines TEEs with cryptographic provenance and fixed-function post-processing units. This yields a practical balance between security, verification granularity, and performance, enabling provenance for realistic mobile video workflows instead of only narrowly scoped or prohibitively expensive settings.

ProvCam addresses the camera RoT [124]. Existing approaches often rely on large TCB in commodity devices, leaving too much camera hardware and software in the attack surface. ProvCam instead introduces a self-contained camera module with a tiny TCB that captures, processes, encodes, hashes, and signs video within the module itself. Its goal is to bootstrap authenticity at the camera sensor and survive the complex pipeline between sensing and encoded output, while remaining practical enough to integrate with modern camera-bearing devices.

Scoop addresses a blind spot that remains even when provenance is strong [125]. In recapture attacks, fake or misleading content is shown on a display medium or printed surface and then captured by a provenance-asserting camera, causing the resulting media to inherit the camera's credibility. Scoop shows that provenance alone does not solve this problem. It then introduces a defense based on comparing depth measured by time-of-flight hardware with depth perceived from RGB content by a learning-based model. The key insight is that misleading recaptures can create a mismatch between physical depth and perceived scene depth, allowing such attacks to be flagged for the viewer.

Taken together, these three works present a broader view of visual trust. They show that authenticity is not a single mechanism or one-time check, but a pipeline property that must be established, preserved, and interpreted. The first contribution of this dissertation is a practical design for fine-grained video provenance during post-processing. The second is a hardware-backed reduction of the TCB at capture time. The third is an analysis of a fundamental attack on provenance systems and a concrete mitigation for that attack. Collectively, they move the field from isolated solutions toward a more complete foundation for trustworthy visual media.

Chapter 2

Vronicle: Verifiable Provenance for Videos from Mobile Devices

2.1 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 [130], (*v*) e-voting (e.g., shareholder or political) platforms using videos from smartphones to identify voters [21], 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 [152, 24].

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 [29]. 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 [129, 98]. Yet another recent example is a fake video used by a parent to disadvantage cheerleading rivals of her daughter [88]. 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 [119, 120, 127, 177, 113, 143] 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 [162, 31]. 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 Vronicle, 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 [131] 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 [83] 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, 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 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.

2.2 Background

2.2.1 Trusted Execution Environment

A 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

[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/

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)* [56, 100, 128] and ARM TrustZone [7, 6]. 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 world OS can implement remote attestation, as in Kinibi OS in Samsung devices [59].

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 [11], which helps developers gain trust in users' devices and it is used in security-critical applications such as finance [107]. SafetyNet provides an attestation API [19], 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 [165].

2.2.2 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,

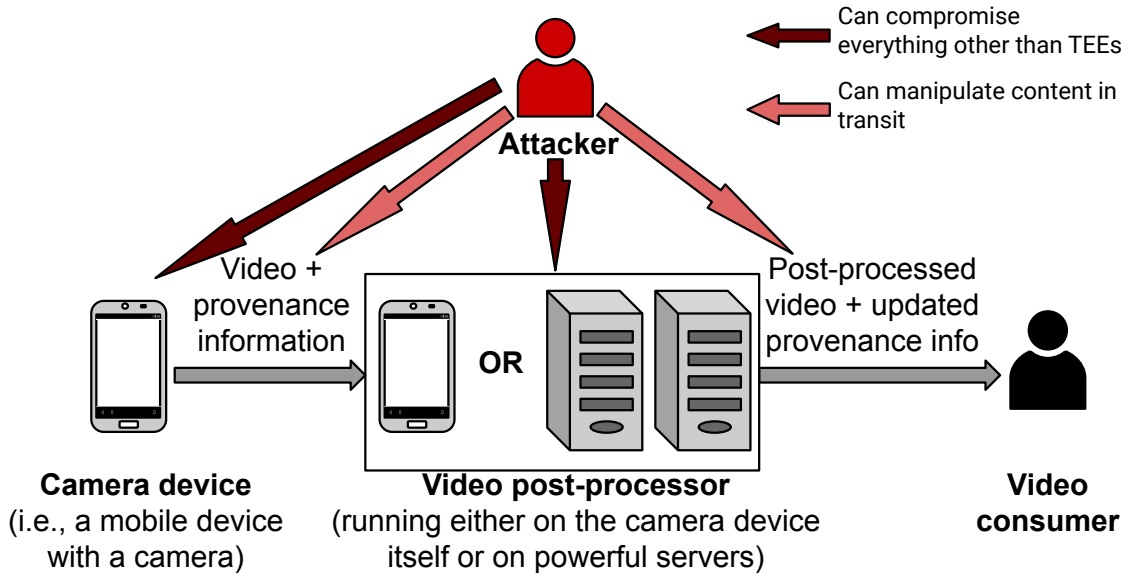


Figure 2.1: Overview of Vronicle.

deflicking, brightness adjustment, saturation adjustment, contrast adjustment, sharpening, whitening, and lens flare [54]. 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.

2.3 Vronicle Design Overview

Figure 2.1 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.

2.4 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.

2.4.1 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: $\texttt{PI} = \{\texttt{AR}_{\texttt{TEE_cam}}, \texttt{F_metadata}, \texttt{Sig}_{\texttt{cam}}\}$. The first element of PI ($\texttt{AR}_{\texttt{TEE_cam}}$) 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 ($\texttt{Sig}_{\texttt{cam}}$) 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: $\texttt{PI}' = \{\texttt{PI}, \texttt{AR}_{\texttt{TEE_pp}}, \texttt{filter_metadata}, \texttt{Sig}_{\texttt{pp}}\}$.

The first element (PI) is included here without modifications. The second element ($\texttt{AR}_{\texttt{TEE_pp}}$) 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 ($\texttt{Sig}_{\texttt{pp}}$) 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.

2.4.2 Verifying Provenance Info

The consumer receives the following information: $\{\mathtt{F}', \mathtt{PI}'\}$. First, the consumer verifies the attestation reports in $\mathtt{PI}'$ (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 $\mathtt{F}$. We will address this issue in §2.6.1. 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.

2.4.3 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 2.2 shows this design.

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

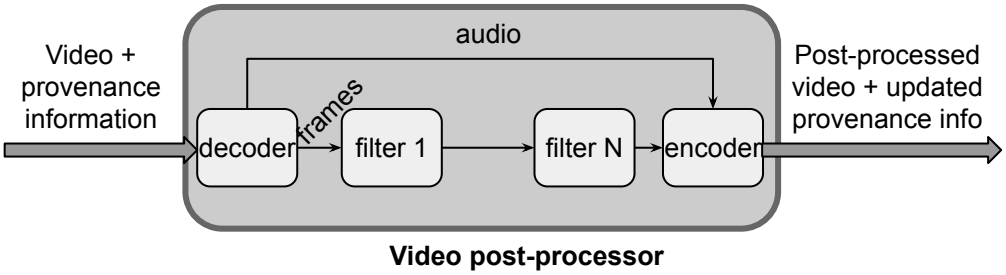


Figure 2.2: High-level view of the post-processor.

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 2.1: 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.

Note that it is possible to perform post-processing on the audio as well, although we do not currently support that in our prototype.

2.4.4 Provenance Info for Videos

In §2.4.1, we discussed the components of the provenance info for a single frame. The provenance info for a video includes additional information, shown in Table 2.1. 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 (§2.8.1).

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 2.1 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 2.1 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 (§2.6).

2.5 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 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 work, 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.

2.6 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

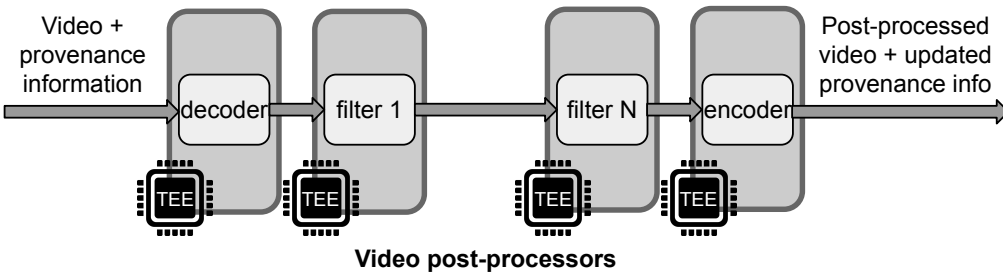


Figure 2.3: Fixed-function post-processing units for video post-processing.

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 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 2.3 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 work. 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.

2.6.1 Minimizing Provenance Info

§2.3 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).

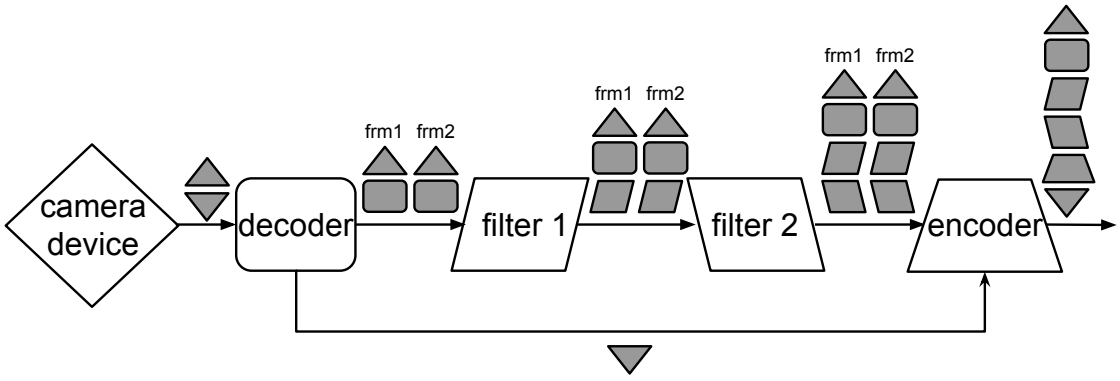


Figure 2.4: Flow of provenance info between fixed-function post-processing units.

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.

2.6.2 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 design the flow of provenance info between units, as discussed next and illustrated in Figure 2.4 (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.

2.7 Implementation

2.7.1 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 §2.2.1, 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.

2.7.2 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.

2.7.3 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 [116]. Our own work, Tabellion [130], 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 [20].

2.7.4 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 [4]).

2.8 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.

2.8.1 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 2.1: 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 2.1: 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 2.1: 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 2.1: 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 2.1: Segment Info), this attack is easily detected.

Frame Rate Manipulation. This is the attack used to create a deepfake video of the

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

US Speaker of the House Nancy Pelosi [129], in which the frame rate of the video segment (where she was talking) was reduced. Vronicle detects this attack because the frame rate (Table 2.1: 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 2.1: 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 2.1: 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 §2.4.4, it is possible use the depth map info to defeat this attack. That is, the camera device can record the depth map (Table 2.1: 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 [58]), or using software-only solutions [86].

2.8.2 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 [115, 63], 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 [78, 149, 105], 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 [75]) or by using an SDK written in a secure programming language (e.g., Rust [167]).

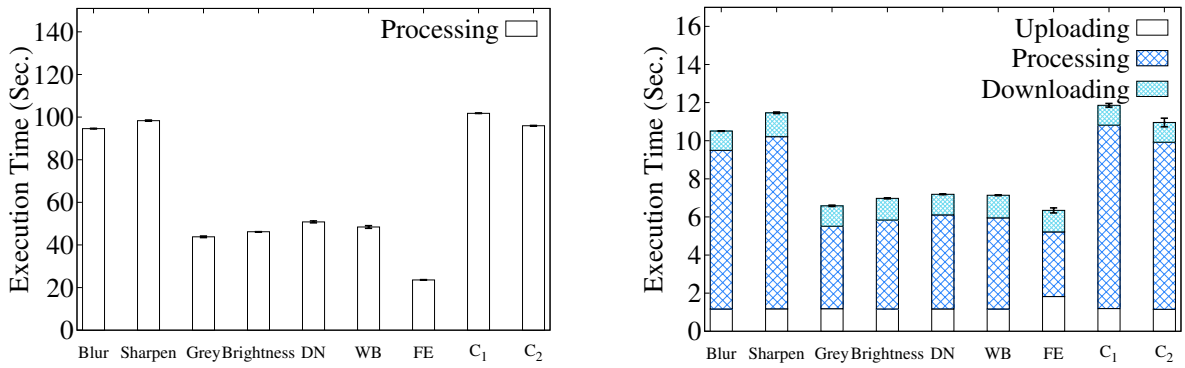


Figure 2.5: Post-processing execution time with different filters in the ARM TrustZone-based post-processor (Left) and the Intel SGX-based post-processor (Right).

2.9 Performance Evaluation

2.9.1 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 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.

2.9.2 Evaluation Results

Post-processing execution time. Figure 2.5 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

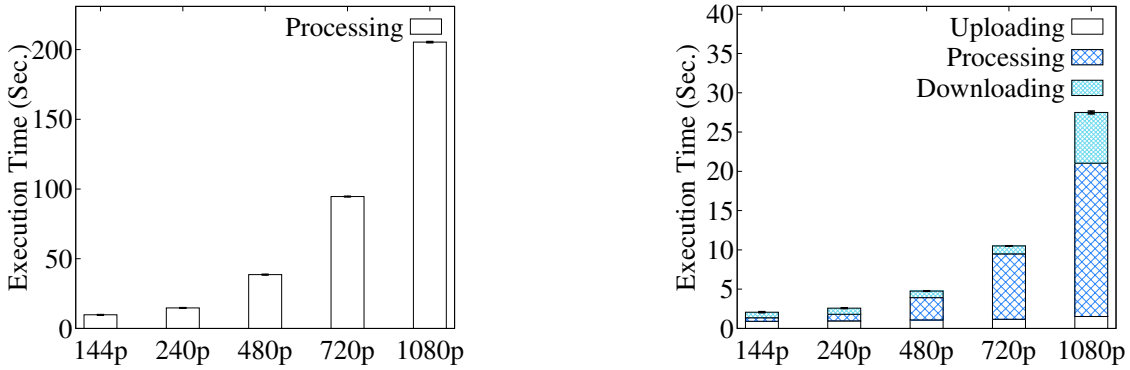


Figure 2.6: Post-processing execution time for videos with different resolutions in the ARM TrustZone-based post-processor (Left) and the Intel SGX-based post-processor (Right).

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 §5.1, 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 2.6 and Figure 2.7 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

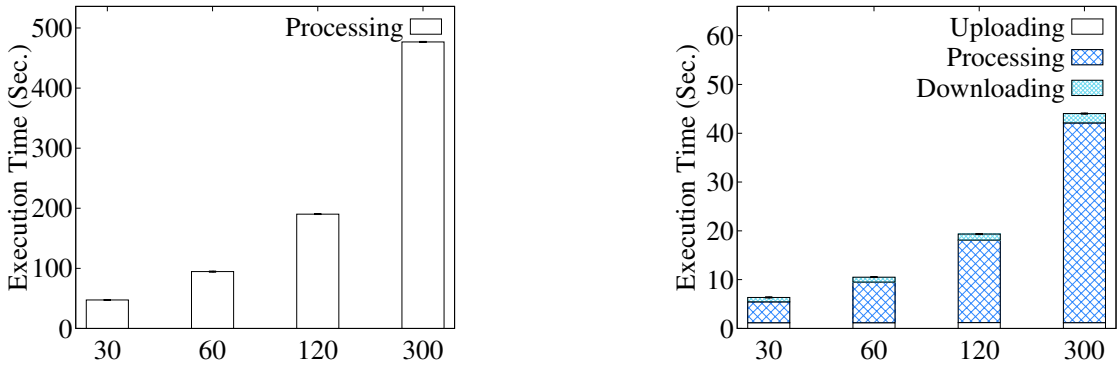


Figure 2.7: Post-processing execution time for videos with a different number of frames in the ARM TrustZone-based post-processor (Left) and the Intel SGX-based post-processor (Right).

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 2.8. 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.,

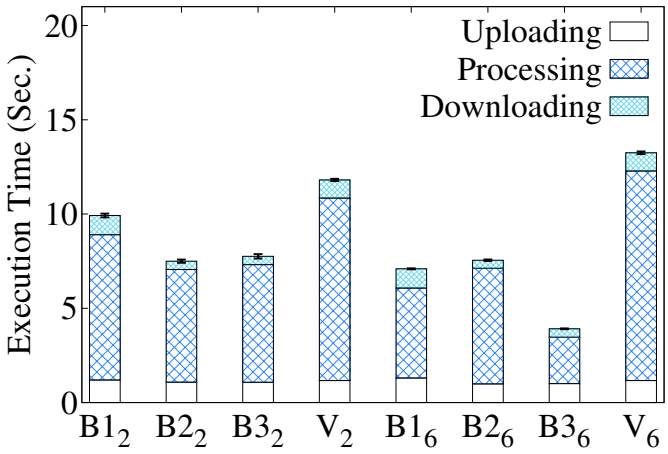


Figure 2.8: Post-processing execution time with different design decisions. B1, B2, B3, and V refer to Basline1, Baseline2, Baseline3, and Vronicle. The last number in the x-axis labels shows the configuration. 2 is the 2-filter configuration and 6 is the 6-filter configuration.

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 2.8.

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,

	Granularity of Provenance	Performance (sec)	Videos?	Method
Alethia [22]	Coarse-grained	No measurements	Yes	Digital Signature
PhotoProof [131]	Fine-grained	673.5 (128x128)	No	Zero Knowledge Proof
YouProve [91]	Coarse-grained	28.0 (JPEG, 1296x972)	No	Fidelity Analysis
FrameProv [55]	Fine-grained	No prototype	Yes	Digital Signature
AMP [83]	Coarse-grained	0.08 ∼0.24	Yes	Fragile Watermarking
Vronicle	Fine-grained	4 (30 frames, 1280x720) ∼40 (300 frames)	Yes	TEE

Table 2.2: *Comparison with related work.*

we believe Vronicle's performance is good and usable for non-real-time use cases.

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) [14] 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* [10]. 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.

2.10 Discussions & Future Work

2.10.1 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 [164, 135], which are not currently available for SGX enclaves.

2.10.2 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.

2.10.3 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 §2.7.2.

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 [157], can be integrated in our solution.

2.11 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 work 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.

Chapter 3

ProvCam: A Camera Module with Self-Contained TCB for Producing Verifiable Videos

3.1 Introduction

The importance of videos in modern society has grown tremendously. Although video authenticity is very important, it is not guaranteed today. Consumers of videos have no reliable means of verifying how the video was captured and what modifications (if any) were made to it. Worse yet, given impressive advances in deepfake [170] and generative AI technologies, consumers cannot even determine whether a video is synthetic or was captured by a real camera.

The prevention-based mitigation approach focuses on video provenance based on cryptographic methods. [123, 131, 83, 20]. It involves authenticating to the consumer the source of video and all filters applied to it during post-processing. For example, Vronicle employs provenance information to attest both the source camera and post-processing filters [123], while TruePic [20] utilizes a TEE and specialized hardware for frame provenance information.

This line of work is more promising than detection (which yields probabilistic outcomes) since it provides strong assurance to video consumers. However, based on our review of state-of-the-art prevention techniques, security of the camera device is the weakest link. In other words, an attacker can compromise the camera device in order to fabricate provenance information for a fake video. This is especially problematic since the owner of the camera device is the adversary, i.e., the party trying to generate fake videos.

In this chapter, we tackle this problem by constructing ProvCam– a camera module for generating verifiable videos. Compared with the large TCB (both hardware and software) of TEE-based solutions (which we discuss in detail in Section 3.2.2), the key contribution of ProvCam is its *tiny TCB* that contains only *the camera module*. It remains secure even against a powerful adversary that owns the device and can physically attack hardware buses! Moreover, because ProvCam does not depend on any TEE or any other CPU feature, ProvCam can be easily adopted by commercial mobile and camera devices.

ProvCam provides strong isolation when recording, and generating provenance of, a video. Specifically, the module fully isolates itself from untrusted device components to prevent compromise until the video is fully recorded. It captures video frames, processes them, encodes the final video, and generates a provenance report containing a signed hash of the entire video – all within the camera module itself. It is worth noting that ProvCam focuses on producing videos in a trustworthy manner; further post-processing of videos (i.e., applying filters) could be handled securely by other systems such as Vronicle [123].

We addressed six tough technical challenges encountered in ProvCam design: some in the design of the module hardware and others – in its corresponding software stack:

1. Raw frames of videos can be fairly large, requiring a substantial amount of memory to buffer them in the module. To avoid this cost, we find a way to instead securely use untrusted memory.

2. Camera hardware data transactions tend to be bursty and unpredictable, complicating hardware hashers, which require a fixed input stream throughput. To address this, we

use an asynchronous hashing technique.

3. To verify the hash of the video in the presence of incomplete frames, we design a frame-based hashing technique.

4. To achieve strong isolation, we introduce a replay-based bare-bones driver, allowing us to run the complete, yet heavily shrunken, camera stack inside camera module.

5. To be compatible with the existing software stack in the untrusted OS, we add a replay layer to the OS to mimic camera hardware.

6. Finally, for backward compatibility with normal camera usage, we introduce Trusted Capture Session (TCS) – a software abstraction in conjunction with the ProvCam hardware module, that allows the camera module to operate in either secure or normal mode.

To demonstrate its viability and practicality, we prototyped ProvCam (both hardware and software) on a Xilinx UltraScale+ MPSoC FPGA board (ZCU106). [1] The hardware part of the prototype encompasses: IMX274 camera sensor, ISP, video encoder, crypto hardware components, and MicroBlaze microcontroller. The software part includes trusted components running on the aforementioned microcontroller and untrusted ones running on a commodity OS (PetaLinux) on a separate ARM Cortext A53 CPU. The ProvCam prototype can record high-definition (720p) high-framerate (60fps) videos with negligible performance overhead (latency and throughput), moderate hardware cost ($\approx$ 53.8M transistors), and minor energy consumption overhead ($<$ 1Watt).

Overall, we believe that this work makes the following contributions:

- Design of ProvCam– a novel camera module with a small self-contained TCB that generates encoded high-definition videos with cryptographic proofs of provenance.
- Successful solutions to six difficult design challenges (see above).

[1] We open source both hardware and software of the prototype for the benefit of users and researchers at: https://github.com/trusslab/provcam

- Fully functional ProvCam prototype (both hardware and software) on a Xilinx FPGA board.
- Thorough evaluation of ProvCam prototype, including its: hardware cost, performance overhead, and energy consumption.

3.2 Threat Model & Motivation

Video provenance is a popular and promising approach for mitigating fake videos [123, 131, 83, 20]. The goal is to generate a cryptographic proof that can be authenticated by the consumer of the video. This proof is tied to the camera that captured the video and the set of post-processing modifications/filters applied to the video. The weakest security link in such provenance systems is *the camera device itself*. While post-processing can be protected, e.g., using TEEs [123] or zero-knowledge proofs [131], protecting the camera against attacks is very challenging. This is especially concerning since it is in physical possession of the prospective attacker (its owner/operator), who may try to compromise the camera in order to generate verifiable provenance information for a fake video. Below, we discuss some current techniques for protecting the camera device and highlight their limitations. Before that, we summarize the threat model.

3.2.1 Threat Model

The ProvCam threat model involves two entities that interact with a video: producer (camera owner/user/operator) and consumer (video viewer). We assume the former to be the adversary who might try to alter or fabricate videos in order to deceive the latter who is not malicious. The adversary can perform the following attacks (assuming a TrustZone-based [7, 6] camera device):

1. Create arbitrary fake videos.
2. Modify videos generated by a camera device, e.g., insert/drop/crop frames, and apply

arbitrary filters to them.

3. Modify provenance info of videos.
4. Compromise the TrustZone normal world software stack of the camera device, including OS and hypervisor.
5. Compromise the TrustZone secure world software stack of the camera device, including the secure world OS.
6. Replace the camera sensor with a malicious gadget for generating fake video data.
7. Perform active bus tapping physical attacks in order to modify data transmitted on buses (e.g., from the camera to the SoC or between the SoC and main memory) [80, 77].

Although the assumed adversary is quite powerful, we believe that it is realistic given that similar attacks were successful in the past [74, 27] including active bus tapping attacks [111, 101, 114, 103, 66]. Given that fake videos could be used in high impact scenarios such as political and legal contexts, we believe that our adversary is motivated to conduct such difficult attacks. Therefore, we believe it is essential to assume such a powerful adversary and protect against it.

Out-of-Scope Attacks: We assume that the adversary cannot perform chip attacks [77], which are the most sophisticated forms of physical attacks used to access the internals of a secure package, e.g., of the SoC or the camera module discussed in §3.4. We also do not consider attacks on cryptographic primitives, i.e., hashing and signing algorithms.

We assume that the camera module manufacturer is trusted and assume that it is capable of securely embedding a cryptographic key pair in the camera module and recording the I/O commands correctly. Finally, ensuring availability of the camera device and recorded video as well as video confidentiality is out of scope as well.

3.2.2 Current Techniques

We now overview some current techniques for generating secure provenance information.

Approach 1: Generate provenance information in the camera app. Techniques such as AMP [83] and Alethia [22] use an application (e.g., a mobile app) to record the video and then generate cryptographic provenance information in the app. This approach entails a very large TCB. Many hardware and software components have access to the video before it is signed and can alter, or even generate, it. We therefore conclude that this does not provide strong assurance.

Approach 2: Generate provenance information in TrustZone. In order to reduce the TCB size, TrustZone (the secure world) on the camera device can be used instead to securely generate provenance information in the camera device [123, 20]. TrustZone and other TEEs have gained popularity in recent years, particularly in smartphones. This is because they reduce the TCB size by making the OS and hypervisor untrusted.

Although this approach is a significant improvement over the previous one and represents the state-of-the-art, it still does not protect against some important attacks (5-7 in our threat model – §3.2.1). Obviously, the adversary capable of attack 5 (compromising TrustZone secure world) can defeat this method. TrustZone secure world has a large software TCB including an OS, drivers (including drivers for the ISP, encoder, and the camera sensor), and apps, and it has been compromised in the past [74, 81, 141, 132]. Attack 6 can defeat this solution as well. Here, the adversary could disconnect the original camera sensor from the device and instead connect its own malicious gadget in order to generate fake data and pass them to the SoC. Defeating this attack requires the SoC to continuously authenticate the camera sensor. To the best of our knowledge, some Apple devices perform some form of camera sensor authentication [57], but not other mobile devices. An adversary capable of attack 7 (active bus tapping attack) can defeat this solution as well. This is because current TEEs (including TrustZone) can not contain the entirety of the camera pipeline within the SoC chip. The video data is generated by the camera sensor, which is outside the SoC. The data are transferred over a peripheral bus to the SoC. Moreover, during processing by ISP and encoder, the data is transferred from the SoC to the main memory chip. The

adversary can therefore modify the data on either of these buses, although tapping on the low-frequency peripheral bus is easier than the memory bus. Defeating this attack requires the use of memory encryption engines with support for data integrity and replay protection for both the camera sensor and the SoC, which, to the best of our knowledge, are not currently available.

In addition to these security weaknesses, Approach 2 has another practical shortcoming: TEEs are not widely available on SoCs of all types of portable camera devices, especially, non-smartphones devices, e.g., car dashcams, and law enforcement bodycams.

3.3 Background

We now briefly describe key components of the pipeline in a typical digital camera: sensor, ISP, and encoder. In most modern mobile devices, ISP and encoder are integrated into the SoC. Given familiarity with these components, this section can be skipped.

Digital camera sensor. The prevalent type of camera sensor in modern mobile and embedded devices is the Complementary Metal-Oxide-Semiconductor (CMOS) image sensor [109]. The popularity of CMOS sensors is due to their small hardware footprint. The sensor's photodiodes (pixels) respond to the intensity (brightness) of light rather than the wavelength (color). To make color imaging possible, an array of tiny color filters (Color Filter Array – CFA) overlays each pixel. The most widely adopted CFA is the Bayer filter [133]. Each filter within the array only permits a specific color (for Bayer filters, it is red, green, or blue) to pass through to the underlying pixels [35]. Individual pixels only capture one color, and the missing color information is then estimated through demosaicing, which involves interpolating the missing colors based on its adjacent pixels' color information.

NOTE: Although ProvCam is constructed around CMOS sensors and their processing pipelines, it can accommodate other types of image sensors with little or no modification.

Image Signal Processor (ISP). ISP is a specialized hardware component used for pro-

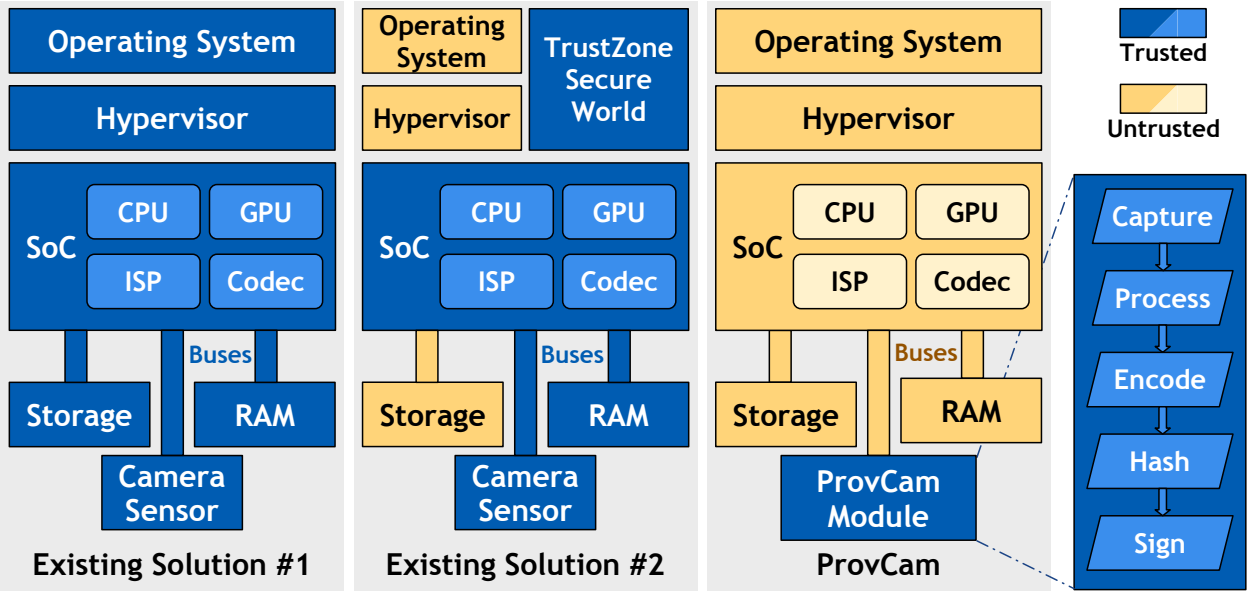


Figure 3.1: Comparison between two current approaches and ProvCam's TCBs. Detailed flow inside ProvCam camera module is shown in Figure 3.3.

cessing of image sensor outputs. It performs a range of image processing tasks in a serialized manner, including demosaicing, light and color corrections, noise reduction, sharpening, and possibly other optimization algorithms.

Video encoder. Each raw frame output from an ISP constitutes a substantial amount of data, and a compression algorithm is needed to make video storage and sharing practical. Advanced Video Coding (AVC, a.k.a., H.264), is a video compression standard widely adopted by modern camera devices [171]. There are other modern coding standards such as High Efficiency Video Coding (HEVC, a.k.a, H.265) [156] and AOMedia Video 1 (AV1) [96]. While we focus on H.264 in this work, ProvCam design can be modified to also support such alternative codecs.

3.4 Design

We now present the design of ProvCam. It is carefully crafted to reduce the TCB of the camera device for generation of video provenance information. ProvCam is a camera module that captures, processes, encodes, hashes, and signs the video, all in a single integrated

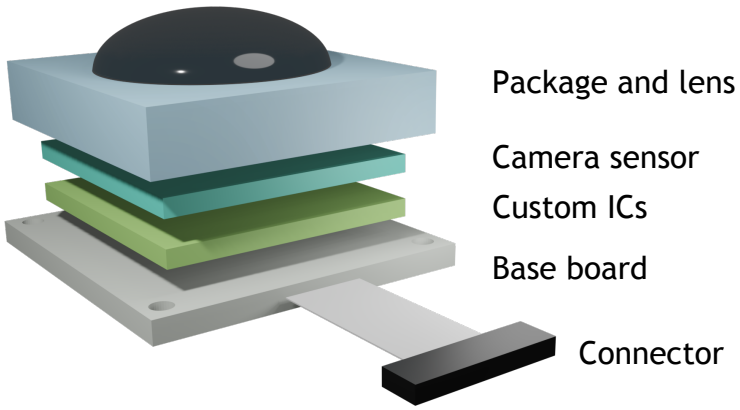


Figure 3.2: 3D-stacked secure packaging of ProvCam's camera module.

circuit package. The signature, along with the camera module certificate, acts as the provenance information, allowing the consumer to authenticate the source camera and verify video integrity. The self-contained camera module facilitates a tiny TCB. Figure 3.1 shows the TCB comparison of two current approaches discussed in §3.2.2 and ProvCam, as well as a high-level overview of how ProvCam is integrated into a mobile device.

Within the module, a sensor unit actively processes light information, and passes the information to an ISP and subsequently an encoder. After the encoder outputs an encoded stream of frames, a hasher calculates a rolling hash (which is based on frame data output by the encoder), and a signer signs the final hash.

To protect against attacks from untrusted hardware and software on a mobile device, ProvCam strongly isolates itself when capturing a video and generating its provenance information. It starts by a hardware reset, which flushes any residual/transient state which might be the result of untrusted usage of the ISP and encoder. (As discussed later in §3.6.3, ProvCam camera module also supports normal and untrusted usage.) The module blocks all commands from the CPU, and uses a simple internal Camera Control Unit (CCU) to fully control the camera pipeline. This strong isolation continues until the OS sends a termination request to the module, at which point the camera module finalizes the video and generates the provenance information.

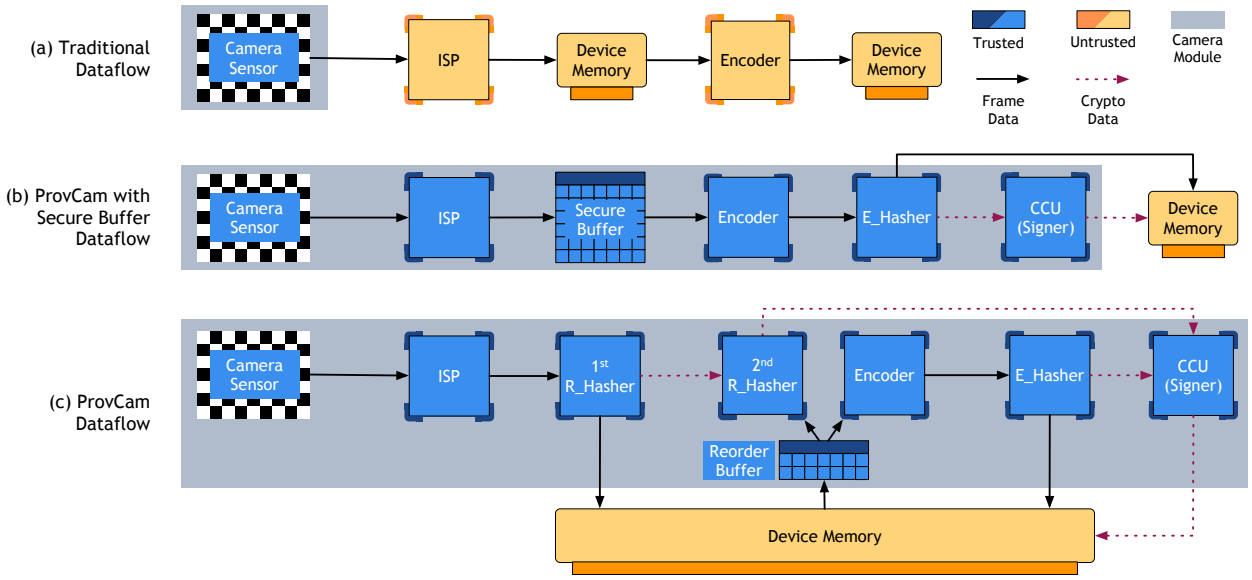


Figure 3.3: Comparisons between a typical video capture pipeline and ProvCam's secure pipelines.

All hardware components of ProvCam need to be housed in secure packaging in order to protect against physical attacks by the camera user, as discussed in §3.2.1. Since we rely on current methods to secure the hardware package against physical attacks, we are not concerned with the specific integrated circuit packaging format of ProvCam's camera module [67, 28].

However, to give an idea of what the module might look like, we illustrate one possibility. Figure 3.2 shows the use of 3D-stacked packaging for producing an ASIC-based camera module. Each layer of the camera is stacked and connected, leaving no pins exposed, thus preventing tapping on the internal data buses. Furthermore, the use of a tamperproof enclosure for the package makes it difficult to break into the package without damaging any internal components. In fact, 3D-stacked packaging is known to provide security benefits [108, 159], and there has been at least one proposal for applying such technology to CMOS camera sensors [112].

3.5 Hardware Design Challenges

3.5.1 The Use of Untrusted Main Memory

As illustrated in Figure 3.3 (a), in a conventional video pipeline, a camera sensor streams data into an ISP for processing. Then, the ISP generates and stores the frames in system memory. Subsequently, a video encoder reads the frames from memory, encodes them into a compressed video stream, which it then stores in memory.

Since ProvCam's primary goal is to generate the final provenance report fully within the module, the output frame from the ISP can not be stored in the untrusted main memory. Thus, in our original design, we used a secure buffer within the camera module, as shown in Figure 3.3 (b). However, a raw frame coming out the ISP can be large, e.g., a high-resolution frame can reach 10 megabytes. Therefore, the use of the secure buffer significantly increases the hardware cost of camera module (as we will empirically show in §3.9.1).

To address this issue, we instead securely use the untrusted camera device's main memory, as shown in Figure 3.3 (c). The key idea is to compute and compare: (1) hash of the frame going from ISP to main memory, and (2) hash of the frame going from main memory to encoder. If they match, it must mean that the frame has not been modified, despite residing in main memory. It it worth noting that this set of hashes is different from the set of hashes ProvCam uses to describe the final recorded video. After attempting this approach, we soon realized that the two hashes do not match, even if the frame was not modified! A further investigation showed that this occurs because ISP and encoder have different frame-access patterns. A high-level example of data transactions in Figure 3.4 (a) illustrates the problem. In writing a processed raw frame to memory ISP transfers 4 pixels linearly in each transaction. Whereas, while reading an ISP-processed raw frame from memory, encoder transfers 4 pixels in a block-like fashion. Different frame-access patterns result in data being written to memory in the different order from when it is read from memory. Thus, the two hashes do not match.

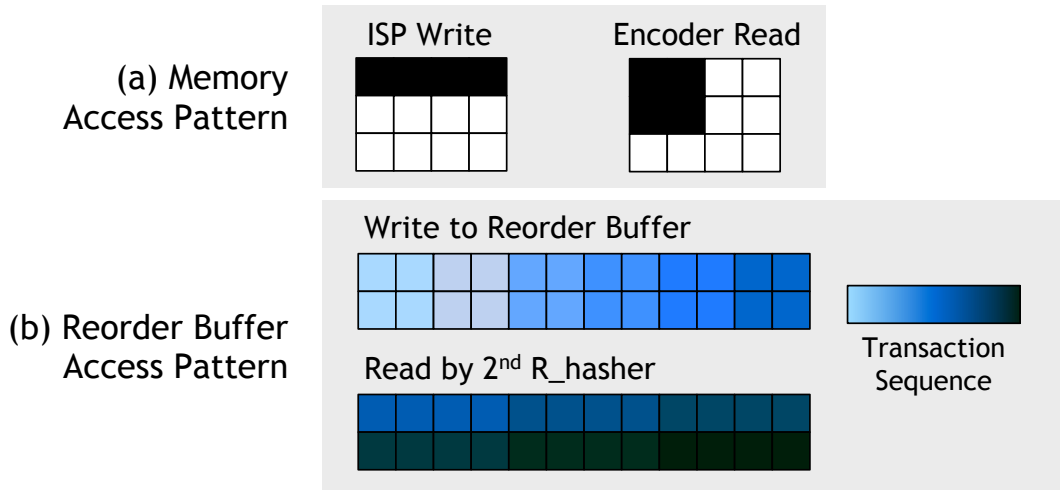


Figure 3.4: Comparison between how ISP and encoder write/read raw frames and how the secure reorder buffer addresses it.

We observe that, despite different access patterns, ISP and encoder both access a few rows of the frame at a time. That is, ISP outputs a few rows before moving on to the next set of rows. Similarly, encoder reads these few rows fully before reading the next ones. In the example of Figure 3.4 (a), the first two rows of pixels generated by ISP match those read by encoder. Therefore, we can create a secure reorder buffer with the smallest possible size between main memory and encoder to reorder the data in the same pattern as ISP writes into the main memory.

Figure 3.3 (c) illustrates the dataflow of ProvCam with the use of untrusted main memory and secure reorder buffer. It shows two hashers (called the first and second *r_hasher*s, where *r* means *raw*). They compute hashes of outgoing and incoming frames, respectively. Each time encoder reads a 4-pixel block from main memory, this block is stored in the buffer with awareness of frame structure, where block pixels are stored in their corresponding locations across the buffer. As illustrated in Figure 3.4 (b), when both lines of pixels are filled up, the reorder buffer starts feeding the second *r_hasher* in the same pattern as ISP writes to main memory, which is also how the first *r_hasher* gets its data. This ensures that, *iff* the frame does not change in main memory, the outputs of the two hashers match.

Note that we need to carefully take care of frame synchronization. The video capture pipeline

is typically loosely synchronized, where ISP could have processed more than one frame before encoder finishes hashing even the first frame. This means that the first *r_hasher* can produce more than one hash before the second *r_hasher* has a chance to finish reading the first frame. To address this, we implemented a small FIFO queue to hold all hashes generated by the first *r_hasher*. Whenever the second *r_hasher* finishes hashing a frame, it compares it with the hash from the head of the queue; If they do not match, a flag is raised in the second *r_hasher*, which will ultimately result in ProvCam refusing to sign the final video.

Finally, we note that the size of the reorder buffer is much smaller than that of the secure buffer in our original design. For example, at 720p resolution, the reorder buffer is 45 times smaller than the secure buffer. This allows us, as intended, to significantly lower the hardware cost.

3.5.2 Asynchronous Hashing

The data stream that serves as input for *r_hasher*s comes in at a very high rate. In contrast, incoming data stream for *e_hasher* (where *e* refers to **e**ncoded) is bursty. These aspects create challenges for the hardware hashers.

First, although a hardware hash engine can achieve relatively high throughput, it might not keep up with the incoming video frames when a high-resolution camera sensor is used (as in our prototype). And high-resolution camera sensors are increasingly common. For example, rapid advancements in camera technology for mobile devices yielded smartphones with 8K cameras [166].

Second, both ISP and encoder generate/read data in a bursty fashion. The bus protocol commonly adopted by mobile systems (and by our prototype), AMBA AXI (Advanced eXtensible Interface), is bursty. In a burst transaction, two AXI-enabled components transfer a pre-negotiated amount of data without the overhead of initiating and terminating transfers [34].

Based on the above, ProvCam hashers can not always satisfy the rate of the incoming data.

Throttling the data rate at any point of the pipeline is not an option because it would unavoidably degrade overall camera performance. Indeed, we tried to halt the pipeline for *e_hasher* to catch up with the encoder. This caused AXI and other hardware components to time out, rendering the camera system unusable.

In the end, we opted to integrate a small cache into both *e_hasher* and *r_hasher* hardware to allow fast access. Hashers use the cache to digest burst transaction data asynchronously, resulting in a consistent hash rate even if the incoming data arrive too fast.

To determine minimum cache size, we consider various parameters, illustrated with an example: Assume that encoder writes 128 bits/cycle for 16 clock cycles in a burst transaction, resulting in 2048 bits of data to be hashed by the *e_hasher*. We also assume a hardware hash module digests 512 bits (one block) every 16 cycles, where it has to wait for 4 cycles for the data to reach a block size. While it takes over 60 cycles to hash the first block, all residual data must be transferred simultaneously. Therefore, in this example, a cache size of $1536 = 2048 - 512$ bits is needed for the corresponding hasher to catch up with the burst transaction.

Our asynchronous hashing strategy introduces average theoretical latency of 2 clock cycles for all ProvCam hashers. This small latency is because data need to be copied to either a cache or an intermediate register within the hasher.

3.5.3 Video Frame-Based Hashing

In the original design, we kept feeding the *e_hasher* with encoded bitstream generated by the encoder. Whenever the entire pipeline halted, we would ask the *e_hasher* to finalize the hash with the size of all the encoded bitstream. However, we soon realized that such approach would create a vulnerability due to the fact that the encoder may be stopped at an arbitrary time. Specifically, there is a chance that ISP has already written a complete frame into memory, where the first *r_hasher* has this frame's hash. On the other hand, encoder is still compressing this frame, where parts of this frame are already encoded and

fed into the *e_hasher*. If the pipeline halts at this moment, the second *r_hasher* would never have a chance to compute this frame's hash, meaning that this frame is never verified (as untampered in untrusted memory) though some of its parts are encoded into the final video and used to produce the final hash thereof. An attacker can use this vulnerability to insert malicious data into the last frame of the video in order to facilitate a hash collision with the final hash.

Admittedly, this is a difficult attack since the attacker can only control less than a frame worth of data. However, we opt to address this: instead of hashing the encoded bitstream blindly, we take a frame-based hashing approach in *e_hasher*. Figure 3.5 illustrates this scheme. First, we make *e_hasher* capable of detecting slices (sub-frames), where it automatically finalizes the hash for each frame. Second, *e_hasher* uses a hash chain to link hashes of all the frames into one final hash. Finally, *e_hasher* always keeps the latest available hash, generated by the last finished frame. This way we can safely discard any incomplete encoded data at the end of a recording session and still verify the final hash.

In this frame-based hashing approach we could alternatively use a Merkle Hash Tree (MHT), instead of a hash chain. Each leaf node in an MHT would correspond to the hash of one of the frames. On advantage of using MHT is that it allows the consumer to authenticate a video segment, without having access to the entire video. Although this would be useful if only part of the video is shared. we postpone this to future work.

3.6 Software Stack Design Challenges

3.6.1 Replay-Based Bare-Bones Driver

With the design of camera module, we now have confidence that the data plane for video capturing is secure. However, all of the hardware components in the camera module need to be properly configured by software drivers for each capture session. These drivers, forming the control plane of the module, normally run on the device OS, which is untrusted.

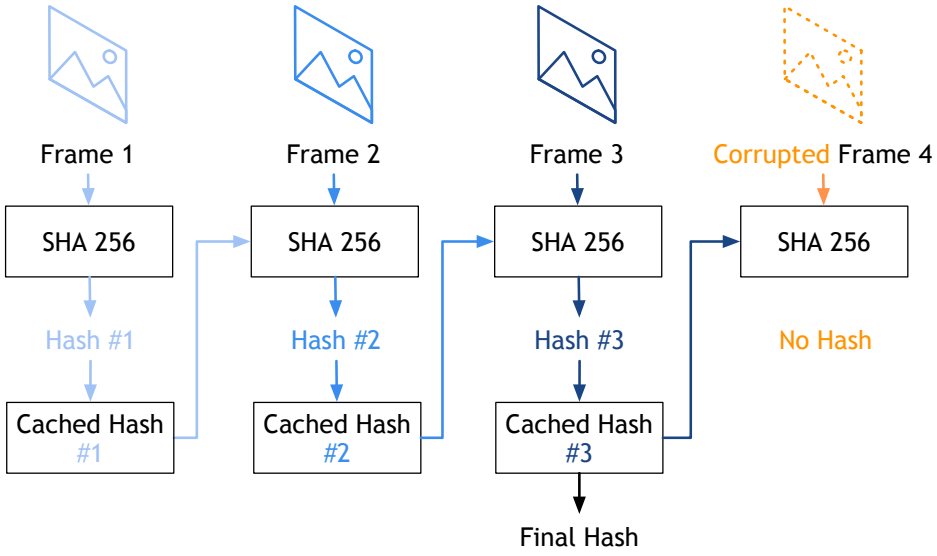


Figure 3.5: How frame-based hashing works in ProvCam.

Our goal is to contain our TCB within the camera module. Therefore, the control plane must be moved to the module as well. To achieve this, we move the control plane to run on the microcontroller in camera module, which is also used for generating provenance report of the video. We refer to this microcontroller as the CCU.

Porting all the drivers and media frameworks to the CCU is not a viable solution due to the limited resources at the microcontroller, and is against our principle of tiny TCB. On the other hand, it is not possible to distill a minimal set of codes from the drivers because they are all designed to be used in a video-capturing pipeline commanded by the hierarchical kernel and user-space media frameworks.

Fortunately, we observe that I/O transactions (i.e., writes/reads to/from hardware registers as well as interrupts) issued by the video-capturing pipeline are typically fixed (e.g., during initialization) or have fixed patterns (e.g., when handling an interrupt). Therefore, we use a record-and-replay strategy to address this problem.

First, we record all I/O transactions for three sets of transactions for each driver: initialization, start, and stop. For each transaction, we record the type (write/read/interrupt), target address, data size, and data content. (Note that in practice the recording process must be

conducted by the camera device manufacturer in a safe environment, where drivers running on the device OS can be trusted.)

We then provide the recorded logs to the CCU to replay when needed. (We currently add them to the CCU's source code, but could use a ROM as well.) Upon boot, the CCU replays the initialization presets for all hardware components inside of camera module. Upon receiving a command for starting a capture session from the device OS, the CCU replays all start presets of that configuration. During a capture session, when the CCU receives the stop command from the device OS, it proceeds to replay all stop presets, generate a signed report, write the report to the device's main memory.

Dealing with interrupts requires special care. We classify interrupts into two categories: deterministic and non-deterministic. Deterministic interrupts are the ones that are raised at known times and with a known frequency. This means that we can include them as part of the record-and-replay solution. On the other hand, there are non-deterministic interrupts, which could be raised at any given time. And depending on the length of the capture session, they may be raised multiple times. In addition, some of these non-deterministic interrupts require a read into the corresponding interrupt status register and perform a certain interrupt service routine (ISR). Therefore, to support non-deterministic interrupts, we port their ISRs from the corresponding drivers.

In our design, interrupts all go to our CCU, and not the untrusted OS. However, one interrupt is still needed by the device OS. More specifically, the device OS needs to be notified whenever there is a new encoded frame ready for it to read. Therefore, the CCU forwards this interrupt to the device SoC/OS. Moreover, this specific interrupt forwarding is a one-way communication: the CCU does not wait for the untrusted OS to clear the interrupt. This is mainly to ensure that this solution does not enlarge our attack surface. This should not cause any issues for the untrusted OS, which needs to keep track of all the interrupts/frames. This is because the device SoC is running at a much higher frequency than our camera module and therefore the OS has ample time to read one frame before the next one is written to

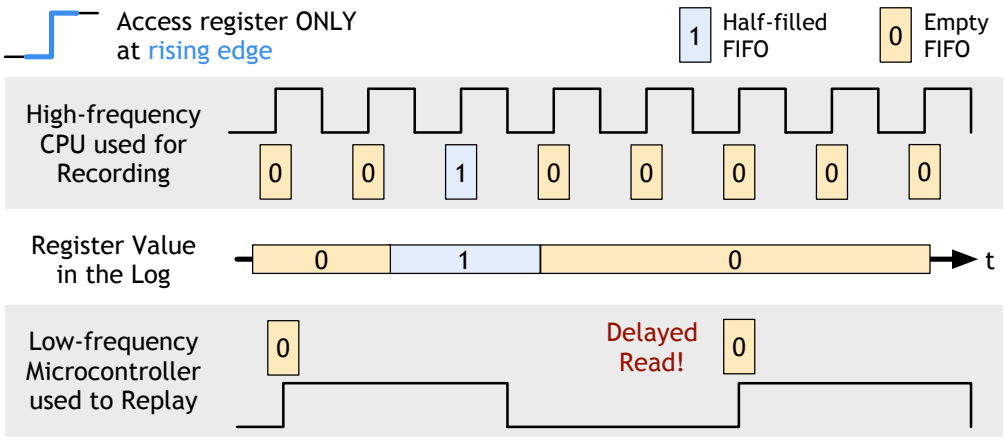


Figure 3.6: An example of mismatched I/O read of a FIFO's status register.

memory.

Finally, it is worth mentioning how we addressed a timing challenge in this design. More specifically, CCU runs at a drastically different frequency compared with the device SoC, which is used for recording the logs in the first place. This means that I/O transactions could be issued at a different rate, and interrupts could also be handled at a different relative time during record and replay. This creates some uncertainty when CCU replays the recorded command presets. In ordinary drivers, I/O reads are typically used to get the state of a hardware component. However, at the moment CCU performs an I/O read, the same hardware component may be in a different state. For some register reads, we may order CCU to keep reading the register until the desired state is reached. On the other hand, it is also possible that the hardware component has already transitioned past the desired state. For example, as illustrated in Figure 3.6, there could be a FIFO presented in a hardware component, where a register read shows if the FIFO is empty or not. The recorded I/O read value may indicate a half-filled FIFO, whereas CCU's I/O read value indicates an empty FIFO. To solve this, we scan through all status indication registers, locate all bits representing non-deterministic states, and ignore these bits throughout the command replaying phase. We also make sure that none of these bits can lead to potential vulnerabilities, where at best, an attacker may launch a Denial-of-Service (DoS) attack, as no custom command is allowed

to pass through CCU during the replay phase.

3.6.2 Double Replay for a Backward-Compatible Software Stack

Our goal is for ProvCam to be compatible with existing camera devices. Therefore, we do not want to introduce a completely new software stack to the OS just for ProvCam, in which case each type of camera device needs to modify its entire software hierarchy to adopt ProvCam. Instead, we would like to share the exact same camera drivers/libraries, media servers/frameworks, and applications with existing video-capturing pipelines, which means that all camera-related drivers' functions are executed normally during a capture session. However, in ProvCam, all I/O commands and most of the interrupts of camera-related components are performed within the module itself.

To solve this problem, we introduce *double replaying*: just like how we replay recorded commands in the CCU to all camera-related hardware components, we also replay the exact same chosen command presets in the device OS. More specifically, when a driver performs a read transaction, we return the read value from the recorded logs. And we silently ignore write transactions (since the actual write transactions are being performed within the camera module itself).

3.6.3 Trusted Capture Session

Capturing video with a modern mobile camera device is a highly customizable process, where users can not only select their desired frame rate and resolution, but also settings such as camera exposure, encoded format, and even some ML-based enhancing algorithms. Unfortunately, supporting all these features requires a significant amount of recorded logs in ProvCam, increasing the storage hardware cost within the module.

Our observation is that not all videos need to be verifiable. Therefore, we introduce the Trusted Capture Session (TCS) abstraction, which empowers our camera module to work both as a new type of trusted camera sensor and a traditional camera sensor. In other

words, ProvCam can either operate as a secure camera to capture a verifiable video (i.e., TCS mode), or can be used as a normal camera (i.e., non-TCS mode). In the TCS mode, we only support a single pre-determined configuration (requiring less than 200 kB of log storage in our prototype). In the non-TCS mode, all configurations are allowed.

We have so far discussed how secure capture (i.e., TCS) works in ProvCam. During a normal video-capturing session (i.e., non-TCS), the OS running on top of the device's CPU gains full control of the camera sensor, ISP, and encoder within the camera module, while the hashers and their assistant components (i.e., buffering modules) remain dormant. The camera module can operate in two modes in a normal capture session. First, the CCU can work as a middleman that forwards all I/O transactions between camera module's hardware components and the device OS, allowing all kinds of configurations to be used across these hardware components. Second, in most modern mobile camera devices, there exists more powerful ISP and encoder in the device SoC. In this case, our camera module can work as an ordinary camera sensor, without ever making use of its internal ISP or encoder.

The CCU always boots in TCS mode, where all other camera module hardware components are reset and initialized securely. After booting, the CCU is ready to accept commands from the device OS. If the very first command is for the TCS mode, the CCU continues to run in the TCS mode, and cannot switch to non-TCS mode before a reset. In contrast, if the very first command is for the non-TCS mode, the CCU switches to non-TCS mode, and cannot switch to TCS mode before a reset.

ProvCam ensures that all hardware is reset when transitioning between the modes. To enforce a reset from the TCS mode to the non-TCS mode, after generating the report in the TCS mode, ProvCam puts itself in an unusable state until it is reset. To enforce a reset from the non-TCS mode to the TCS mode, ProvCam uses a simple hardware flag. Any commands passing to the CCU in the non-TCS mode causes the flag to be set. A set flag prevents the use of the CCU in the TCS mode. The only way to clear this flag is to reset the CCU, which triggers a full reset of the entire camera module.

3.7 Prototype

We have built a prototype of ProvCam on a Xilinx Zynq Ultrascale+ MPSoC ZCU106 FPGA board with a Leopard IMX274MIPI-FMC camera sensor. We treat the ARM Cortex-A53-based processor, its DDR RAM, codes running on top of them (including the PetaLinux OS), all of its connected buses, and all their connected components except the camera module as untrusted. ProvCam's TCB is the camera module, which (with the exception of the camera sensor) we have implemented in the programming logic (PL) of the board, including MIPI CSI lanes, ISP, video codec unit (VCU), ProvCam hashing IP, a Microblaze based soft core working as the microcontroller (CCU), codes running on the microcontroller, memory (Block RAM for Microblaze and UltraRAM as secure/reorder buffer), and AXI buses connecting these components within the module.

Overall, the binary executable we generate for the microcontroller is around 450 kB (including 200 kB of recorded logs, which is equivalent to 6k 4-byte commands).

3.8 Security Analysis

We analyze and evaluate the security of our solution against various attacks in the adversary's arsenal (§3.2.1).

Attack 1 (generating a fake video) fails as the adversary does not have access to the cryptographic key and thus cannot produce provenance info for the video to prove its authenticity. Attacks 2 and 3 (modifying an existing video and/or its provenance info) fail since the consumer will not be able to verify the authenticity of the video given its provenance info. Attacks 4 and 5 (compromising the TrustZone normal world and/or secure world software stacks such as OS, hypervisor, and the secure world OS) fail since the camera module is fully isolated from these software components and generates the video and its provenance fully within the module. Attack 6 (replacing the camera sensor/module) fails too because the adversary cannot access the sensor within the module due to secure packing. Moreover,

replacing the module altogether is useless since the provenance of the video is produced inside the module itself. Attack 7 (active bus tapping) fails too since (1) the buses inside the module are protected by the secure packaging and (2) all data coming out of the camera module is already protected using cryptographic measures.

Therefore, our solution defeats the strong adversary discussed in our threat model.

One additional attack vector is noteworthy. The adversary could use ProvCam to record a video of a fake video played on a screen in front of it. In this case, ProvCam generates verifiable provenance info for this fake video. We leave addressing this attack vector to future work. But here we discuss some possible solutions. First, similar to the proposed solution in Vronicle [123], a more advanced (and hence more expensive) ProvCam could incorporate a depth camera and include the depth information in the signed video file. Alternatively, a cheaper option would be to include a flash in ProvCam and use that to extract depth information [86]. Second, recording off a screen could potentially create side effects in the recorded video, which could be detected post-factum, although we have not tested this hypothesis yet.

3.9 Prototype Evaluation

We perform all evaluations and experiments on our prototype reported previously. In addition to the ProvCam's prototype (which implements the design in Figure 3.3 (c)), we develop two other prorotypes in order to compare with ProvCam. One is the *baseline* which only captures, processes, and encodes videos (Figure 3.3 (a)). The other is a prototype of ProvCam with the use of a secure buffer inside the module (Figure 3.3 (b)). We build the latter in order to be able to evaluate the benefits and potential overheads of our solution of using the untrusted memory (§3.5.1).

We evaluate hardware cost, performance, and energy consumption. All camera pipeline related IPs run at 200MHz, except the MicroBlaze-based microcontroller (CCU) running at

100MHz. The quad-core ARM Cortex-A53 based application processor, used to implement the untrusted part of a camera device, runs at 1.2 GHz. For reference, Apple's A16 processor runs at 3.46 GHz [94], and Qualcomm's Snapdragon 8 Gen 2 processor runs at 3.36 GHz [140]. Therefore, we believe that our camera module, if implemented in ASIC, would achieve even higher performance and lower power consumption compared to what we report here. For all of the following experiments, our encoder uses H.264 intra-prediction mode, with 8-bit color, 720p resolution, and 60 fps.

3.9.1 Hardware Cost

We report the expected number of extra transistors needed for realizing ProvCam. We estimate it by measuring the number of additional look-up tables, flip flops, and SRAMs (including both UltraRAM and Block RAM) used by all additional (compared to the baseline) IPs and buses for connecting them. For conversion to transistor count, we assume each look-up table is composed of 6 NAND gates [139], each of which requires 6 transistors [155], bringing it to a total of 36 transistors for each look-up table. We assume each flip-flop requires 24 transistors [150]. Lastly, we assume each SRAM cell (bit) uses the 6-transistor design [60].

Table 3.1 shows the breakdown cost of ProvCam (main design and the design with the secure buffer). Our calculations reveal that ProvCam requires an additional (compared to the baseline) 53.8 M transistors. Moreover, they show that our use of the untrusted memory significantly reduces the hardware cost. More specifically, ProvCam with secure buffer requires an additional (compared to the baseline) 181.7 M transistors. This means that our use of the untrusted memory results in 70.4% reduction in the additional (compared to the baseline) transistors needed by ProvCam. The saving is because this design avoids using the large secure buffer in the module.

In fact, this saving will become more significant as the resolution of the video increases. This is because the size of secure buffer grows linearly with the number of pixels, whereas

Design	Resource type	Count	Expected transistor count
ProvCam with secure buffer	Look-up table	38,256	1,377,216
	Flip flop	48,393	1,161,432
	SRAM (buffer)	22,118,400 (bits)	132,710,400
	SRAM (others)	7,741,440 (bits)	46,448,640
	Total Transistors:		**181,697,688**
ProvCam	Look-up table	45,576	1,640,736
	Flip flop	58,838	1,412,112
	SRAM (buffer)	62,464 (bits)	2,998,272
	SRAM (others)	7,953,408 (bits)	47,720,448
	Total Transistors:		**53,771,568**

Table 3.1: Extra hardware cost in both hardware prototypes.

	480p (SD)	**720p (HD)**	**1080p (FHD)**	**2160p (UHD)**
Secure Buffer	44.2 M	132.7 M	298.6 M	1194.4 M
Reorder Buffer	1.5 M	3 M	4.4 M	8.9 M
Ratio	30×	45×	67.5×	135×

Table 3.2: Comparisons of the amount of transistors needed for different resolutions between secure buffer and reorder buffer for untrusted memory.

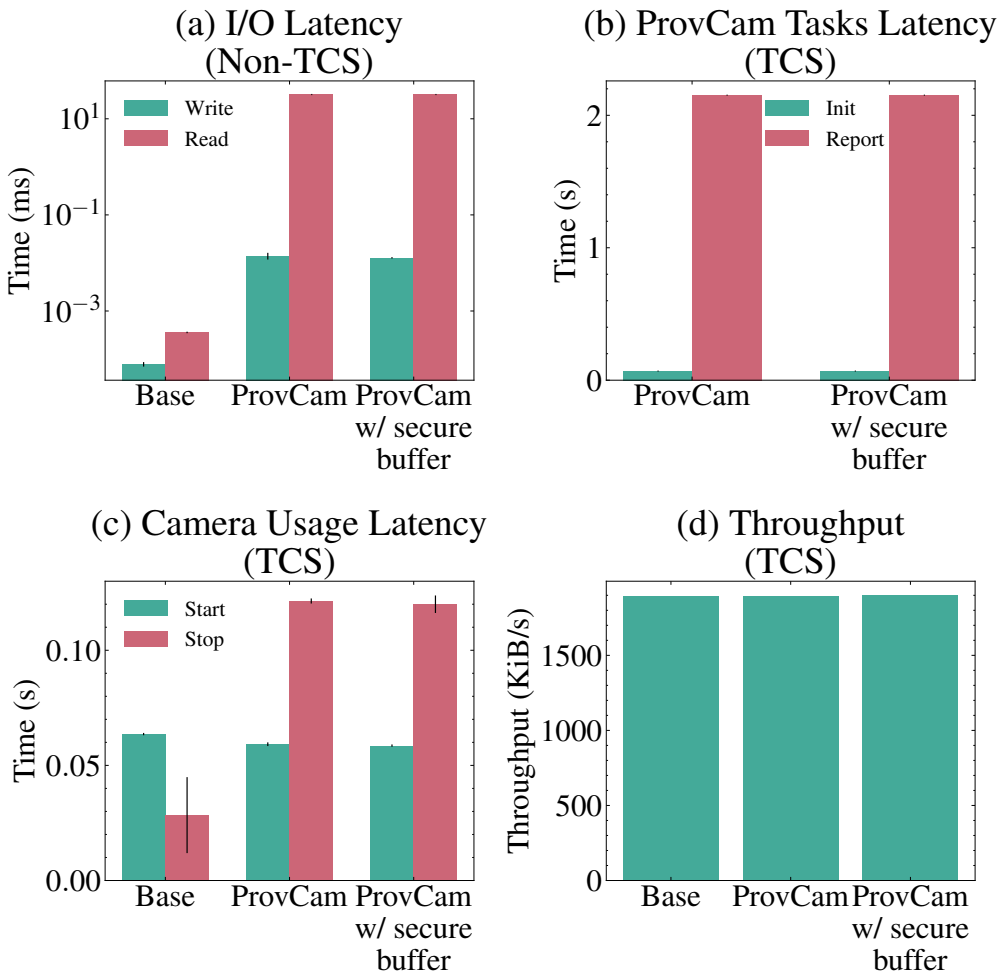


Figure 3.7: Performance evaluations of ProvCam. The histograms show the average. The error bars show the standard deviation.

the size of reorder buffer for untrusted memory only grows logarithmically. Table 3.2 shows the expected number of transistors needed for both designs for higher resolutions. It shows that our use of untrusted memory can lower the required transistors by 95.2% for the UHD resolution.

Finally, we note that the overall hardware cost of our camera module is trivial if compared with the amount of transistors used by the modern mobile SoC's. For example, both Apple A16 and Qualcomm Snapdragon 8 Gen 2 have around 16 B transistors [79].

3.9.2 Software TCB

Software-wise, ProvCam only trusts codes running on the microcontroller. Here we report LoCs of ProvCam's codes running in the TCB: ~4k for drivers, ~4k for bare-metal libraries, and ~60k for C runtime library. It is noteworthy that not all 60k lines of codes of the C runtime library are compiled into the final binary. In comparison, we also report our estimated LoCs of a TEE-based alternative implemented with OPTEE [15]. There are ~100k for drivers, 231k+ for media API frameworks (e.g., GStreamer and V4L2) that are required by camera/video pipeline drivers, and ~350k for OPTEE. To summarize it, ProvCam achieves at least 10 times smaller software TCB compared to its TEE alternative.

3.9.3 Performance

We evaluate latency and throughput, with two types of experiments: microbenchmarks and macrobenchmarks. For microbenchmarks, as shown in 3.7 (a), we measure the amount of time needed for I/O transactions in the non-TCS mode, since in this mode, the transactions need to be forwarded by the microcontroller. In addition, in 3.7 (b), we show latencies of various tasks' performed by the microcontroller in TCS mode, where the report generation time does not necessarily block the pipeline, and can be done in the background.

For macrobenchmarks, we present the latency of starting and stopping the video capturing pipeline in 3.7 (c). We also measure and report the throughput in 3.7 (d). Throughput here refers to the throughput of generated encoded bitstream.

Our macrobenchmark results show that both our hardware prototypes perform similarly compared to the baseline. We would like to highlight that our throughput experiment shows that ProvCam can sustain a high resolution (720p) at a high framerate (60fps). Moreover, this shows that the *r_hashers* in the module manage to sustain a much higher throughput (i.e., 82.9 MiB/s) since they operate on raw data.

Overall, we conclude that although ProvCam brings moderate overhead in microbenchmarks, the impact on macrobenchmarks is negligible.

3.9.4 Energy Consumption

We estimate the energy consumption using the power report of our hardware prototypes generated by the Xilinx Vivado software [174]. We set the ambient temperature to be 25.0 °C (77.0 °F), with no heatsink expected on a target board size of JEDEC 2S2P. Moreover, we run the power report multiple times with maximum possible load on all IPs to always measure the worst-case power consumption results. Our baseline design, which has a typical modern video recording pipeline, running at a typical load with the same configuration above, consumes 5.274 watts of power. Note that all numbers we report below are additional amount of energy needed by ProvCam camera module. There are a total of three fixed tasks performed by ProvCam camera module: starting the pipeline, ending the pipeline, and generating provenance report, where they last for a fixed amount of time during each recording session. For starting the pipeline, ProvCam with secure buffer consumes 0.028 ±0.0003 joules, where ProvCam consumes 0.035 ±0.0005 joules. For ending the pipeline, ProvCam with secure buffer consumes 0.057 ±0.0018 joules, where ProvCam consumes 0.071 ±0.0007 joules. For provenance report generating, which is solely executed by the microcontroller, ProvCam with secure buffer consumes 0.616 ±0 joules, where ProvCam consumes 0.613 ±0 joules. In total, ProvCam with secure buffer needs 0.701 ±0.0021 joules to handle all three fixed tasks in each recording session, where ProvCam needs 0.719 ±0.0012 joules. On the other hand, video recording time varies among different videos; therefore, we also report both hardware prototypes' power consumption rates, which can be multiplied with expected recording duration time to get the total energy consumption for a recording session. ProvCam with secure buffer uses 0.476 watts, where ProvCam uses 0.587 watts. Both prototypes show that ProvCam introduces tolerable energy consumption overhead compared to the baseline video capturing pipeline. Moreover, this consumption is insignificant compared with the large capacity of batteries on modern camera devices, such as Apple iPhone 14 Pro Max, which has a 16.75 Wh (60300 joules) battery [89], and Samsung Galaxy S23 Ultra, which has a 18.5 Wh (66600 joules) battery [82].

	Approach	Performance	Videos?	Vulnerable to attacks (Refer to §3.2.1)	TCB
Aletheia [22]	Digital Signature	No measurement	Yes	4,5,6,7	OS
PhotoProof [131]	Zero Knowledge Proof	673.5 secs for 1 frame at 128x128	No	4,5,6,7	OS
YouProve [91]	Image Understanding	28 secs for 1 frame at 1296x792	No	4,5,6,7	OS
TrustEYE.M4 [172]	TPM	11 fps at 320x240	Yes	6,7	TEE + others
TrustCAM [173]	TPM	25 fps at 640x480	Yes	6,7	TEE + others
FrameProv [55]	TEE	No prototype	Yes	5,6,7	TEE + others
Vronicle [123]	TEE	Native	Yes	5,6,7	TEE
ProvCam	Secure Camera Module	Native	Yes	None	Tiny

Table 3.3: Comparison with related work. Note that TEE + others means the system TCB includes not only TEE, but also other (potentially less trustworthy) components

3.9.5 Programming Effort

As there are various models of camera sensors, ISPs, and encoders, we expect different combinations of them to be used for making a ProvCam-based camera module. We expect a small amount of effort when adapting our system. We measure the potential amount of codes that need to be changed in the drivers.

First, the driver needs to include our provided recording, replaying, and TCS libraries. Calling into these libraries needs around 20 to 50 lines of code. Second, in case the driver has any dynamic (unpredictable) interrupt handling, the corresponding handler codes need to be moved to the CCU for TCS mode. Also, any interrupt that requires forwarding in either TCS or non-TCS mode needs to have a CCU's GPIO interrupt registered in the OS device tree.

Overall, we expect a small amount of effort to be spent on adapting ProvCam to use different hardware components inside the module.

3.10 Discussions & Future Work

3.10.1 Trusted and Normal Videos

We have mentioned in Section 3.6.3 that we allow both the secure and normal video capture pipelines to operate at the same time on a mobile device, generating two videos using raw

frames captured by the same camera sensor: one trusted but with relatively low image quality, and one with high image quality but untrusted. We envision some use cases of this capability. For example, users can store the trusted video locally and share the high-quality one with others. In the future, if needed (e.g., if the video is found to be used as evidence in a court case), the user can present the trusted video for verification. The challenge here is to verify that the two videos have captured the same content. Machine learning and fuzzy hashing are two possible approaches that can be used to address this challenge. However, as they are likely to have different resolutions, FPS, color space and so on, where details preserved in one video could be almost completely lost in the other one, we can at best hope for an approximation-based solution (e.g. giving a confidence score to the normal video based on its similarity with the trusted video).

3.10.2 Privacy and Identity

In ProvCam's current design, each camera module is given a fixed ECDSA key pair at manufacturing time, and it uses this key to sign all videos it captures. This raises a privacy issue: videos captured by the same user can be linked together, potentially revealing their identity. This might be problematic as some users might not want to reveal their identities when sharing a video. One could use a group signature scheme such as Direct Anonymous Attestation (DAA) [70] and lattice-based group signature [93], similar to the one used in modern TEEs, to provide full anonymity. However, this solution might not be ideal either as users may want to reveal their identities for some of the videos they capture. For example, a citizen journalist may want to reveal their identity in order to receive credit for a video they capture. We leave finding a desirable solution to this problem to future work.

3.10.3 Key Revocation

As mentioned above, each camera module is equipped with a fixed ECDSA key pair at manufacturing time, which is later used to sign captured videos. Although we put the entire

camera module into ProvCam's TCB, it is still possible that the camera module and/or its embedded key pair is compromised by attackers. In this case, a key revocation process is necessary. We believe existing key revocation solutions should be applicable here as well.

3.11 Conclusions

ProvCam ensures the authenticity of videos by protecting camera capture, image processing and video encoding in a tiny TCB and hardware design that defends against sophisticated attackers. The cryptographic proof from our system, possibly combined with proofs from other systems that verify subsequent stages of video post-processing, can finally give the viewer the assurance that what they are seeing happened in reality. This assurance is becoming more urgent in an age of deepfakes, generative AI, and AI embedded inside smartphone camera pipelines. The implications span many important parts of society, including justice and news. We expect the technology will be applicable even beyond smartphones and body cameras, to eventually include security cameras, cameras on autonomous vehicles and robots, and remote tele-medicine.

Chapter 4

Scoop: Mitigation of Recapture Attacks on Provenance-Based Media Authentication

4.1 Introduction

Today, digital media is constantly produced and consumed in enormous volumes. It is an undisputed fact that modern society is dependent on and, to some degree, even addicted to it. Due to the global popularity and affordability of digital cameras and camera-equipped smartphones (and, increasingly, eye-wear and AR/VR headsets), almost everyone can create visual content (photos and videos) anywhere anytime. Furthermore, ubiquitous Internet connectivity allows visual content to be easily shared online. It is estimated that over 3.2 billion images and 720,000 hours of videos are shared daily on various social media platforms [160].

Unfortunately, lots of visual content found on the Internet is not real, ranging from manipulated to fully synthetic. The unprecedented rise in misinformation attributed to deepfakes triggers an urgent need for authenticity and integrity of visual content. To this end, both



Figure 4.1: Demonstration of effectiveness of recapture attacks. One of these photos is real (i.e., captured by a camera in a real environment) and one is a recapture attack (i.e., captured by a camera pointed at a screen showing a modified photo). Even though we have used a mid-range TV, detecting the attack is very challenging (especially if the user is not suspecting an attack). Section A.1 mentions which one of these photos is real.

industry and academia developed various provenance-based techniques [18, 25, 123, 124, 131, 91]. The key idea behind this line of work is the use of cryptographic metadata attached to visual content, coupled with a secure camera design. The end-result is the ability for anyone to authenticate the source camera, the content it captured, and any post-processing filters applied to that content. Based on secure components, such as a TEE [3, 123] and specialized hardware, such as a secure camera module (sensor) [124], provenance-based techniques aim to provide strong security for generation and subsequent (benign) modification of visual content.

Many prominent industry players from diverse sectors (e.g., Canon, Adobe, BBC, and Microsoft) already committed to – or even commercialized – provenance-based media authentication techniques [18]. For example, Truepic teamed up with Qualcomn to provide provenance-based media authentication for smartphones with Qualcomm Snapdragon SoCs [49]. Another example is Adobe's recent integration of content authenticity into its tools [39].

This chapter sheds light on an important vulnerability in provenance-based techniques: vulnerability to *recapture attacks*. In such an attack, the adversary prepares some fake content, which it then either: (1) displays on a screen, e.g., TV, monitor, or projector, or (2) prints or replicates on some physical surface, e.g., paper, cardboard, or canvas. Finally, the adversary captures this fake content with a camera that uses a provenance-based technique. The fake content might comprise the entire frame or part(s) thereof.

The adversary's goal in a recapture attack is to present all captured content as real, including the fake parts. Since a provenance-based camera cannot distinguish real content from displayed, painted, or printed one, the entirety of captured content gains credibility as having been generated by a secure camera. Figure 4.1 demonstrates the effectiveness of this attack. It shows two photos, both of which are captured by the same smartphone. One of these photos is captured in the real physical environment depicted in the photo. The other is a fake, manipulated version of the original photo displayed on a TV and recaptured by the same smartphone. Detecting the malicious photo is very challenging, if not impossible, especially if the viewer is unaware of the attack. To demonstrate this vulnerability, we conduct a user study. In §4.2.3, we present the results of this user study that show that people are not able to to distinguish between an original photo and a photo recaptured from a digital screen.

To mitigate recapture attacks in general, we introduce Scoop. It uses depth information captured at the same time as the visual content with a time-of-flight (ToF) sensor present on the camera to detect flat display mediums that could be used to mount recapture attacks (e.g., TVs, printed cardboard cutouts, and projector screens). Scoop tries to detect *misleading* recaptures, i.e., a recaptured photo or video where the presence of a display medium is not visually identifiable. We note that not all misleading recaptures are malicious, hence we intend Scoop as an assistant tool, helping the user detect recapture attacks.

Although the use of depth may seem intuitive and/or trivial, it poses an important challenge: the majority of flat surfaces are not display mediums that can be used in recapture attacks,



Figure 4.2: An example of a recapture attack on a provenance-based secure camera.

e.g., walls and floors. To address this challenge, Scoop computes a learning-based depth map of the scene, which provides us with the perceived depth in the same photo. When the perceived depth map fails to match the real depth map generated by the ToF sensor, Scoop marks that region of the photo as *misleading*, prompting the viewer to pay more attention to it.

To evaluate the efficacy and practicality of Scoop, we assemble a first-of-its-kind dataset, which contains 122 unique data points, including 78 recapture scenarios and 44 benign, ordinary daily scenes.

To demonstrate Scoop's viability and practicality, we built a prototype composed of two parties: producer (camera) and consumer (viewer). The former runs on both Apple iPhone 14 Pro (w/ dToF sensor) and Samsung Galaxy S20 Plus (w/ iToF sensor), while the latter runs on a Linux desktop. Evaluation results show that Scoop achieves up to 94.81% accuracy of detecting with as low as 0.02% false positives on an iPhone and 74.03% accuracy and 17.78% false positives, respectively, on a Samsung phone.

This chapter makes the following contributions:

- Discussion and analysis of recapture attacks against provenance-based techniques.
- Design of Scoop, a technique that blends depth and visual information and can effectively detect misleading recaptures using flat display medium.
- A first-of-its-kind dataset containing 122 unique data points (both photos and videos) including 78 recapture scenarios, serving as a cornerstone for evaluating systems such

as Scoop.

- A fully functional Scoop prototype on commodity devices and a thorough evaluation thereof, including effectiveness, performance and storage overhead, and energy consumption.

4.2 Recapture Attack

Workflow and example. Figure 4.2 illustrates a recapture attack. The photo on the left is real and captured by a smartphone. The one in the middle is a recapture attack. To conduct this attack, the adversary digitally modifies the original photo to remove the wallet, displays the modified photo on a TV screen, and recaptures it using a provenance-based secure camera. The result is a fake photo that looks very real and, even worse, is endorsed by a provenance-asserting camera, creating a false and dangerous sense of authenticity for the viewer.

Admittedly, such attacks raise the bar for the adversary in terms of the amount of effort required. Purely digital fakes can be easily generated in large quantities by software tools with minimal human involvement. In contrast, recapture attacks require the adversary to physically stage the scene. However, the cost is not prohibitive since recapture attacks can be orchestrated with minimal equipment (e.g., a TV) and relative ease (e.g., mounting a phone in front of a TV).

History. Recapture attacks are not a new concept [84, 106, 161, 71, 163, 118]. In the past, these attacks have been used to deceive face authentication systems (aka face spoofing), since most early facial recognition systems lacked liveness detection and/or depth-based defense means [126]. Printed photos were a common way of mounting such attacks [126]. However, the display technology evolved to the point where a screen could display images with the same quality as that of printing. Plus, digital displays support high-definition video playback. Consequently, screens became the dominant tool used in recapture attacks [68].

Much effort has been made to mitigate these attacks, and some countermeasures are already deployed commercially [117, 122, 85, 50, 32].

Why relevant now? We found that many current countermeasures focus on one specific problem (i.e., face authentication), rather than providing a systematic approach to mitigating recapture attacks. One important reason is that recapture attacks have only targeted specific systems, such as spoofing face authentication systems. Indeed, in the past, there was no need for recapture attacks to deceive people with some fabricated information, since no provenance-based techniques were deployed and the adversary could simply present fake visual content.

4.2.1 Categorizing Recapture Attacks

The envisioned recapture attack is conducted by displaying fake content on some display medium and capturing that with a provenance-asserting camera.

One component needed to mount the attack is the *display medium* – the physical medium used to display fake content. We identify three categories for the display medium:

1. Screen-based (e.g., LCD/OLED TVs)
2. Projection-based
3. Print-based (e.g., Inkjet/laser/3D printing)

Moreover, the display medium could be *flat*, *curved*, or custom-shaped (e.g., a 3D-printed model).

We also categorize recapture attacks into *full-recapture* and *partial-recapture* attacks, which are demonstrated in Figure 4.3. In the former, all contents (i.e., pixels) captured with the camera are recaptured, e.g., the camera points directly at a TV screen and captures part of the screen, while the TV is displaying fake content. In the latter, captured contents (i.e., pixels) contain both genuine and recaptured information, e.g., the camera points at a TV, while a real person is standing in front of the TV screen. In this example, the screen could



(a) **Full-Recapture**



(b) **Partial-Recapture**

Figure 4.3: Illustration of full-recapture (a) and partial-recapture (b) attacks.

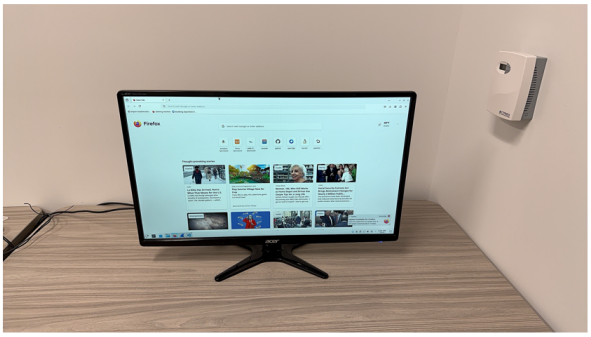




Figure 4.4: These two example photos, even if display mediums (i.e., a monitor and a TV) are presented in them, do not constitute an attack since the presence of the medium can be recognized by the viewer.

display a place of interest, which would create a false sense of the person being physically present at the displayed location.

4.2.2 Terminology

We categorized recaptured attacks based on the type of display medium. However, the presence of one of the aforementioned mediums in a photo/video does not constitute an attack if the viewer can clearly recognize it. Moreover, whether a recapture is an attack or not depends on the content, i.e., whether it has been maliciously created. In this subsection, we more accurately define the terms we use to refer to recaptures.

We define a photo/video to be a *recapture* if it is taken off a display medium (fully or partially). However, sometimes, the presence of the display medium might be visually iden-

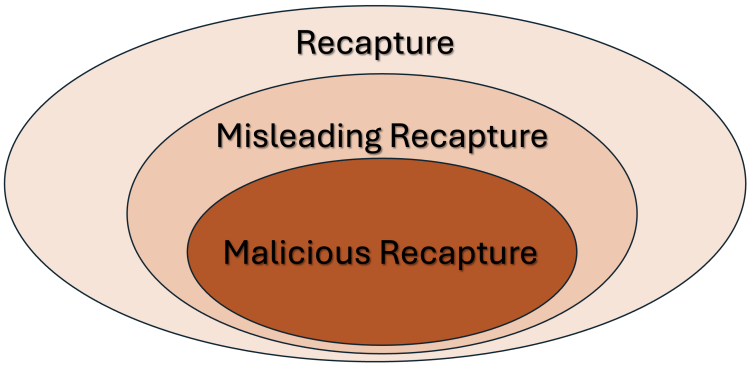


Figure 4.5: Terminology of recaptured visual content. Note that Recapture is the largest set, with Misleading Recapture as a subset of it, followed by Malicious Recapture as a subset of Misleading Recapture.

tifiable. This happens when the frames/boundary/edges of the medium are clearly visible. Figure 4.4 shows two such examples.

When the display medium is not visually identifiable, we refer to the recapture as *misleading* since the viewer might not realize that the content is recaptured. Note that a misleading recapture is not necessarily malicious. For a misleading recapture to count as an attack, the content shown on the display medium must be maliciously crafted in order to fool the viewers in some way. For example, consider a user taking a photo of a large TV showing some important event (a *full-recapture*). Viewers of this photo might be misled into thinking that the photo was taken during the event. However, this photo is not maliciously crafted and does not intend to be an attack. Figure 4.5 illustrates the aforementioned terminology. As mentioned earlier, Scoop's goal is to identify misleading recaptures and highlight them to the viewer, enabling the viewer to further analyze them in order to detect attacks.

4.2.3 User Study

As mentioned, Scoop's goal is to identify misleading recaptures. A key question is whether users themselves are able to detect misleading recaptures or not. In other words: are today's display mediums and cameras capable of creating a misleading recaptured photo/video with adequate quality that goes undetected by a human viewer?

We conduct a user study to answer this question. In the user study, we only focus on using large, high-quality TVs (see § 4.8 for the TV models) as the display medium. While we believe high-quality projectors and printed cardboard cutouts can also go undetected by viewers, we do not own such display mediums in order to test that hypothesis. In addition, we focus on *full-recaptures*. This is because we deem *full-capture* attacks as the most common and effective recapture attack at the moment. We do however believe that *partial-recaptures* can also be effectively used to mount attacks, but we do not evaluate that in this user study. In the user study, we showed 16 photos (8 original and 8 recaptured) to each user in a random order. Photos are selected to cover as much diversity as possible, i.e., lighting conditions, distance to objects, and complexity of the scenes, where all of them can be found in Appendix A.2. We recruited a total of 43 adult participants from the authors' institution. They have an average age of 24 years old, where 28 of them are in graduate degrees, and 15 of them are in undergraduate degrees. Among all participants, 25 of them are male and 18 of them are female.

Each session of the user study was conducted in person with one participant. All participants used the same device: a laptop featuring a 14-inch 2K resolution screen. To minimize potential psychological stress, the user study was unsupervised, where each participant took the survey with the laptop in an empty room using the QuestionPro [5] platform. The survey first provided each participant with the definitions of an original photo and a recaptured photo. Then for each photo in the survey, it gave them two choices to select from: original and recaptured. At the end of the survey, we asked participants about their experience completing it and confidence in their responses.

The results prove the fact that original and recaptured photos are perceptually indistinguishable. To demonstrate it, we conducted a one-sample t-test comparing our mean accuracy against chance performance. The results show that participants' (correct classification) accuracy 50.15% ($SD = 13.89\%$) is close to pure chance at 50% ($t(42) = 0.071$, $p = 0.944$), where the 95% confidence interval is between 45.88% and 54.42%. The negligible effect size

($d = 0.011$) indicates that any deviation from chance performance is practically meaningless. We also note that 44% of our participants classified exactly 8 out of 16 photos correctly. Additionally, most participants of our survey also stated that they did not have any confidence when they were making their decisions.

Finally, we note that in the user study, the participants were made aware of the presence of misleading recaptures and yet could not identify them. In realistic attack scenarios, users will be unsuspecting and hence are very unlikely to identify a misleading recapture.

4.3 Threat Model

There are two entities in Scoop interacting with photos or videos: producer (camera owner/operator) and consumer (content viewer). We assume the adversary to be the former, who might try to capture malicious photos or videos in order to deceive the latter who is the victim here. The adversary uses a provenance-asserting camera to take a photo/video of a display medium showing fake content. The adversary can show arbitrary fake content on the display. They cannot, however, modify the final photo/video captured by the provenance-asserting camera. Adversary's attempts to tamper with the captured content will be detected in the provenance verification process. (Please see §4.10.5 for a discussion on some limited form of provenance-based post-processing that could be allowed on the final photo/video.) The adversary cannot mount any software or hardware attacks on the camera device either, which is deemed trusted (e.g., secured with a trusted camera sensor [124]). In other words, we assume that the captured content's provenance information can be reliably and securely generated. This assumption is based on significant advancements in minimizing the TCB for provenance-based media authentication systems by both academia [123, 124] and industry [49, 18, 25] over recent years.

In this work, we focus on recapture attacks using flat display mediums (see §4.2.1). Attacks using curved or custom-shaped display mediums are out of our scope. We note that our

solution is likely able to handle curved display mediums. However, we do not have any samples of such attacks in our dataset and hence cannot evaluate our approach. (But we note that even the flat display mediums we use in our dataset are not completely flat and they can be slightly curved, textured, or bumpy.) On the other hand, our solution, as it stands, fails against custom-shaped medium (e.g., a 3D-printed model) since the depth map of the medium matches the learning-based perceived depth map. (Please see §4.10.1 for a more detailed discussion on 3D display-based attacks.)

In addition, we consider any objects outside of the supported range of the depth sensors as out of scope as well. Existing ToF sensors on smartphones have limited range – about 8 meters or 26 feet in our experiments. Those on autonomous vehicles have much higher range. While not a fundamental limitation of Scoop, we believe the ≈ 8 meter range is sufficient to detect most indoor misleading recaptures, as the cost of very large screens to mount attacks beyond 8 meters is very high. We hope that Scoop inspires smartphone vendors to include more powerful ToF sensors. Some professional-grade cameras have ToF sensors with more range, such as the DJI Zenmuse L2 with a range of 450 meters or 1476 feet [52].

Finally, we note that Scoop cannot distinguish between a flat surface such as a wall, and a display showing an image of a wall. In such situations, both the ground truth depth map and the learning-based depth map would match. We do note that meaningful attacks could exist in such scenarios, such as a recapture of a digitally modified signed contract. Scoop is not capable of detecting such recaptures.

4.4 Overview

Scoop tries to detect misleading recaptures. These recaptures can then be brought to the attention of the viewer, who can determine whether they are malicious or not.

Our first key idea to detect recaptures is to use a Time-of-Flight (ToF) depth sensor, available in some modern smartphones (almost all newer iPhone models and some Android devices)

and cameras, to capture depth information of the scene. ToF depth sensor is a kind of camera sensor that measures the distance between the sensor and objects in front of it, thus providing a precise depth map [48]. Since we focus on recaptures that use flat surfaces, the idea is to use the ToF sensor to detect misleading recaptures.

However, mere use of depth information to mitigate recapture attacks faces an important problem: the existence of many non-display flat surfaces (e.g., walls). Scoop's goal is to detect display mediums, but not these other flat surfaces. To do so, Scoop computes a learning-based depth map of the scene. By comparing this map with the one generated by the ToF sensor, Scoop can detect content that has visual depth information and is shown on a display medium.

Insights behind our approach. We begin by looking at how and why recapture attacks work. They work by trying to make the viewer believe that the content shown in the photo/video is captured at its original scene rather than on a display medium, such as a poster or a TV screen. These attacks mainly abuse the fact that the viewer can only observe the captured content at the viewpoint decided by the photographer, i.e., in 2D. Moreover, since both printing and digital displaying technologies quickly advance, it becomes hard for a human eye to tell if they are looking at a real scene or a photo/video being shown on a medium at certain angles. Therefore, certain recaptures can mislead viewers into believing that a fake scene is real, i.e., has depth. As a result, detecting misleading recaptures requires us to understand human perception of the depth of an image. The main idea here is to achieve that using AI.

As is well known, AI has been dramatically growing in popularity in recent years. Various learning-based models have advanced to the point where they perceive information akin to humans, including monocular depth estimation, which is a kind of model that estimates depth using only a single photo containing RGB information.

After some initial experiments, we found that, similar to humans, such models can be easily tricked by recapture attacks. For instance, we conducted an experiment where an RGB

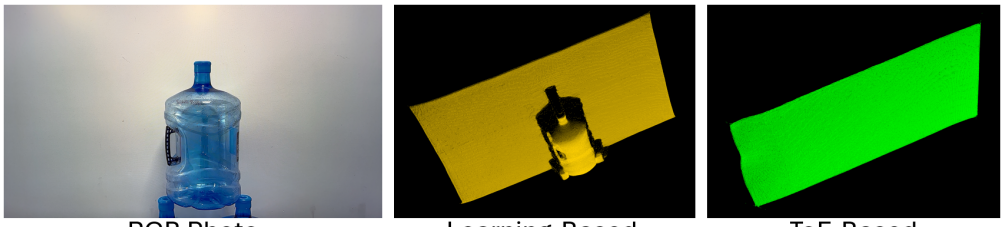


Figure 4.6: Illustration of a learning-based depth estimation model got tricked by a photo displayed on a TV, while a ToF sensor captured depth map correctly represents the depth of the real scene.

photo of a real scene was displayed on a TV screen and then was captured with: (1) TV screen frame being visible (i.e., a non-misleading recapture) and (2) without it being visible (i.e., a misleading recapture). In (1), depth estimation models did not recognize any depth in the photo, just a flat surface. In contrast, in (2), all models [65, 175, 176, 178, 102, 142, 64, 92, 62, 137] estimated depth information based on the content displayed on the TV screen. While monocular depth estimation models fail to recognize a recapture attack in (2), the ToF depth sensor can recognize that there is no depth in this scene, since it measures the physical distance between the camera and each point in the scene. Figure 4.6 illustrates the depth maps generated using monocular depth estimation model and the ToF sensor.

Based on the above observations, our approach involves comparing the depth information generated by the learning-based method and the ToF sensor. If there is a significant discrepancy between the two, it is likely that we have identified a misleading recapture.

Alternative approaches. One might wonder whether providing verifiable provenance for time and location of content might be adequate to detect misleading recaptures. We note that these two types of metadata can help identify some misleading recaptures, but they are not always useful, i.e., when the content itself does not suggest a clear time or location. Moreover, providing authentic time and location metadata in the camera is not straightforward. For time, the camera needs to synchronize its clock to an external reference server. For location, it needs to use an external signal, e.g., GPS. However, GPS is unavailable for indoor scenarios and vulnerable to spoofing through low-cost tools [169, 144, 179].

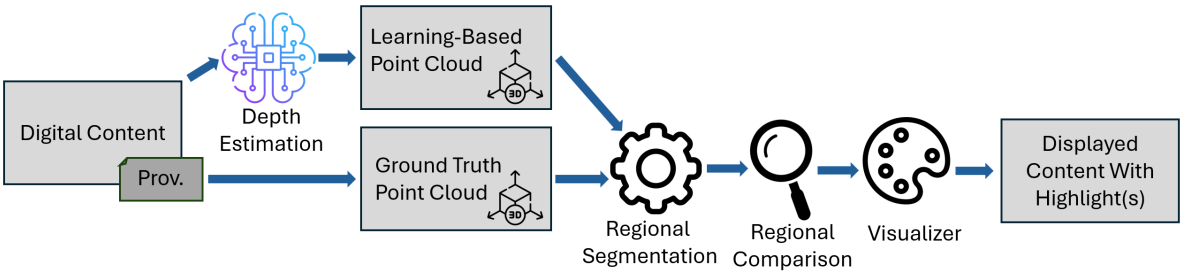


Figure 4.7: Scoop viewer's workflow.

4.5 Depth Estimation Preliminaries

We now summarize two main building blocks used in Scoop: ToF depth sensor and learning-based monocular depth estimation.

4.5.1 ToF Depth Sensor

ToF sensor, also known as ToF camera, is a special camera sensor used for measuring distance between the camera lens and the measured object. Similar to traditional camera sensor, it usually has multiple pixels, where each pixel can produce a distance value. Combining all pixels together, a depth map can be produced to recreate the 3D scene that the ToF sensor is seeing. ToF sensor can achieve high precision to the range of centimeters or even millimeters on some high-end models [61]. Also, because a ToF sensor works by detecting light it emits, it works well under low-light conditions.

There are mainly two types of ToF sensors: direct time-of-flight (dToF) sensors and indirect time-of-flight (iToF) sensors. A dToF sensor works by directly counting the time difference between the time that the sensor emits the light and the time that the sensor receives that light. Apple devices, such as iPhone 14-Pro, use a dToF (i.e., LiDAR) sensor [33]. An iToF sensor works by comparing the phase of emitted light and received light, where the phase difference is used to derive the distance value. Some Android devices, such as Samsung Galaxy S20 Plus, use an iToF sensor [23]. Due to the technical principle of a iToF sensor, it is more vulnerable to environmental lights and is also more likely to get confused when its

emitted light bounce around before coming back [87, 51, 26].

4.5.2 Learning-based Monocular Depth Estimation

Monocular depth estimation is the ability to determine distance to each pixel in a photo with only a single RGB image. With the rise of AR/VR and autonomous driving, it has become a hot research topic in computer vision community in recent years. Its task is usually carried out by a machine learning model, trained on a large number of datasets with various scenarios and their corresponding depth maps.

The task has also been classified into two categories: relative and absolute depth estimation. The former aims to give the relative depth relationship among multiple objects in a photo, while the latter provides a precise estimation of the distance value for each pixel in the photo. Recent advances in absolute depth estimation can achieve centimeter-level accuracy [65]. However, just like human beings, the learning-based approach sometimes make mistakes, especially on flat surfaces showing content.

4.6 Design

We design Scoop to be integrated with both existing camera pipelines and provenance-based media authentication systems [124, 123, 18, 25]. In Scoop, in addition to the ordinary RGB information, the camera also captures depth information with its ToF sensor and embeds it as part of the final captured content's provenance information. The embedded depth information is essentially another photo containing the same scene of the RGB photo, but it keeps an absolute metric depth value for each pixel instead of a RGB value.

Whenever the captured content (a photo/video) reaches the viewer side, it can be opened with any content viewer, but the embedded depth information is only utilized when the content is opened with a Scoop-compatible viewer. During content playback, Scoop reads the depth information and generates a ground truth point cloud. At the same time, it also

uses the corresponding RGB information to produce a depth map with a learning-based monocular depth estimation model. After that, the depth map is used to generate another point cloud. Scoop then compares these two point clouds against each other with Scoop comparison techniques. If any mismatch is found between the two point clouds, the content viewer highlights the mismatch to warn the users.

Note that Scoop only warns users of misleading (i.e., potentially malicious) recaptures by highlighting the part(s) of the frame flagged by its techniques. We made this design choice as we do not view Scoop as a determination tool of telling users if the digital content should be trusted or not, but as an analysis tool to assist users to make their own decisions. To better understand if this approach would work out or not, a future user study needs to be conducted. Another solution we have in our mind is to have a third party service running Scoop and make a decision, which helps offloading users' need to analyze Scoop's highlights. We also note that advanced reasoning AI could be a great fit to fulfill the need to analyze if Scoop's highlights may pose serious misinformation.

Figure 4.7 shows the entire workflow of Scoop viewer. We discuss more technical details of Scoop in §4.7.

Algorithm 1 Scoop's depth map comparison algorithm

Inputs: $\textbf{depth}_{learning}$, $\textbf{depth}_{truth}$
Output: **violated_pRegions**

Create $\textbf{pCloud}_{learning}$ with $\textbf{depth}_{learning}$
Create $\textbf{pCloud}_{truth}$ with $\textbf{depth}_{truth}$
Extract $\textbf{pRegions}_{truth}$ from $\textbf{pCloud}_{truth}$ using regional segmentation
Initialize **violated_pRegions**
for Every $\textbf{pRegion}_{truth}$ in $\textbf{pRegions}_{truth}$ **do**
    if Technique_1 (direct comparison) with $\textbf{pRegion}_{truth}$ and $\textbf{pCloud}_{learning}$ returns true **then**
        Add $\textbf{pRegion}_{truth}$ to **violated_pRegions**
    else
        Continue
    end if
    if Technique_2 (deviation correction) with $\textbf{pRegion}_{truth}$ and $\textbf{pCloud}_{learning}$ returns true **then**
        Add $\textbf{pRegion}_{truth}$ to **violated_pRegions**
    else
        Continue
    end if
    if Technique_3 (transformation) with $\textbf{pRegion}_{truth}$ and $\textbf{pCloud}_{learning}$ returns true **then**
        Add $\textbf{pRegion}_{truth}$ to **violated_pRegions**
    else
        Continue
    end if
end for
Return **violated_pRegions**

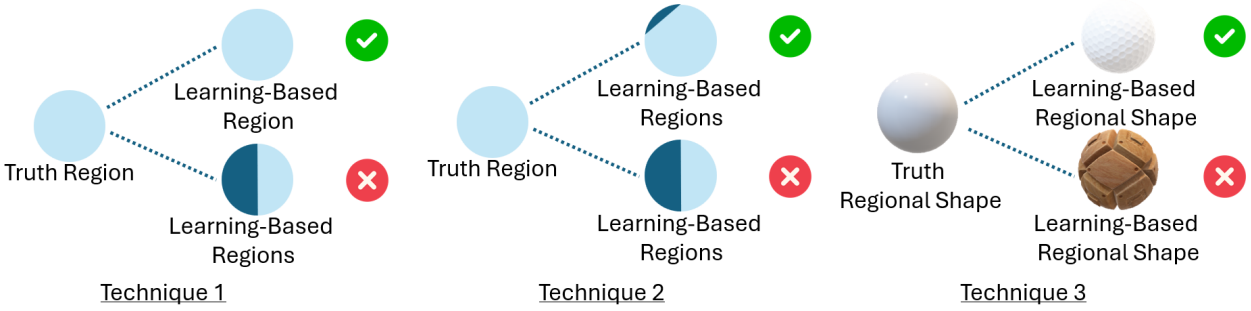


Figure 4.8: Scoop viewer's techniques. In each technique, we try to match the region extracted from the ground truth point cloud with its corresponding (pixel-level correspondence) region in the learning-based point cloud. Only if it cannot be matched, we move to the next technique, until we run out of techniques, where we would flag the region as suspicious.

4.7 Depth Maps Comparison

As stated in §4.6, when the digital content is opened with an Scoop-compatible viewer, Scoop generates two point clouds (for each photo or video frame): a ground truth point cloud based on the ToF sensor captured depth map and a learning-based point cloud based on the estimation model perceived depth map using RGB information. Scoop then compares these two point clouds by first performing regional segmentation on the ground truth point cloud, followed by correlating each of the ground truth regions with its pixel-level corresponding part in the learning-based point cloud using multiple techniques: direct comparison, deviation correction, and transformation. Algorithm 1 shows the algorithm used by Scoop, where details are explained in the below subsections.

4.7.1 Regional Segmentation

The technical goal of Scoop is to find mismatches between the two point clouds. Unfortunately, this is not as easy as calculating the difference between two point clouds. First of all, although ToF sensors can produce depth maps with high precision, their resolutions are usually relatively low, especially when compared with RGB camera sensors; additionally, noises are common in the captured depth map. On the other hand, learning-based point clouds can achieve high resolutions, but their precision is usually relatively low, especially

when compared with ToF sensors' captured depth maps. Even if we match their resolutions by scaling, their depth values are still drastically different from each other, and the difference changes across the same photo. In our observation, the learning-based point cloud usually represents the relative depth of a single object well in one photo, but when combined with other objects and backgrounds, its precision drops significantly.

To overcome these challenges, Scoop does not blindly compare absolute depth values across all pixels, but instead it compares by regions. After initial noise reduction is applied to both point clouds, Scoop uses a *region growing algorithm* to perform cluster segmentation on the ground truth point cloud. We choose a region growing algorithm due to the nature of surfaces, where it is hard to find any perfectly flat ones. Even if this work focuses on display mediums that are flat, our algorithm still needs to compute on depth information of non-flat surfaces and objects in the photo. We think the region growing algorithm is well suited for our use cases and can correctly separate all different surfaces existing in one scene. After the first segmentation, Scoop applies the same algorithm to the learning-based point cloud's corresponding regions (i.e., regions created with the previous segmentation on the ground truth point cloud) as well, which is detailed in §4.7.2

We note that parameters for performing cluster segmentation varies across depth maps for different capture/generation methods and scenes. For the former, difference in parameters is almost always fixed, where for the latter, parameters might need to be specially tuned for different objects. For example, human bodies tend to be continuous and could be recognized as a single surface with our default parameters, where stricter thresholds need to be set for correctly segmenting human bodies. In this case, learning-based image understanding models [44] could be used to first recognize different types of objects in the photo and then pick the best parameters for conducting cluster segmentation.

4.7.2 Regional Comparison

After performing cluster segmentation, we end up with many regions in the ground truth point cloud. For each of these regions, we compare it with its counter part in the learning-based point cloud, where the counter part is extracted by matching X and Y axes of all individual pixels in the region. We use multiple techniques to perform region comparison, where our goal here is to understand the similarity between them, and the reason we apply multiple of them is that there is no single technique that can cover all kinds of scenes, and there are also always imperfections in cluster segmentation, which could render some of them unusable. There are three techniques in Scoop (as illustrated in Figure 4.8): (1) direct comparison, (2) deviation correction and (3) transformation.

In Technique (1), we perform cluster segmentation again on the learning-based point cloud's part to check if we are only getting one region, which tells us that the learning-based point cloud agrees with the ground truth point cloud in this region. In Technique (2) (which is used when Technique (1) cannot be satisfied), we then try to relax the standard for checking. For example, we further examine if the biggest sub-region covers more than a certain threshold (e.g., 90%) percentage of the entire region (that is being checked). In Technique (3) (which is used when both Techniques (1) and (2) cannot be fulfilled), we use the iterative closest point (ICP) [42] and clustered viewpoint feature histogram (CVFH) [97] algorithms to perform the last attempt for getting the two regions matched with each other. The ICP algorithm tries to register one point cloud in another by transforming depth points, where once it succeeds, we check its convergence score to see if it falls below our threshold. On the other hand, the CVFH algorithm first create feature descriptors of geometric properties of two point clouds and then compare them. The reason we need CVFH in addition to ICP is that ICP is known to get stuck in local minima, which then it would falsely register two point clouds that are drastically different from each other. Therefore, we need a fail-safe to verify the convergence result given by ICP. As CVFH tends to be computationally heavy, we first calculate the mean squared error (MSE) difference between two point clouds, and CVFH

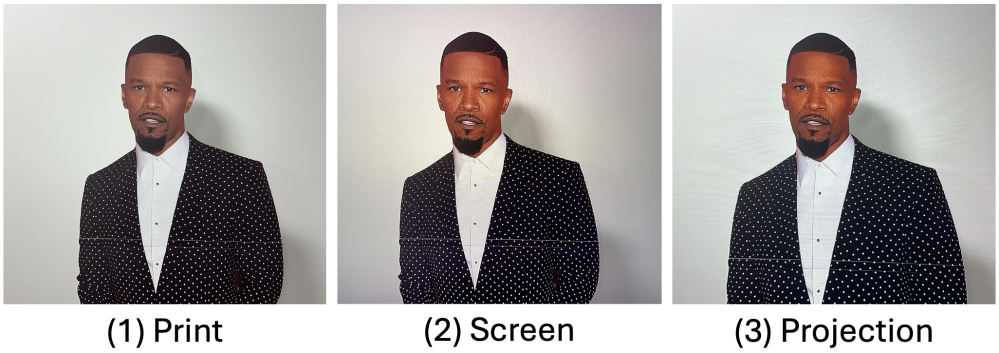


Figure 4.9: Example from our dataset showing different display mediums used.

is only used if their it exceeds a set threshold. The logic behind this is that a lower MSE difference represents higher similarity and better alignment of two point clouds, and they are both positive indicators for ICP.

4.8 Dataset

Even though the Internet contains many fake/recaptured photos and videos, they do not contain depth information making them unsuitable for evaluating Scoop. Moreover, although there are existing RGBD datasets, to the best of our knowledge, none of them is constructed with recaptures in mind. We found that existing RGBD datasets consists of various 3D objects but have little or no recapture scenes, meaning that they cannot fully assess the capability of systems like Scoop. Therefore, in order to properly evaluate Scoop, we collect an RGBD dataset containing photos and videos captured with recaptures in mind. Our dataset covers a wide range of recapture scenarios. More specifically, we cover all flat display mediums (listed below) as well as a diverse set of content to be shown on them. We note that our main focus is on TV-based *full-recapture* attacks, which we believe to have the most imminent threat, but we also collected data of other recaptures with different flat display mediums and *partial-recapture* scenes so that we can thoroughly evaluate the potential of Scoop. On the other hand, we did not include any potential recapture attack that is defined as out of scope in §4.3, e.g., a misleading recapture of a flat surface/object shown on a display

medium.

Display mediums. We use the following display mediums to collect our dataset:

1. Screen: a 85-inch Samsung The Frame LS03B LCD TV, which has a special coating for reducing reflection and tuned color and brightness mode for displaying photos like realistic arts, a 77-inch LG B4 OLED TV, and a 65-inch TCL R646 Mini-LED TV.

2. Projection: an Epson PowerLite L510U projector.

3. Printed: three life-size whole-body cardboard cutouts with different skin colors (two of them are celebrities and another one is an author of this work) and a life-size celebrity face cutout.

Figure 4.9 demonstrates three example photos for each of the three options in the flat surface category: Figure 4.9(1) is a life-size cardboard cutout of a celebrity; Figure 4.9(2) is Figure 4.9(1) displayed on a TV; Figure 4.9(3) is Figure 4.9(1) displayed on a projector. Any of the three options may potentially fit into a malicious scenario; for example, they could be used to claim the presence of that celebrity at a certain location using the GPS location embedded in the provenance information [25] of those photos.

Our dataset also covers benign cases where display mediums are present, but their presence can be clearly recognized by the content viewer, as mentioned in §4.2.1 and shown in Figure 4.4.

Content. We collect some photos/videos of real environments to be used in the dataset. These contents are captured by a smartphone camera and are included in the dataset as benign scenarios. We also use the same photos/videos in the recapture scenarios. That is, we show them on our display mediums and capture photos/videos of the mediums in a way that it is not obvious that they are being displayed. For our dataset, we assign the ground truth to each photo/video to be either benign content or misleading recapture.

To thoroughly evaluate the effectiveness of our techniques, we try to collect a diverse content set. To do so, we use a total of three different categorization methods when collecting the

content: background, object, and lighting.

Our categories for "background" include:

1. Plain (i.e., very simple color and/or depth distributions, such as white walls)
2. Textured (i.e., relatively simple color and/or depth distributions, such as patterned mats)
3. Complex (i.e., complex color and/or depth distributions, such as a shelf fully stacked with items)

Our categories for "object" include:

1. None (i.e., no specific object other than the background)
2. Item (i.e., item(s) is shown as the main object; the item could be small (e.g., a water bottle) or large (e.g., a TV))
3. Human/Animal (i.e., living being such as a person)
4. Mixed (i.e., mix of the above two)

Our categories for "lighting" include (note that all options can be either indoor or outdoor):

1. Very dark (i.e., from no light to the captured footage being barely visible)
2. Dark (i.e., most parts of the captured footage are visible, but only little light is presented)
3. Lit (i.e., the captured footage is clearly visible, but shadows can be easily found)
4. Well lit (i.e., the captured footage has almost no shadow)

Please refer to Appendix A.3 for sample photos of different categories. We mix and match almost all options of the above three categories to the best of our effort in order to have the content set contain scenes as versatile as possible. We note that in some scenes, the

object category's option could completely cover the background category's option, making the latter invisible.

4.8.1 Collection of Our Dataset

We first mix and match the normal three categories to capture different scenes, where we have multiple different settings for each type of scene, and there is one photo and one 10-second video clip for the data point of each setting. Followed by that, we then replay them using options in various display mediums. This results in a total of 488 data points collected in the final dataset, comprising 122 unique ones captured as photos and videos and on two different smartphones. We made use of two smartphones in order to learn the difference between different types of depth sensors (dToF vs. iToF); more specifically, we used two smartphones (Apple iPhone 14 Pro and Samsung Galaxy S20 Plus to capture each data point. More technical details about our dataset such as RGB and depth resolutions can be found in §4.9.

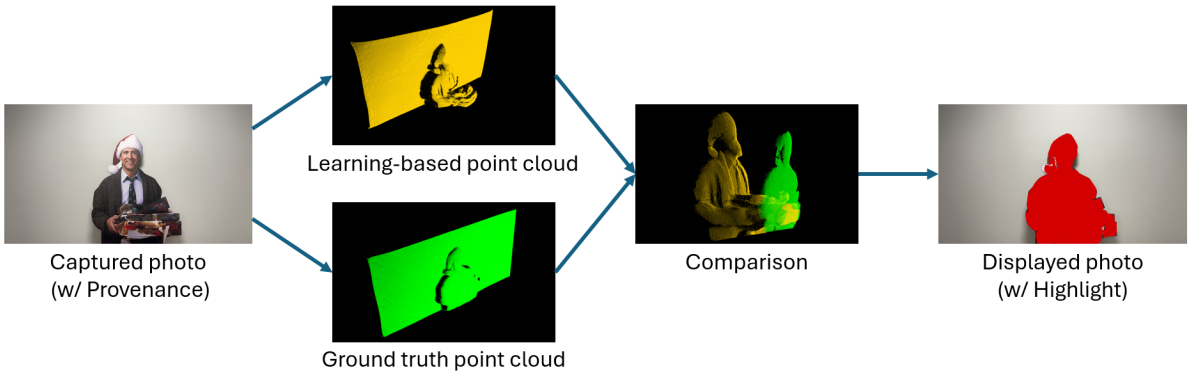


Figure 4.10: Demonstration of Scoop viewer prototype's workflow with a sample photo containing a life size cardboard cutout of a celebrity.

4.9 Prototype & Evaluation

We have implemented a complete prototype of Scoop. The producer (camera) side is implemented on both iOS and Android platforms. On the iOS platform, Scoop captures depth

frames with a resolution of 768×432 during photo capture and 320×180 at 30 frames per second during video capture. On the Android platform, Scoop captures depth frames with a resolution of 310×205 during both photo capture and same resolution at 20 frames per second during video capture. On both producer platforms, their captured RGB photo has a resolution of 3840×2160 with JPEG codec, and same resolution for RGB video at 30 frames per second with H.265 codec. The iOS platform is evaluated on an Apple iPhone 14 Pro running iOS 18.2, and the Android platform is evaluated on a Samsung Galaxy S20 Plus running Android 13. It is noteworthy mentioning that the dToF sensor on the iPhone is pre-calibrated with the primary RGB camera sensor, where the iToF sensor on the Samsung phone is not. Such calibration is crucial for Scoop to work properly, as we rely on the fact that both depth and RGB camera sensors capture the exact same content. Although we manually calibrated the Samsung phone's iToF sensor with its primary RGB camera sensor, the lack of sensors' factory information make it hard for our calibration result to match with the iPhone. We used our producer prototype on both phones to capture our dataset, and iOS captured footage is used for playback on display mediums due to its higher picture quality.

The consumer (viewer) side has two components: generator and analyzer, where generator is responsible for producing the RGB frame based depth map using monocular depth estimation learning-based model and analyzer contains the algorithms for comparing the two depth maps for each RGB frame. The generator uses a state-of-the-art model: depth-pro [65], which utilizes PyTorch 2.5.1 [136] with CUDA 12.4 [2]. The analyzer is implemented with OpenCV 4.x (11-02-2024) [69] and PCL (10-27-2024) [145] libraries using C++. Both generator and analyzer run on top of Ubuntu 24.04.

The consumer uses a single CPU core and a GPU. We use the Intel Xeon Gold 6438M CPU (which has a lower single-thread performance than desktop CPUs) and an Nvidia RTX-4090 GPU, which is a consumer-grade GPU. Admittedly, this is a powerful GPU. We have not evaluated the performance of the consumer on weaker GPUs. But we note that

Scoop's consumer could be performed off-line (e.g., in a server), the result of which could be simply appended as metadata to a photo/video (which then could be easily verified in any consumer device). We leave it to future work to optimize Scoop's consumer to run with good performance in various consumer devices.

Figure 4.10 show-cases Scoop viewer's prototype. We can clearly see that some facial features, details of clothes, hands, and gift boxes have depth information registered in the learning-based point cloud, which contradicts with the ground truth point cloud. In this case, we simply highlight all the violated parts with red color in the final displayed photo.

4.9.1 Effectiveness

The prototype of Scoop viewer and its parameters are developed using some test data we collected in our daily lives. Similar to the dataset we have, those test data covers a wide range of camera use cases, including capturing benign scenes and display mediums. We note that this set of test data is completely disjoint from our evaluation dataset. We then evaluate it on the dataset we collected, for both true positive rate (TPR) and false positive rate (FPR). True positive means successfully classifying the existing of a misleading recapture, where false positive means falsely classifying a benign scenario as a misleading recapture. Figure 4.11 shows the results on different display mediums; Figures 4.12(a-c) show the results on different benign content categories; and Figure 4.13 shows the overall results. In Figure 4.11, each of the first four x-axis options means the screen is captured without its frame being visible in photos, making them fall into *full-recapture* scenarios. The cutouts option represents a cutout put in front of either plain or textured background without any other objects, which is counted as a part of *partial-recapture* scenarios. Lastly, the mixed option indicates the photos contain either multiple cutouts or display mediums blended with real 3D object(s) , which also falls into *partial-recapture* attacks. In summary, the iPhone 14 Pro (w/ dToF) based prototype achieves exceptional overall results with 94.81% of TPR and only 0.02% of FPR; The Galaxy S20 Plus (w/ iToF) based prototype achieves good overall results as well

with 74.03% of TPR and 17.78% of FPR.

One observation is that the Android prototype struggles in complex scenes; for instance, the mixed option in Figure 4.11 points out that the Android prototype cannot even reach half the TPR as of what the iOS prototype is capable of achieving. Another observation about the Android prototype is it seems to be more sensitive to reflective surfaces; for example, the LG TV's screen is more reflective than other display mediums, and even if our data on that TV was captured in a very dark room, as shown in Figure 4.11, the Android prototype still does not perform well. We then further discovered that whenever we have scenes allowing lights to be bounced back and forth in different paths and distances, the Android prototype seems to perform worse. Such observation aligns with how the industry reviews dToF and iToF sensors: that dToF sensors generally have greater capability under complex lighting situation and long distance measurement and iToF sensors tend to suffer more from noise (e.g., multi-path interference) [87, 51, 26]. In addition to the fundamental technical difference, proper optimizations of hardware, firmware, and software (i.e., algorithms) matter as well. And our observation is that iPhone's dToF sensor is better calibrated.

Comparison with human's performance. In the user study we conducted, the participants have achieved 44.19% of TPR and 56.1% of FPR. In comparison, for the same set of photos, our iOS prototype achieves 100% of TPR and 12.5% of FPR, where the Android prototype achieves 87.5% of TPR and 12.5% of FPR. We believe that Scoop is more capable in terms of recognizing misleading recaptures than human beings.

To learn about the effect of different depth resolution, we have also conducted another round of experiments on the same iOS captured content, but with downscaled depth resolution (320×180), which roughly matches with the Android counterpart. As seen in Figure 4.13, at the lower depth resolution, iOS prototype still manages to retain a great TPR result of 90.91%, but suffers from a worsen FPR of 17.78%. We noticed that in benign scenes with lower depth resolutions, whenever most of the captured scene is filled with flat surfaces, it is more likely that the ground truth depth map would disagree with the learning-based depth

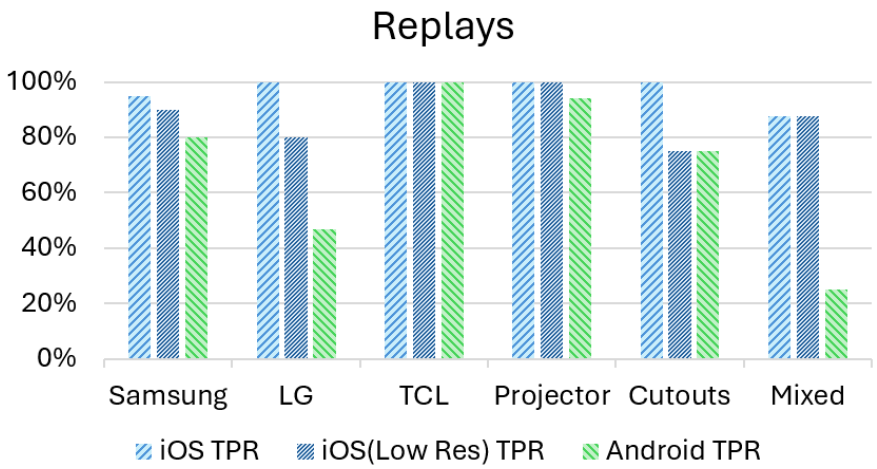


Figure 4.11: Results of Scoop's prototype under different display mediums.

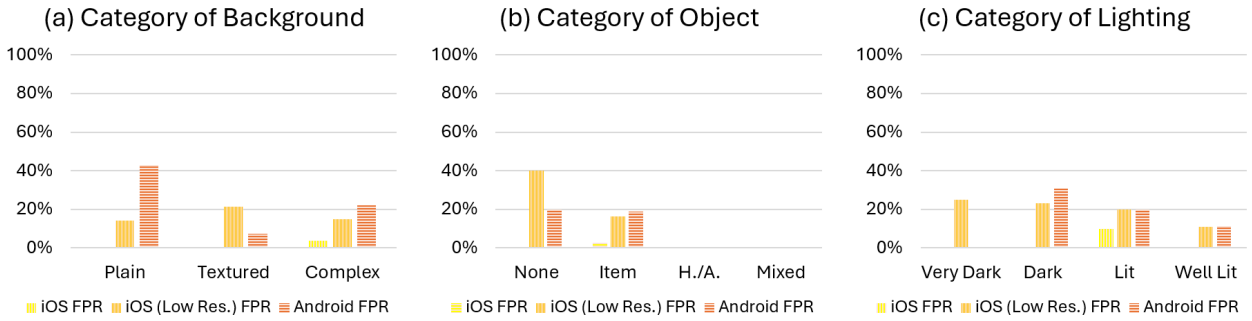


Figure 4.12: Results of Scoop's prototype under different benign content categories.

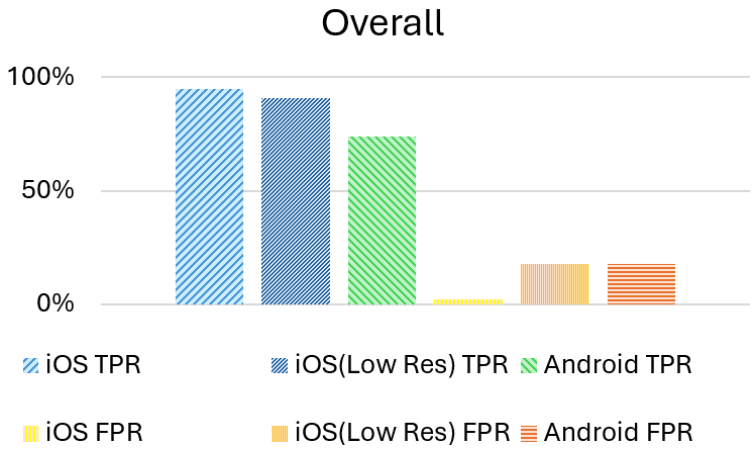


Figure 4.13: Overall effectiveness of Scoop's prototype.

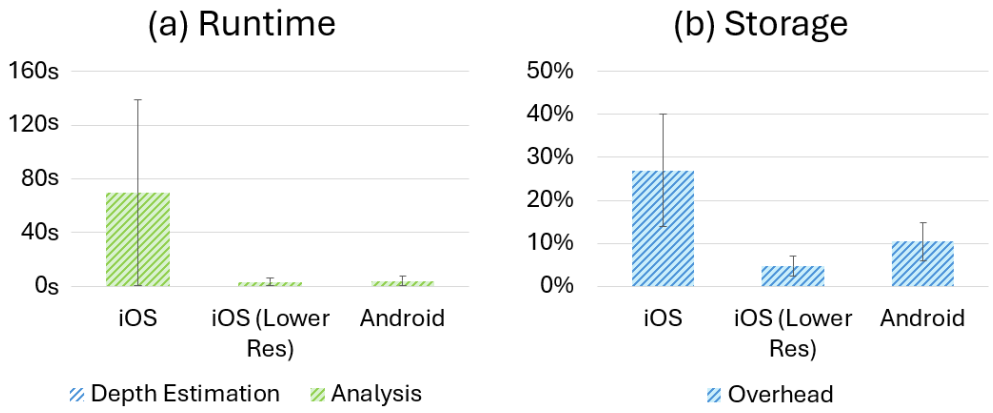


Figure 4.14: Both performance and storage overheads of Scoop's prototype.

map as learning-based depth maps tend to contain more subtle details in these scenes, and this is proven by the relatively high false positive rates of both the iOS prototype with lower depth resolution and the Android prototype on None and Item options in Figure 4.12(b).

4.9.2 Overheads

Performance. Scoop introduces computational overhead in three main places: camera capture, learning-based depth estimation, and viewer analysis. At camera capture time, although additional computation is performed, no human noticeable runtime overhead has been observed in both photo and video capturing. As Figure 4.14(a) shows, depth estimation takes on average of 0.28 ±0.02 second for the iOS prototype captured photo (no matter what the depth resolution is) and 0.26 ±0.01 second for the Android prototype captured photo; Scoop viewer's analysis, on the other hand, introduces the most runtime overhead: an average of 69.38 ±95.17 seconds for the iOS prototype, 2.96 ±3.48 seconds for the iOS prototype with lower depth resolution, and 3.68 ±2.99 seconds for the Android prototype. Note that the analysis duration varies significantly due to difference in scene complexity and usage of our techniques, resulting in large standard deviations. While this statistical variation produces negative values, all actual measured durations in our experiments were positive. The reason our depth estimation runs much faster is it uses well-optimized software (i.e., PyTorch with CUDA support) and powerful hardware (i.e., RTX 4090); in contrast, our

viewer analysis runs slow, as it is currently not optimized for performance. We also note that standard deviation of analysis' runtime for both platforms are large. This is due to our techniques (§4.7.2), where sometimes running only one technique is enough, and other times all techniques are needed. We can also observe that the content captured by the iOS platform has worse runtime overhead than the Android prototype. This is caused by both the fact that the iOS prototype capturing depth maps with higher resolution than the Android counterpart and our picks of point cloud algorithms where some of them have runtime complexities worse than linear.

An alternative approach to boost performance is to perform frame-level comparison instead of region-level comparison. As only one comparison needs to be done in this way, the overhead can be significantly reduced and such light version of Scoop is still capable of detecting misleading *full-recaptures*, but not *partial-recaptures*.

Storage. Figure 4.14(b) shows the percentage of extra storage space Scoop takes for each photo. The overhead on the iOS prototype is 648 KB (about 26.91 ±13.09%) for higher depth resolution and 112.5 KB (about 4.69 ±2.27%) for lower depth resolution. The overhead on the Android prototype is 125 KB (about 10.4 ±4.47%). As can be seen, the depth map size stays unchanged across photos, whereas photo sizes vary depending on the complexity of scenes (which determines how much compression can be done) and compression ratios determined by different software platforms (iOS v.s. Android).

Energy consumption. We use the Samsung Galaxy S20 Plus for this experiment, as there is no direct way to get detailed power consumption data on the iOS platform. By leveraging Android *Power Profiler* [37], we managed to measure the energy consumption of both capturing a RGB photo and a 10-second video using Scoop prototype camera. We also introduce a baseline application that functions the same but do not capture depth information. During photo capture, the baseline consumes energy at a rate of 1986.67 ±48.17 microamps and the Scoop prototype consumes energy at a rate of 2793 ±98.85 microamps. As it takes about 0.09 ±0.01 second to capture a RGB photo and 0.1 ±0.01 second to capture

a RGB photo and a depth map, our prototype captures photos with an energy consumption overhead of 56.2%. During video capture, the baseline consumes energy at a rate of 1764.1 ±152.08 microamps and the Scoop prototype consumes energy at a rate of 2435.7 ±193.04 microamps. Since there is no difference in capture times for both of them (10s for videos), we conclude that our prototype incurs an energy consumption overhead of 38.07% during video capture.

4.10 Additional Considerations

4.10.1 3D Display-Based Attacks

In Scoop, we consider attacks using 3D display mediums to be out of scope. For these attacks, we assume custom-shaped (e.g., a 3D-printed model) objects are used along with display mediums (e.g., 3D projection). To some extent, we do not think our system would fail easily against naive versions of such an attack. As explained in 4.7, Scoop works by applying regional analysis based on absolute depth information, so in order to spoof Scoop, the 3D objects need to be refined enough to surpass our comparison thresholds. Furthermore, if captured objects are not made in real life-size (e.g., miniatures), Scoop can also utilize triangulation to figure out the difference in sizes between the ground truth depth map and the learning-based depth map. While we do not perform any direct size comparison in our current prototype, we note that Scoop's Technique (3) (as mentioned in §4.7.2) is likely able to handle this due to the use of rigid transformation without any uniform scaling, where a large size difference will cause the convergence score to exceed our set threshold. In other words, for such an attack to succeed, it has to be mounted with life-size refined custom shapes along with high quality display mediums. That said, we opt to not claim any capability against 3D display-based attacks without a comprehensive evaluation, which will be a future work of Scoop. Moreover, we believe that flat display mediums are the immediate threat to provenance-based media authentication, and this work's main contribution is about

tackling flat display-based recapture attacks.

4.10.2 Binocular Vision

In this work, we explicitly choose to make use of ToF-based depth sensors, as they are widely used in the smartphone industry. We considered binocular vision technology [38], which works similarly to human eyes by making use of two camera sensors in a device to measure the absolute distance between the device and captured objects. However, such devices are rare and relatively new [40, 41]. It is unclear how well they work for distance estimation, and particularly estimating the relative distance between objects and their depth. It is possible that their use, in addition to ToF, could make Scoop more robust.

4.10.3 Forward Looking Infrared (FLIR)

Some recent phones, such as the Google Pixel 9, are equipped with a sensor [47] that can measure the temperature of distant surfaces. Some prior work uses temperature to generate depth maps [146, 30], which works exceptionally well on scenes containing human bodies, and we acknowledge that this is still an ongoing research topic. However, they are mostly limited to known objects with relatively fixed ranges of temperature - knowingly human beings. To make matters worse, such sensors on existing smartphones have limited resolution for distant objects. There is still potential light in this path though, military-grade FLIR cameras can produce 720p resolution heat maps for up to 10 kilometers or 6 miles [45]. We hope that over time, the price point and energy consumption of those sensors can be brought down to meet the requirements to be included in smartphones.

4.10.4 Videos

As videos are essentially bundles of photos (frames) being play-backed in sequence, Scoop supports videos out of the box. However, its support of videos is also very preliminary and

can be vastly improved. Here we list three points that we believe can be used to enhance Scoop for videos, but we do note that more can be done. First, consecutive frames in videos tend to contain continuous motion, especially in videos captured with a camera; with the support of either software algorithms and/or hardware such as gyroscope and accelerometer, point clouds can be built across multiple frames for higher accuracy. Second, as frames are being play-backed at high speed, where each frame has limited visible time, assigning suspiciousness on a frame-by-frame based is most likely not needed. Third, similar to the second point, depth maps may not need to be captured on a per-frame based.

Although various enhancements can be applied to videos, it is noteworthy to mention that the adversary may also take advantage of unique features of videos. For example, they may move the camera and display medium freely at capture time to craft an illusion of movement. Another dangerous factor is that videos have more post-processing potentials. Therefore, we believe that video-specific enhancements should be considered thoroughly before being applied to Scoop.

4.10.5 Post-Processing

The captured content may need further post-processing, such as cropping, coloring, or applying other filters. Similar to the provenance information proposed by Vronicle [123], the embedded depth information needs to get updated along with its corresponding RGB photo. This does not mean that the depth information can be freely updated whenever a change is applied to its RGB counterpart, as this could easily open up an attack vector for the adversary. Instead, the depth information can only be applied with subtraction-based update. This means that every time an update is being applied, the update must be a removal of depth information. For example, when the RGB photo is cropped, its depth information is also cropped with the same scale. Another example would be when a person's face is blurred in the RGB photo, the face's depth information is also removed (by setting 0 to all face-related pixels). On the other hand, if an object is artificially added to the RGB photo,

no depth of that object is applied to the depth information of the original photo; instead, all pixels covered by the newly added object would have their depth counterparts zeroed out, effectively turning the covered area into a flat surface with no depth value, and this can be picked up by Scoop viewer. During post-processing, all updates to the depth information are done automatically according to how the corresponding RGB pixels are modified and users (editors) are not allowed to directly make any change to the depth information, where all of these are enforced by provenance-based secure post-processing systems such as Vronicle [123].

4.11 Conclusions

Visual content is an essential form of information consumption by humans. A variety of consequential news is obtained in this medium, including presidential addresses, parliamentary debate, police action, and war footage. We now increasingly rely on such content for automation of critical tasks, including driving, manufacturing, and border control. Provenance-based techniques from both academia and industry protect the pipeline from camera sensor to consumption display, but fail to protect us from recapture attacks in front of the camera sensor. We presented Scoop, a solution designed to protect us from an adversary launching recapture attacks, where the physical scene is manipulated with digital screens or posters. We showed that it achieves high accuracy in finding misleading recaptures, which it presents to viewers to help them detect attacks. We hope our work will inspire adoption of ToF sensors by more cameras, such as security cameras. We expect that the cost, range, and resolution of ToF sensors will improve as adoption increases. We expect that with increasing use of multi-modal AI, even more sophisticated techniques will be built to interpret the depth information collected by Scoop. All these trends will contribute to making Scoop more powerful against such attackers.

Chapter 5

Related Work

5.1 Secure Media Attestation

5.1.1 Secure Video Provenance

Several video provenance methods have been proposed in the past [90, 55, 151, 22, 83, 99], including watermarking, hash chaining, use of digital signatures, and use of blockchain. Table 2.2 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 [131] 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 [91] analyzes a photo to understand where it was edited.

Some works extend photo provenance to provide provenance for a video. One example is Aletheia [22], 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 Frame-

Prov [55], 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 [99, 16, 20]. All of these systems suffer from high latency because of their dependency on blockchain. AMP [83] utilizes the Confidential Computing Framework [17] 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 [8] to prove image originality. However, later on it was proven that there was a flaw in the system [9]. More recently, startup companies such as Truepic [20] and Serelay [13] 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 [12]. 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) [18] 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.

5.1.2 Secure Camera and Sensors

Some works utilize Trusted Platform Module (TPM) to implement secure cameras. TrustCAM [173] uses a non-migratable key protected by a TPM on a surveillance camera to sign

frames before they are streamed. TrustEYE.M4 [172] 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 [147, 121]. Their work cannot, however, provide fine-grained provenance for videos, especially when offloaded.

5.1.3 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 [164] (which is a proprietary emulator) and Telekine [104]. Visor [138] 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.

5.1.4 Motion-based Video Timestamps Attestation

C-14 [158] 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.

5.2 Securing Sensing and I/O

5.2.1 Replay-based I/O

Some works also secure I/O using record-and-replay. Both RT-TEE [168] and driverlets [95] use TrustZone to run their debloated replay-based drivers for controlling sensors, where

RT-TEE aims to provide flexibility close to the ordinary drivers, allowing parameters to be dynamically configured into presets. GPUReplay [134] is also a middleman running in a secure world such as TrustZone that replays presets of commands, and it is built for securely controlling GPU to perform machine learning workloads. It aims to replace the massive GPU stack existing in typical ML work and can handle more complex jobs for GPU. These three works, however, can only support either simple sensors or a specific hardware device; they cannot secure the video capture process, which has a complex processing pipeline that needs seamless coordination with other hardware components.

5.3 Media Content Authentication

5.3.1 Display Screen Recapture Detection

There has been quite some research effort put on detecting recaptured images that were displayed on a screen, where most of them have been focusing on detection of LCD screen displayed images [106, 161, 71]. They rely on the unique moiré pattern [43] that could be detected on LCD displays. There is also work that further extends this technique to OLED display [163]. Most of those works utilize machine learning to help recognizing the almost invisible unique patterns of each display, where the models need to be trained on the exact display model's panel. Given the fast development of display technology, rapid release, and extensive amount of existing displays in the market, it is hard to come up with a model that is capable of detecting all display models. Furthermore, there are more kinds of display methods that can be used to display arbitrary photos and videos, such as projecting, printing, and so on.

5.3.2 Face anti-spoofing

There are many previous works that focus specifically on face recapture detection [117, 122, 85, 36]. This is due to the fact that face is widely used as an authentication method

nowadays, where deceiving the facial authentication system may bring enormous benefits to the adversary. Some commercial solutions [50, 32] already exist. They use additional sensors to help detecting displayed images by either filtering unwanted display surfaces (e.g., printed photos or display screens) or reconstructing 3D model of the user's face. There is also research work [85] done for attempting to reconstruct user's 3D face model without the use of additional sensor. Other works [117, 122] utilize facial feature extraction and analysis with machine learning model to figure out the difference between real faces and displayed faces. Another line of work [36] is to conduct active liveness detection, which requires the user to perform randomly assigned actions in front of the camera. Although all these above works are reasonably powerful, they are highly limited to only one specific use case - face authentication, where Scoop aims to be a more generic approach for both photos and videos.

5.3.3 Object anti-spoofing

Serval previous works [76, 110, 153, 118] have considered object anti-spoofing on top of face anti-spoofing, where most of them have all opted for machine learning based approaches mToFNet [110] also makes use of the ToF sensor presented on the camera, but only as an auxiliary input to the model. Additionally, these solutions are limited to either the type of displays that their models are trained on or the kind of objects that their models are optimized for.

Bibliography

[1] 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.

[2] NVIDIA CUDA. http://www.nvidia.com/object/cuda_home_new.html.

[3] Truepic Breakthrough Charts a Path for Restoring Trust in Photos and Videos at Internet Scale. https://www.prnewswire.com/news-releases/truepic-breakthrough-charts-a-path-for-restoring-trust-in-photos-and-videos-at-internet-scale-301152998.html.

[4] VLC. https://www.videolan.org/vlc/, 2001.

[5] QuestionPro: Online Survey Software and Tools. https://www.questionpro.com/, 2002.

[6] 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, 2004.

[7] TrustZone: Integrated Hardware and Software Security: Enabling Trusted Computing in Embedded Systems. In *ARM White Paper*, 2004.

[8] Canon Original Data Security Kit. http://www.canon.co.jp/imaging/osk/osk-e3/index.html, 2010.

[9] Canon Original Data Security System Compromised: ElcomSoft Discovers Vulnerability. https://www.elcomsoft.com/PR/canon_101130_en.pdf, 2010.

[10] Profile battery usage with Batterystats and Battery Historian. https://developer.android.com/topic/performance/power/setup-battery-historian, 2015.

[11] Protect against security threats with SafetyNet. https://developer.android.com/training/safetynet, 2017.

[12] Security certifications for the Secure Enclave Processor. https://support.apple.com/en-gb/guide/sccc/sccca7433eb89/web, 2017.

[13] Serelay: Trusted Media Capture. https://www.serelay.com/, 2017.

[14] Android NDK. https://developer.android.com/ndk/index.html, 2018.

[15] Open Portable Trusted Execution Environment (OP-TEE). https://www.op-tee.org/, 2018.

[16] Amber: Instilling trust into video. https://app.ambervideo.co/, 2019.

[17] CCF: A framework for building confidential verifiable replicated services. https://github.com/microsoft/CCF/blob/master/CCF-TECHNICAL-REPORT.pdf, 2019.

[18] Content Authenticity Initiative. https://contentauthenticity.org/, 2019.

[19] Google SafetyNet Attestation API. https://developer.android.com/training/safetynet/attestation, 2019.

[20] Truepic: Photo and Video Verification You Can Trust. https://truepic.com/, 2019.

[21] Votz. https://voatz.com/, 2019.

[22] Aletheia: Fight Fake News by Signing Your Files. https://danielquinn.github.io/aletheia/, 2020.

[23] Check out the new camera functions of the Galaxy S20 | S20+ | S20 Ultra. https://www.samsung.com/uk/support/mobile-devices/check-out-the-new-camera-functions-of-the-galaxy-s20-plus-s20-ultra, 2020.

[24] 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--, 2020.

[25] Coalition for Content Provenance and Authenticity. https://c2pa.org/, 2021.

[26] Comparison between dToF and iToF sensors. https://faster-than-light.net/TOFSystem_C4, 2021.

[27] CVE Details. Op-tee: Vulnerability Statistics. https://www.cvedetails.com/product/56969/Linaro-Op-tee.html, https://www.cvedetails.com/product/42749/Linaro-Op-tee.html, https://www.cvedetails.com/product/36161/Op-tee-Op-tee-Os.html, 2021.

[28] Does Your Package Size Affect Security. https://cerberus-laboratories.com/blog/package_size_matters/, 2021.

[29] 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, 2021.

[30] Heat Map Spectrum to Grayscale (Depth Maps). https://github.com/ImageMagick/ImageMagick/discussions/3126, 2021.

[31] 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, 2021.

[32] Windows Hello face authentication. https://learn.microsoft.com/en-us/windows-hardware/design/device-experiences/windows-hello-face-authentication, 2021.

[33] iPhone 14 Pro. https://support.apple.com/guide/iphone/iphone-14-pro-iph6928b4ea3/ios, 2022.

[34] AMBA AXI and ACE Protocol Specification Version E. https://developer.arm.com/documentation/ihi0022/e/AMBA-AXI3-and-AXI4-Protocol-Specification, 2023.

[35] Color filter array - Glossary. https://www.digitizationguidelines.gov/term.php?term=colorfilterarray, 2023.

[36] FaceRecognition-LivenessDetection-Android. https://github.com/MiniAiLive/FaceRecognition-LivenessDetection-Android, 2023.

[37] Power Profiler. https://developer.android.com/studio/profile/power-profiler, 2023.

[38] Stereo Vision and Depth Perception in Computer Vision. https://sumitkrsharma-ai.medium.com/stereo-vision-and-depth-perception-in-computer-vision-8aff8dd04e7c, 2023.

[39] Adobe Content Authenticity. https://contentauthenticity.adobe.com, 2024.

[40] Apple Vision Pro gets extraordinary $30,000 dual-lens camera that can handle almost 118 million pixels. https://www.techradar.com/pro/apple-vision-pro-gets-extraordinary-usd30-000-dual-lens-camera-that-can-handle-almost-118-million-pixels, 2024.

[41] Canon Now Accepting Orders for Spatial Video Lens Previewed at WWDC. https://www.macrumors.com/2024/11/03/canon-spatial-video-lens-pre-orders/, 2024.

[42] Iterative Closest Point Algorithm. https://cs.gmu.edu/~kosecka/cs685/cs685-icp.pdf, 2024.

[43] moiré pattern. https://www.britannica.com/science/moire-pattern, 2024.

[44] Moondream AI. https://moondream.ai/, 2024.

[45] Ranger® HDC. https://www.flir.eu/products/ranger-hdc/, 2024.

[46] Sora. https://openai.com/index/sora/, 2024.

[47] Take the temperature of objects with your Pixel phone. https://support.google.com/pixelphone/answer/14103759, 2024.

[48] Time-of-Flight principle. https://www.terabee.com/time-of-flight-principle, 2024.

[49] Trust what you see: How Truepic authenticates images and videos in the age of deepfakes. https://www.qualcomm.com/news/onq/2024/11/trust-what-you-see-how-truepic-authenticates-images-and-videos, 2024.

[50] Use Face ID on your iPhone or iPad Pro. https://support.apple.com/en-us/108411, 2024.

[51] What is LiDAR? https://www.symphotony.com/lidar/principle/, 2024.

[52] Zenmuse L2 - DJI. https://enterprise.dji.com/zenmuse-l2, 2024.

[53] Gemini AI video generation. https://gemini.google/overview/video-generation, 2025.

[54] Adobe. 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, 2021.

[55] M. Ahmed-Rengers. FrameProv: Towards End-to-End Video Provenance. In *Proc. ACM New Security Paradigms Workshop (NSPW)*, 2019.

[56] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for CPU based attestation and sealing. In *Proc. ACM International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[57] Apple. Importance of Service by Trained Technicians Using Genuine Apple Cameras. https://support.apple.com/en-us/HT212002#:~:text=With%20iPhone%2012%20models%20and,Apple%20Part%22%20next%20to%20Camera., 2022.

[58] A. D. Article. Capturing Photos with Depth. https://developer.apple.com/documentation/avfoundation/cameras_and_media_capture/capturing_photos_with_depth, 2017.

[59] N. Asokan and A. Paverd. Remote Attestation. Building trust in things you can't see. Tutorial, 2021.

[60] P. Athe and S. Dasgupta. A Comparative Study of 6T, 8T and 9T Decanano Sram Cell. In *IEEE Symposium on Industrial Electronics and Applications*, 2009.

[61] AVSystem. ToF: Time-of-Flight - Overview, Principles, Advantages. https://avsystem.com/blog/linkyfi/time-of-flight, 2024.

[62] S. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller. Zoedepth: Zero-shot transfer by combining relative and metric depth. *arXiv preprint arXiv:2302.12288*, 2023.

[63] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A. Sadeghi. The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel SGX. In *Proc. USENIX Security Symposium*, 2018.

[64] R. Birkl, D. Wofk, and M. Müller. Midas v3.1 – a model zoo for robust monocular relative depth estimation. *arXiv preprint arXiv:2307.14460*, 2023.

[65] A. Bochkovskii, A. Delaunoy, H. Germain, M. Santos, Y. Zhou, S. Richter, and V. Koltun. Depth pro: Sharp monocular metric depth in less than a second. *arXiv*, 2024.

[66] J. Boone. TPM Genie: Interposer Attacks Against the Trusted Platform Module Serial Bus. https://research.nccgroup.com/2018/03/09/tpm-genie-interposer-attacks-against-the-trusted-platform-module-serial-bus/, 2018.

[67] S. Borel, L. Duperrex, E. Deschaseaux, J. Charbonnier, J. Cledière, R. Wacquez, J. Fournier, J. C. Souriau, G. Simon, and A. Merle. A Novel Structure for Backside Protection against Physical Attacks on Secure Chips or Sip. In *2018 IEEE 68th Electronic Components and Technology Conference (ECTC)*, pages 515–520. IEEE, 2018.

[68] Z. Boulkenafet, J. Komulainen, L. Li, X. Feng, and A. Hadid. Oulu-npu: A mobile face presentation attack database with real-world variations. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, 2017.

[69] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[70] E. Brickell, J. Camenisch, and L. Chen. Direct Anonymous Attestation. CCS '04, 2004.

[71] H. Cao and A. Kot. Identification of recaptured photographs on lcd screens. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2010.

[72] P. R. Center. How Americans Get News on TikTok, X, Facebook and Instagram. HowAmericansGetNewsonTikTok,X,FacebookandInstagram, 2024.

[73] P. R. Center. Social Media and News Fact Sheet. https://www.pewresearch.org/journalism/fact-sheet/social-media-and-news-fact-sheet/, 2024.

[74] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto. SoK: Understanding the Prevailing Security Vulnerabilities in Trustzone-assisted TEE Systems. In *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2020.

[75] T. Cloosters, M. Rodler, and L. Davi. TeeRex: Discovery and Exploitation of Memory Corruption Vulnerabilities in SGX Enclaves. In *Proc. USENIX Security Symposium*, 2020.

[76] V. Costa, A. Sousa, and A. Reis. Image-based object spoofing detection. In *Combinatorial Image Analysis*, 2018.

[77] V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[78] V. Costan, I. Lebedev, and S. Devadas. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *Proc. USENIX Security Symposium*, 2016.

[79] R. Cotta. Snapdragon 8 Gen 2 vs A16 Bionic Which Is the Better Chip? https://www.videogamer.com/tech/smartphone/snapdragon-8-gen-2-vs-a16-bionic/, 2023.

[80] A. Delorenzo. Hardware Hacking 101: Interfacing with SPI. https://riverloopsecurity.com/blog/2020/02/hw-101-spi/, 2020.

[81] C. Details. Op-tee: Vulnerability Statistics. https://www.cvedetails.com/product/56969/Linaro-Op-tee.html and https://www.cvedetails.com/product/42749/Linaro-Op-tee.html and https://www.cvedetails.com/product/36161/Op-teeOp-tee-Os.html, 2021.

[82] DXOMARK. Samsung Galaxy S23 Ultra Battery test. https://www.dxomark.com/samsung-galaxy-s23-ultra-battery-test/, 2023.

[83] 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. AMP: Authentication of Media via Provenance. 2020.

[84] H. Farid. Image forgery detection. *IEEE Signal Processing Magazine*, 2009.

[85] H. Farrukh, R. Aburas, S. Cao, and H. Wang. Facerevelio: a face liveness detection system for smartphones with a single front camera. MobiCom '20, 2020.

[86] H. Farrukh, R. M. Aburas, S. Cao, and H. Wang. FaceRevelio: A Face Liveness Detection System for Smartphones with a Single Front Camera. In *Proc. ACM MobiCom*, 2020.

[87] S. Feng. Understanding Principle of Time of Flight Technology. https://industry.goermicro.com/blog/tech-briefs/understanding-the-principle-of-3d-time-of-flight-camera.html, 2024.

[88] J. Fingas. Woman allegedly made deepfakes to kick rivals off daughter's cheerleading squad. https://www.engadget.com/woman-creates-deepfakes-for-cheerleader-daughter-215024991.html, 2021.

[89] W. Gallagher. Apple's iPhone 14 Battery Capacities Revealed in Filing. https://appleinsider.com/articles/22/09/12/apples-iphone-14-battery-capacities-revealed-in-filing, 2022.

[90] A. Gehani and U. Lindqvist. VEIL: A System for Certifying Video Provenance. In *Proc. IEEE International Symposium on Multimedia (ISM)*, 2007.

[91] P. Gilbert, J. Jung, K. Lee, H. Qin, D. Sharkey, A. Sheth, and L. P. Cox. YouProve: Authenticity and Fidelity in Mobile Sensing. In *Proc. ACM SenSys*, 2011.

[92] C. Godard, O. Mac Aodha, M. Firman, and G. Brostow. Digging into self-supervised monocular depth prediction. 2019.

[93] S. Gordon, J. Katz, and V. Vaikuntanathan. A Group Signature Scheme from Lattice Assumptions. Cryptology ePrint Archive, Paper 2011/060, 2011. https://eprint.iacr.org/2011/060.

[94] GSMARENA. IPhone 14 Pro Max with A16 Chipset Appears on Geekbench with Minimal Performance Improvement, 2022. https://rb.gy/mz1un.

[95] L. Guo and F. Lin. Minimum Viable Device Drivers for ARM Trustzone. EuroSys, 2022.

[96] J. Han, B. Li, D. Mukherjee, C. Chiang, A. Grange, C. Chen, H. Su, S. Parker, S. Deng, U. Joshi, Y. Chen, Y. Wang, P. Wilkins, Y. Xu, and J. Bankoski. A Technical Overview of AV1. *Proceedings of the IEEE*, 2021.

[97] X. Han, Z. Feng, S. Sun, and G. Xiao. 3d point cloud descriptors: state-of-the-art. *Artif. Intell. Rev.*, 2023.

[98] D. Harwell. 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/, 2019.

[99] H. R. Hasan and K. Salah. Combating Deepfake Videos Using Blockchain and Smart Contracts. *IEEE Access*, 7:41596–41606, 2019.

[100] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. D. Cuvillo. Using Innovative Instructions to Create Trustworthy Software Solutions. 2013.

[101] G. Hotz. PS3 Glitch Hack. https://www.blogger.com/blogin.g?blogspotURL=http://geohotps3.blogspot.com/2010/01/hello-hypervisor-im-geohot.html&type=blog, 2010.

[102] M. Hu, W. Yin, C. Zhang, Z. Cai, X. Long, H. Chen, K. Wang, G. Yu, C. Shen, and S. Shen. Metric3d v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation. 2024.

[103] A. Huang. Hacking the Xbox: an Introduction to Reverse Engineering. https://hackingthexbox.com/, 2003.

[104] T. Hunt, Z. Jia, V. Miller, A. Szekely, Y. Hu, C. J. Rossbach, and E. Witchel. Telekine: Secure Computing with Cloud GPUs. In *Proc. USENIX NSDI*, 2020.

[105] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM Transactions on Computer Systems (TOCS)*, 2018.

[106] I. Hussain, D. Hussain, R. Kohli, M. Ismail, S. Hussain, S. Sajid Ullah, R. Alroobaea, W. Ali, and F. Umar. Evaluation of deep learning and conventional approaches for image recaptured detection in multimedia forensics. *Mobile Information Systems*, 2022.

[107] M. Ibrahim, A. Imran, and A. Bianchi. SafetyNOT: On the usage of the SafetyNet Attestation API in Android. In *Proc. ACM MobiSys*, 2021.

[108] F. Imeson, A. Emtenan, S. Garg, and M. Tripunitara. Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, 2013.

[109] I. Insights. CMOS Image Sensor Sales Stay On Record-Breaking Pace. https://electroiq.com/files/blog/2018/05/cmos-image-sensor-sales-stay-on-record-breaking-pace/, 2022.

[110] Y. Jeong, D. Kim, J. Lee, M. Hong, S. Hwang, and J. Choi. mtofnet: Object anti-spoofing with mobile time-of-flight data. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2022.

[111] M. Khelif, J. Lorandel, O. Romain, M. Regnery, D. Baheux, and G. Barbu. Toward a Hardware Man-In-The-Middle Attack on Pcie Bus. *Microprocessors and Microsystems*, 2020.

[112] J. Knechtel, O. Sinanoglu, I. Elfadel, J. Lienig, and C. Sze. Large-Scale 3D Chips: Challenges and Solutions for Design Automation, Testing, and Trustworthy Integration. *IPSJ Transactions on System and LSI Design Methodology*, 2017.

[113] P. Korshunov and S. Marcel. DeepFakes: a New Threat to Face Recognition? Assessment and Detection. *CoRR*, abs/1812.08685, 2018.

[114] N. Lawson. How the PS3 Hypervisor Was Hacked. https://rdist.root.org/2010/01/27/how-the-ps3-hypervisor-was-hacked/, 2010.

[115] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang. Hacking in Darkness: Return-Oriented Programming Against Secure Enclaves. In *Proc. USENIX Security Symposium*, 2017.

[116] M. Lentz. Assurance and Control over Sensitive Data on Personal Devices. https://https://doi.org/10.13016/nx2k-waen, 2020.

[117] H. Li, W. Li, H. Cao, S. Wang, F. Huang, and A. Kot. Unsupervised domain adaptation for face anti-spoofing. *IEEE Transactions on Information Forensics and Security*, 2018.

[118] J. Li, C. Kong, S. Wang, and H. Li. Two-branch multi-scale deep neural network for generalized document recapture attack detection. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

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

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

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

[122] Y. Liu, A. Jourabloo, and X. Liu. Learning deep models for face anti-spoofing: Binary or auxiliary supervision. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[123] Y. Liu, Y. Nakatsuka, A. Amiri Sani, S. Agarwal, and G. Tsudik. Vronicle: Verifiable provenance for videos from mobile devices. MobiSys '22, 2022.

[124] Y. Liu, Y. Nakatsuka, A. Amiri Sani, S. Agarwal, and G. Tsudik. Provcam: A camera module with self-contained tcb for producing verifiable videos. ACM MobiCom '24, 2024.

[125] Y. M. Liu, H. Farrukh, A. A. Sani, S. Agarwal, and G. Tsudik. Scoop: Mitigation of recapture attacks on Provenance-Based media authentication. In *34th USENIX Security Symposium (USENIX Security 25)*, 2025.

[126] V. Matyás and Z. Ríha. Biometric authentication - security and usability. In *Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security: Advanced Communications and Multimedia Security*, 2002.

[127] S. McCloskey and M. Albright. Detecting GAN-generated Imagery using Color Cues. *CoRR*, abs/1812.08247, 2018.

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

[129] S. Mervosh. 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, 2019.

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

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

[132] NIST. CVE-2015-6639 Detail. https://nvd.nist.gov/vuln/detail/CVE-2015-6639, 2016.

[133] R. Palum. Image Sampling with the Bayer Color Filter Array. In *PICS*, pages 239–245, 2001.

[134] H. Park and F. Lin. GPUReplay: A 50-KB GPU Stack for Client ML. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.

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

[136] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[137] L. Piccinelli, Y. Yang, C. Sakaridis, M. Segu, S. Li, L. Van Gool, and F. Yu. UniDepth: Universal monocular metric depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

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

[139] M. Posner. How Many Asic Gates Does It Take To Fill an FPGA? https://blogs.synopsys.com/breakingthethreelaws/2015/02/how-many-asic-gates-does-it-take-to-fill-an-fpga/, 2015.

[140] Qualcomm. Snapdragon 8 gen 2 mobile platform, 2023. https://www.qualcomm.com/products/mobile/snapdragon/smartphones/snapdragon-8-series-mobile-platforms/snapdragon-8-gen-2-mobile-platform.

[141] Quarklab. BREAKING SAMSUNG'S ARM TRUSTZONE. https://i.blackhat.com/USA-19/Thursday/us-19-Peterlin-Breaking-Samsungs-ARM-TrustZone.pdf, 2019.

[142] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

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

[144] A. Rustamov, N. Gogoi, A. Minetto, and F. Dovis. Assessment of the vulnerability to spoofing attacks of gnss receivers integrated in consumer devices. *2020 International Conference on Localization and GNSS (ICL-GNSS)*, pages 1–6, 2020.

[145] R. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011.

[146] I. Sarandi, T. Linder, K. Arras, and B. Leibe. Metrabs: Metric-scale truncation-robust heatmaps for absolute 3d human pose estimation. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 2021.

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

[148] A. Satariano and P. Mozur. The People Onscreen Are Fake. The Disinformation Is Real. 2023. https://www.nytimes.com/2023/02/07/technology/artificial-intelligence-training-deepfake.html.

[149] M. Shih, S. Lee, T. Kim, and M. Peinado. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs. In *Proc. NDSS Symposium*, 2017.

[150] Y. Shizuku, T. Hirose, N. Kuroki, M. Numa, and M. Okada. A 24-Transistor Static Flip-Flop Consisting of Nors and Inverters for Low-Power Digital Vlsis. In *2014 IEEE 12th International New Circuits and Systems Conference (NEWCAS)*, 2014.

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

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

[153] J. Stehouwer, A. Jourabloo, Y. Liu, and X. Liu. Noise modeling, synthesis and classification for generic object anti-spoofing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[154] G. Stocking, P. Van Kessel, M. Barthel, K. Eva Matsa, and M. Khuzam. Many Americans Get News on YouTube, Where News Organizations and Independent Producers Thrive Side by Side. https://www.pewresearch.org/journalism/2020/09/28/many-americans-get-news-on-youtube-where-news-organizations-and-independent-producers-thrive-side-by-side/, 2020.

[155] V. Strumpen. Introduction to Digital Circuits: Basic Digital Circuits. http://bibl.ica.jku.at/dc/build/html/basiccircuits/basiccircuits.html, 2015.

[156] G. Sullivan, J. Ohm, W. Han, and T. Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 2012.

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

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

[159] Tezzaron. 3D-ICs and Integrated Circuit Security. 2008. https://tezzaron.com/media/3D-ICs_and_Integrated_Circuit_Security.pdf.

[160] T. Thomson, D. Angus, and P. Dootson. 3.2 billion images and 720,000 hours of video are shared online daily. Can you sort real from fake? https://theconversation.com/3-2-billion-images-and-720-000-hours-of-video-are-shared-online-daily-can-you-sort-real-from-fake-148630, 2020.

[161] T. Thongkamwitoon, H. Muammar, and P. Dragotti. An image recapture detection algorithm based on learning dictionaries of edge profiles. *IEEE Transactions on Information Forensics and Security*, 2015.

[162] R. Toews. 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/, 2020.

[163] A. Trabelsi, M. Pic, and J. Dugelay. Recapture detection to fight deep identity theft. VSIP '22, 2023.

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

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

[166] M. Wales. The Best 8K Phones in 2023. https://filmora.wondershare.com/8k/best-8k-resolution-phones.html, 2023.

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

[168] J. Wang, A. Li, H. Li, C. Lu, and N. Zhang. RT-TEE: Real-time System Availability for Cyber-physical Systems using ARM TrustZone. In *2022 IEEE Symposium on Security and Privacy (SP)*, 2022.

[169] K. Wang. Time and position spoofing with open source projects. 2015.

[170] M. Westerlund. The Emergence of Deepfake Technology: A Review. 2019. http://doi.org/10.22215/timreview/1282.

[171] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H. 264/Avc Video Coding Standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

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

[173] T. Winkler and B. Rinner. 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)*, 2010.

[174] Xilinx. Achieving an Accurate Power Analysis Using Vivado Report Power. https://docs.xilinx.com/r/en-US/ug907-vivado-power-analysis-optimization/Achieving-an-Accurate-Power-Analysis-Using-Vivado-Report-Power, 2023.

[175] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, 2024.

[176] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao. Depth anything v2. *arXiv:2406.09414*, 2024.

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

[178] W. Yin, C. Zhang, H. Chen, Z. Cai, G. Yu, K. Wang, X. Chen, and C. Shen. Metric3d: Towards zero-shot metric 3d prediction from a single image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[179] Z. Zhang, X. Zhan, and H. Xu. Development and validation of a low-cost gps spoofing simulator. *Journal of Aeronautics, Astronautics and Aviation*, 2014.

Appendix A

Scoop: Mitigation of Recapture Attacks on Provenance-Based Media Authentication

A.1 Answer to The Figure

In Figure 4.1, the right one is a real photo. The left photo was displayed on our TCL Mini-LED TV, with the person in the photo digitally erased.

A.2 User Study

A.2.1 Photos

Figure A.1 shows all the photos we used in our user study. Original photos include (d), (e), (h), (j), (l), (n), (o), and (p). Recaptured photos include (a), (b), (c), (f), (g), (i), (k), and (m).



(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

Figure A.1: Photos used in our user study.

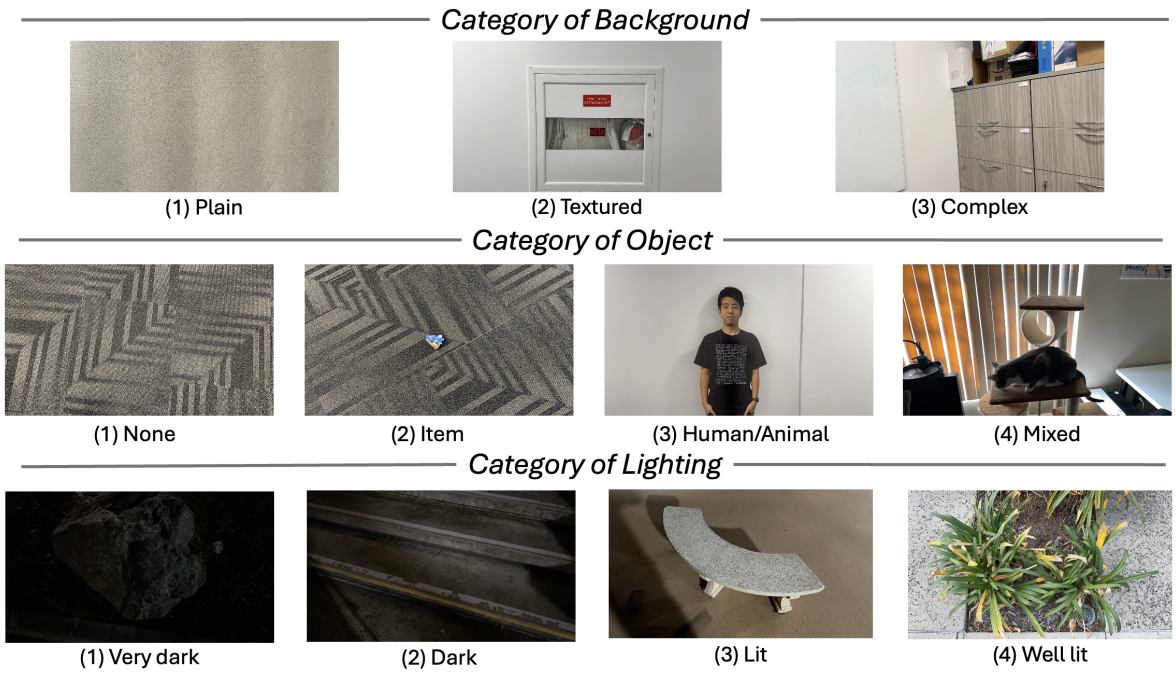


Figure A.2: Our dataset's three categories: background, object, and lighting.

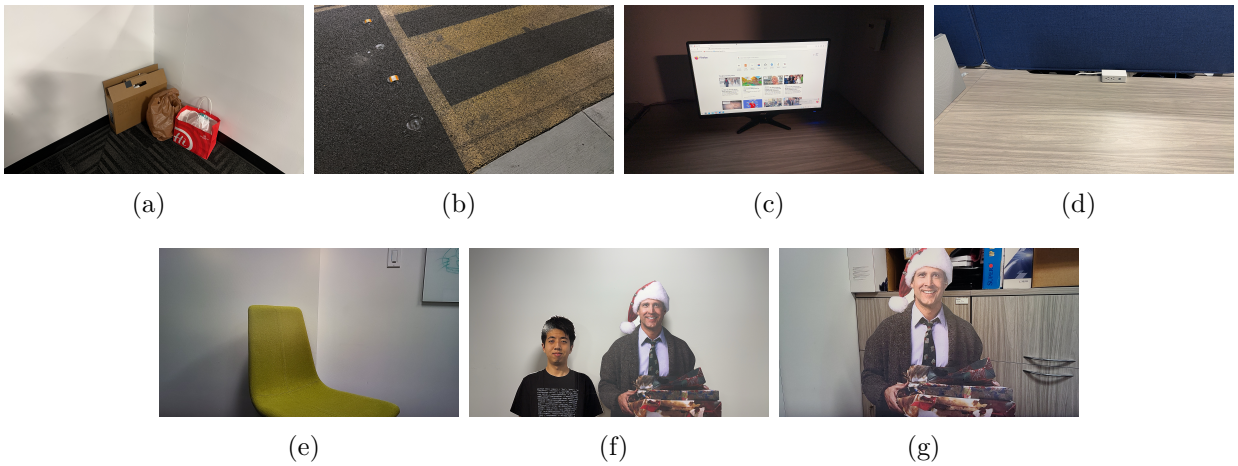


Figure A.3: More sample photos in our dataset.

A.3 Dataset Photos

Figure A.2 shows an example for each option of the three categories as mentioned in the main body. In addition to the sample photos for different categories in the dataset and user study, Figure A.3 shows more photos from our dataset. Original photos include (a), (b), (c), and (d). Recaptured photos include (e), (f), and (g).

A.4 Artifact

There are four artifact components in our work.

- **Viewer**: The Scoop Viewer is the main tool that we developed to detect the existence of misleading recaptures in photos/videos.
- **iOS App**: The iOS App is a mobile application that can be used to capture photos/videos that can be analyzed by the Scoop Viewer.
- **Android App**: The Android App is a mobile application that can be used to capture photos/videos that can be analyzed by the Scoop Viewer.
- **Dataset**: The dataset that can be used to evaluate systems such as Scoop.

They can all be found on our Zenodo page at https://doi.org/10.5281/zenodo.15611904. There are three ways to test the Scoop system, which are described in the following subsections.

A.4.1 Quick Test (Recommended)

We provide a quick way to test out the Scoop system. Please download the Scoop Viewer (viewer.zip), which contains everything needed to conduct a quick test of Scoop.
The Scoop Viewer requires the following dependencies:

- C++ compiler (e.g., g++, clang++)

- CMake 3.5 or later
- OpenCV 4.5 or later
- PCL (Point Cloud Library) 1.12 or later
- Boost 1.75 or later
- Eigen 3.3 or later
- Python 3.9 or later

You may set up the dependencies yourself and modify the CMakeLists.txt file accordingly, or you can use our provided script to set up the environment automatically. Our script is tested on Ubuntu 24.04 LTS, but it should work on other Linux distributions (e.g., Fedora, Arch Linux) and MacOS (with Homebrew) as well. Assuming the viewer is now at a directory called `$VIEWER_DIR`, you can run the following commands to set up the environment:

```
cd $VIEWER_DIR
bash ./scripts/install_required_libraries.sh
```

After the installation is complete, you can run the following commands to build the Scoop Viewer:

```
cd $VIEWER_DIR
cmake .
make -j$(nproc)
```

After the build is complete, you can run the quick test with the following command:

```
cd $VIEWER_DIR
python3 ./scripts/eval.py sample_data/ 0000 9999
```

This command will run the Scoop Viewer on the sample data in the `sample_data` directory, which contains 8 sets of data provided (with 4 unique data points). Among the 4 data points, 1 is original and the rest 3 are recaptured photos (with 2 TVs and 1 projector).

A.4.2 Test with Your Own Data

You can also test the Scoop Viewer with your own data. To do this, you can either use our iOS or Android app to capture photos/videos, or you can use your own camera to capture photos/videos. However, please make sure you follow the guidelines in the iOS/Android app repositories for correctly capturing the data or refer to the Scoop Viewer repository for the correct format of the data. For building iOS and Android apps, please refer to the respective repositories for instructions.

After you have extracted the data from your iOS/Android app, you need to generate perceived depth data for each photo/video. Please refer to the Scoop Viewer repository for instructions on how to generate perceived depth data. You can pick any depth estimation model that you prefer, but we recommend using the ml-depth-pro model, which we included a script for using it in the Scoop Viewer repository. You may need to refer to the script to see how to generate the perceived depth data for your photos/videos and make sure the generated data is in the correct format so that the Scoop Viewer can analyze it. After you have generated the perceived depth data, you can run the Scoop Viewer on your data with the commands provided in the Scoop Viewer repository.

A.4.3 Test with the Dataset

You can also test the Scoop Viewer with our dataset. We provide full instructions on how to use the dataset in the Scoop Viewer repository. You can also refer to the Scoop dataset repository for more information about the dataset.