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

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

Zhihao Yao
zhihao.yao@njit.edu
New Jersey Institute of Technology
Newark, New Jersey, USA

Mingyi Chen
mingyi.chen@uci.edu
University of California, Irvine
Irvine, California, USA

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

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

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

Abstract

Our perception of reality is under constant threat from ever-improving video manipulation techniques, including deepfakes and generative AI. Therefore, proving authenticity of videos is increasingly important, especially in legal and news contexts. However, it is very challenging to prove it based on post-factum video content analysis.

In this work, we take a preventative stance and construct ProvCam, a novel camera module that generates a cryptographic proof of video authenticity. Our solution greatly reduces the size of Trusted Computing Base (TCB) to include the module itself. Moreover, it mitigates tampering during the numerous processing steps between video capture by the camera sensor and generation of the digital video output. To confirm its practicality, we present a complete prototype of ProvCam on a Xilinx FPGA evaluation board. As experiments show, ProvCam incurs a negligible performance overhead (latency and throughput) and small energy consumption overhead when recording a video. It imposes a moderate hardware cost but is relatively small compared to other major components such as SoC. Moreover, it does not change the existing camera software stack and thus can be easily integrated with various camera-bearing devices, such as smartphones.

MobiCom '24, November 18-22, 2024, Washington, D.C., USA

ACM ISBN 979-8-4007-0489-5/24/09
https://doi.org/10.1145/3636534.3649383

CCS Concepts

• **Security and privacy → Hardware security implementation**; **Mobile platform security**.

Keywords

Video provenance, Deepfakes, Secure camera

ACM Reference Format:
Yuxin (Myles) Liu, Zhihao Yao, Mingyi Chen, Ardalan Amiri Sani, Sharad Agarwal, and Gene Tsudik. 2024. ProvCam: A Camera Module with Self-Contained TCB for Producing Verifiable Videos. In *The 30th Annual International Conference On Mobile Computing And Networking (MobiCom '24), September 30-October 4, 2024, Washington, D.C., USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3636534.3649383

1 Introduction

The importance of videos in modern society has grown tremendously. Prior to the appearance of affordable and ubiquitous cameras and smartphones, videos had a high barrier to entry, i.e., production studios, licensing, broadcast means and rights, etc. Whereas now, any camera-equipped personal device can produce videos and they are mostly watched on various social media platforms. However, videos still serve very important societal functions, such as observing law enforcement actions, legal proceedings, citizen journalism, or government negotiations. Indeed, user-generated videos are increasingly popular: 26% of U.S. adults get their news from YouTube, though 42% of the most popular news channels on YouTube are not associated with any news or journalism agency [1]. Video is also used for evidential purposes in courts. Around 80% of all crimes involve some video evidence [2].

In all of these use cases, 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 [3] and generative AI technologies, consumers cannot even determine whether a video is synthetic or was captured by a real camera. Deepfake technology can alter arbitrary elements in an existing video or turn a picture into a video, e.g., swap people's faces, synchronize the movement of someone's lips with an arbitrary audio piece, and change the background or an object in video frames [4–6]. By analyzing existing videos in a training set and producing new frames that mimic existing ones, deepfake methods can fabricate videos of events that never happened [7–9]. One notable example is a manipulated video of the president of Ukraine, Volodymyr Zelenskyy, discussing the war between Russia and Ukraine [10]. Another well-known deepfake example is the video of the Speaker of the U.S. House of Representatives, Nancy Pelosi, which widely circulated on social media [11]. Deepfake technology has been abused to spread disinformation, and perpetrators are producing increasingly convincing videos with constantly improving tools [12].

There are two technical[1] mitigation approaches: detection- and prevention-based.

The former detects known flaws in manipulated and fabricated videos [13–18]. Although human eyes can not easily identify various nuances, machine learning algorithms can analyze video frames and detect certain known inconsistencies, such as unnatural body movement, lighting, and texture. Unfortunately, deepfake technology is advancing rapidly, thus making detection increasingly difficult [12]. This results in the usual (and potentially never-ending) "arms race" between attackers and defenders. Perhaps fighting AI with AI is futile since the deepfake technology can continuously learn and improve, based on advances in detection algorithms [19].

The prevention-based approach focuses on video provenance based on cryptographic methods. [20–23]. 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 [20], while TruePic [23] utilizes a Trusted Execution Environment (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 paper, we tackle this problem by constructing ProvCam– a camera module for generating verifiable videos.[2] Compared with the large TCB (both hardware and software) of TEE-based solutions (which we discuss in detail in Section 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 [20].

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)

[1] There are, of course, legal means of mitigating deepfakes as well.

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

– 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). 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).
- 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.

2 Threat Model & Motivation

Video provenance is a popular and promising approach for mitigating fake videos [20–23]. 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 [20] or zero-knowledge proofs [21], 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.

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 [24, 25] 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) [26, 27].

Although the assumed adversary is quite powerful, we believe that it is realistic given that similar attacks were successful in the past [28, 29] including active bus tapping attacks [30–34]. 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 [27], 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 §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.

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 [22] and Alethia [35] 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 [20, 23]. 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 – §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 [28, 36–38]. 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 [39], 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 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.

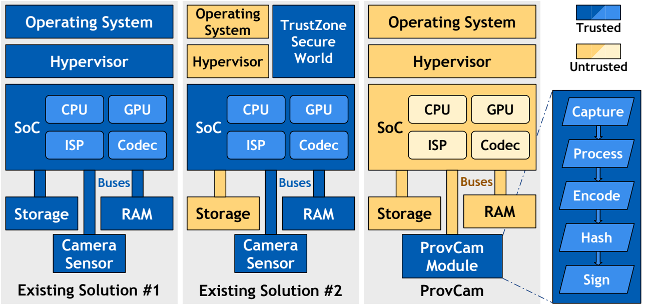

Figure 1: ***Comparison between two current approaches and ProvCam's TCBs. Detailed flow inside ProvCam camera module is shown in Figure 3.***

Digital camera sensor. The prevalent type of camera sensor in modern mobile and embedded devices is the Complementary Metal-Oxide-Semiconductor (CMOS) image sensor [40]. 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 [41]. 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 [42]. 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 processing 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 [43]. There are other modern coding standards such as High Efficiency Video Coding (HEVC, a.k.a, H.265) [44] and AOMedia Video 1 (AV1) [45]. While we focus on H.264 in this paper, ProvCam design can be modified to also support such alternative codecs.

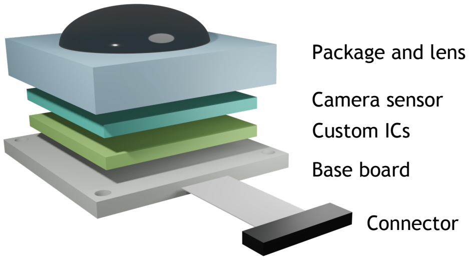

Figure 2: ***3D-stacked secure packaging of ProvCam's camera module.***

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 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 1 shows the TCB comparison of two current approaches discussed in §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 §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.

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 §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 [46, 47].

However, to give an idea of what the module might look like, we illustrate one possibility. Figure 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 [48, 49], and there has been at least one proposal for applying such technology to CMOS camera sensors [50].

5 Hardware Design Challenges

5.1 The Use of Untrusted Main Memory

As illustrated in Figure 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 (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 §9.1).

To address this issue, we instead securely use the untrusted camera device's main memory, as shown in Figure 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 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.

We observe that, despite different access patterns, ISP and encoder both access a few rows of the frame at a time. That

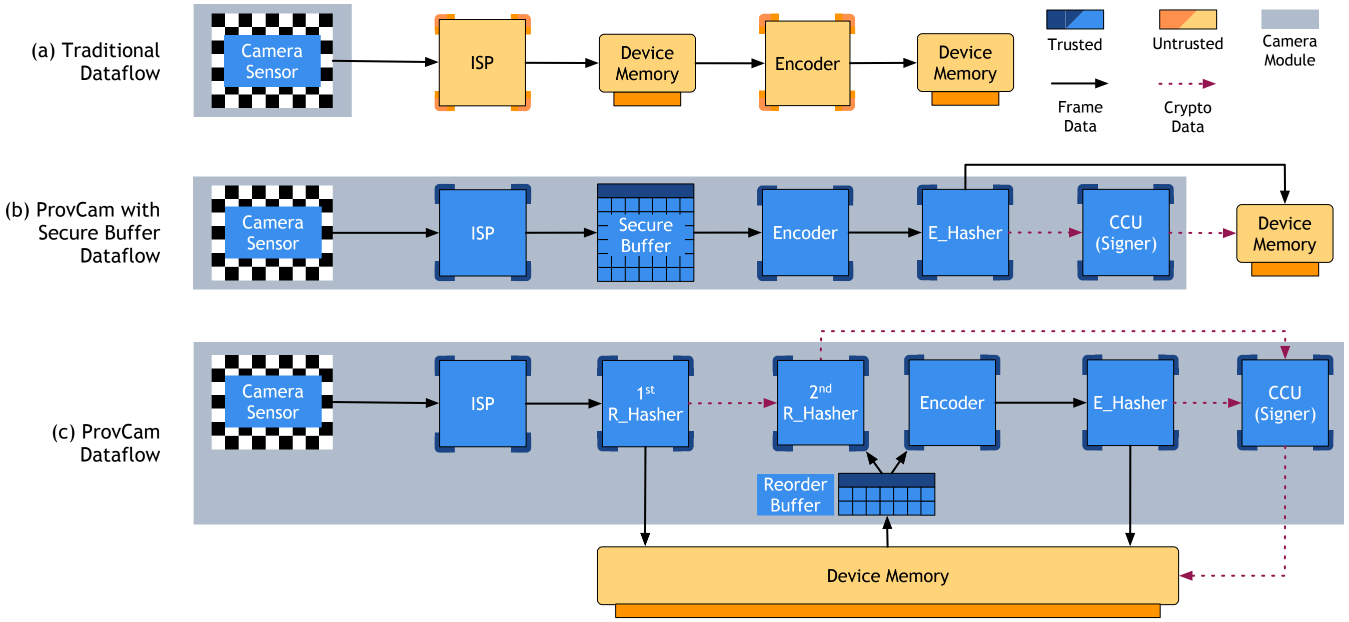

Figure 3: *Comparisons between a typical video capture pipeline and ProvCam's secure pipelines.*

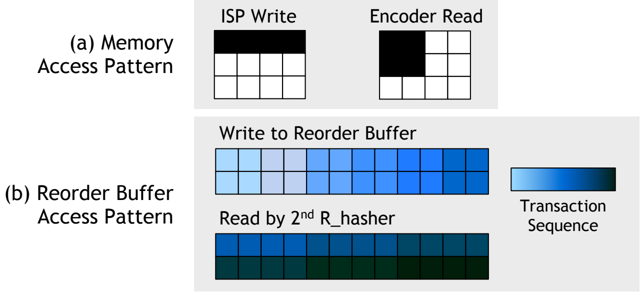

Figure 4: *Comparison between how ISP and encoder write/read raw frames and how the secure reorder buffer addresses it.*

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 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 (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_hashers*, 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 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.

5.2 Asynchronous Hashing

The data stream that serves as input for *r_hashers* 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 [51].

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 [52].

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.

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.

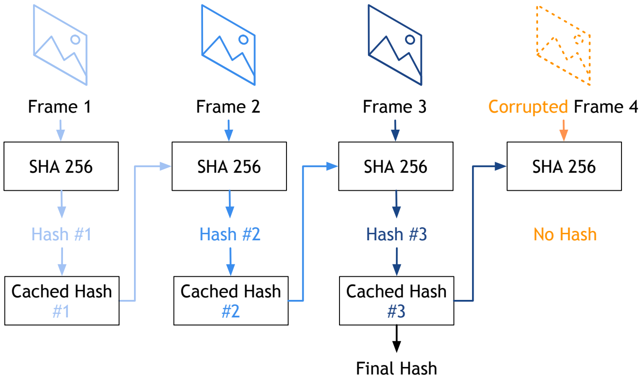

Figure 5: *How frame-based hashing works in ProvCam.*

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 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.

6 Software Stack Design Challenges

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.

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.

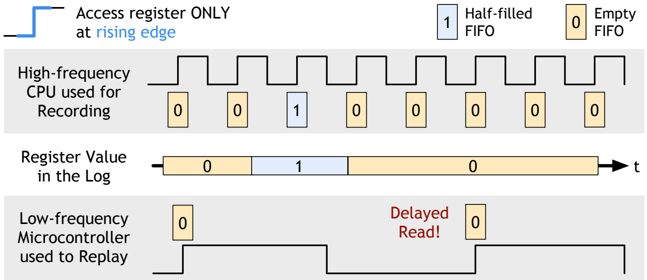

Figure 6: ***An example of mismatched I/O read of a FIFO's status register.***

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 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 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.

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).

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.

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).

8 Security Analysis

We analyze and evaluate the security of our solution against various attacks in the adversary's arsenal (§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 [20], 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 [53]. 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.

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 (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 (a)). The other is a prototype of ProvCam with the use of a secure buffer inside the module (Figure 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 (§5.1).

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 1: Extra hardware cost in both hardware prototypes.

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 [54], and Qualcomm's Snapdragon 8 Gen 2 processor runs at 3.36 GHz [55]. 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.

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 [56], each of which requires 6 transistors [57], bringing it to a total of 36 transistors for each look-up table. We assume each flip-flop requires 24 transistors [58]. Lastly, we assume each SRAM cell (bit) uses the 6-transistor design [59].

Table 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

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

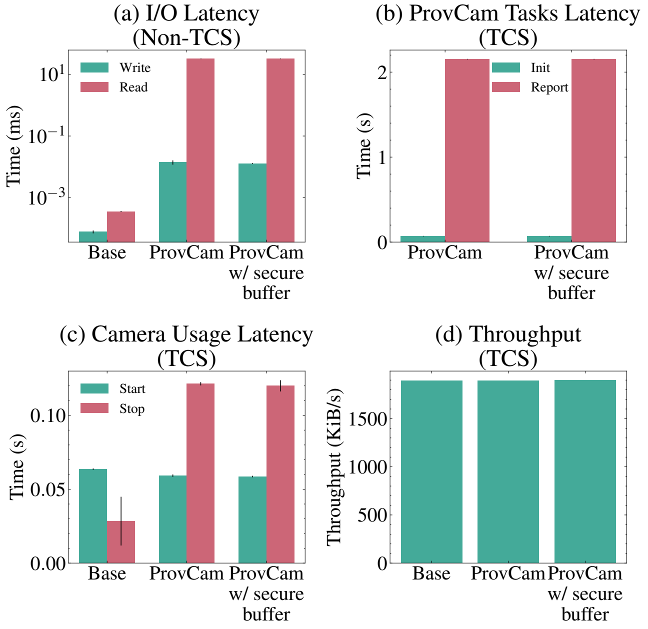

Figure 7: ***Performance evaluations of ProvCam. The histograms show the average. The error bars show the standard deviation.***

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 the size of reorder buffer for untrusted memory only grows logarithmically. Table 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 [60].

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 [61]. 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.

9.3 Performance

We evaluate latency and throughput, with two types of experiments: microbenchmarks and macrobenchmarks. For microbenchmarks, as shown in 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 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 7 (c). We also measure and report the throughput in 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.

9.4 Energy Consumption

We estimate the energy consumption using the power report of our hardware prototypes generated by the Xilinx Vivado software [62]. 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 [63], and Samsung Galaxy S23 Ultra, which has a 18.5 Wh (66600 joules) battery [64].

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.

10 Discussions & Future Work

10.1 Trusted and Normal Videos

We have mentioned in Section 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).

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) [69] and lattice-based group signature [70], 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.

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.

11 Related Work

Securing Camera by Provenance. Several previous works have used provenance to secure videos [22, 35, 68, 71–73], such as by watermarking, hashing, signing, and chaining. Other works provide provenance for photos only.

	Approach	**Performance**	**Videos?**	**Vulnerable to attacks** (Refer to §2.1)	**TCB**
Aletheia [35]	Digital Signature	No measurement	Yes	4,5,6,7	OS
PhotoProof [21]	Zero Knowledge Proof	673.5 secs for 1 frame at 128x128	No	4,5,6,7	OS
YouProve [65]	Image Understanding	28 secs for 1 frame at 1296x792	No	4,5,6,7	OS
TrustEYE.M4 [66]	TPM	11 fps at 320x240	Yes	6,7	TEE + others
TrustCAM [67]	TPM	25 fps at 640x480	Yes	6,7	TEE + others
FrameProv [68]	TEE	No prototype	Yes	5,6,7	TEE + others
Vronicle [20]	TEE	Native	Yes	5,6,7	TEE
ProvCam	Secure Camera Module	Native	Yes	None	Tiny

Table 3: Comparison with related work. Note that TEE + others means the system TCB includes not only TEE, but also other (potentially less trustworthy) components

Photoproof [21] makes use of zero-knowledge proof to cryptographically show that the photo is edited properly. YouProve [65] performs analysis on the photo to study changes. Vronicle [20] generates fine-grained provenance information and use it to keep track of all processing history of the video throughout its lifespan. In addition, there are industrial efforts to secure the camera-captured content. Canon made an attempt in trustworthy photographing by developing its Original Data Security Kit [74] for proving image originality, which was later found as flawed [75]. Recent efforts include Truepic [23] and Serelay [76], which use TEEs to secure photos. Moreover, Adobe's Content Authenticity Initiative (CAI) [77] currently has 52 participating members, including the above-mentioned two and others such as Microsoft. There has also been some research effort on securing the camera sensor using TEE. TrustCAM [67] and TrustEYE.M4 [66] make use of the Trusted Platform Module (TPM) for signing frames produced by the camera sensor.

However, to the best of our knowledge, ProvCam is the first solution for securing the entire camera pipeline that can be adapted by all kinds of camera devices, and has adequate performance for capturing high-resolution videos. Table 3 compares ProvCam against other key related works.

Secure Sensors. Saroiu et al. utilize Intel Trusted eXecution Technology for authenticating sensors' readings such as camera and GPS's data [78, 79]. Shahid et al. introduce a QR-code based solution for securing speech audio in live recordings [80]. CamShield [81] also provides a system-wide solution for securing a camera device by shielding it with a trusted camera device. These solutions, however, do not address the authenticty of recorded video.

Replay-based I/O. Some works also secure I/O using record-and-replay. Both RT-TEE [82] and driverlets [83] 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 [84] 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.

12 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.

Acknowledgments

The work of UCI authors was supported in part by the NSF Awards #1763172, #1953932, #1956393, and #2247880 as well as NSA Awards #H98230-20-1-0345 and #H98230-22-1-0308. The authors thank the anonymous shepherd and reviewers for their insightful comments.

References

[1] 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.

[2] D. Garrison. Advanced Video Forensics, 2014. https://www.oceansystems.com/newsroom/documents/201407ETMAdvancedForensicVideoAnalysis_000.pdf.

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

[4] I. Perov, D. Gao, N. Chervoniy, K. Liu, S. Marangonda, C. Umé, Mr. Dpfks, C. Facenheim, L. RP, J. Jiang, S. Zhang, P. Wu, B. Zhou, and W. Zhang. Deepfacelab: Integrated, Flexible and Extensible Face-Swapping Framework. https://arxiv.org/abs/2005.05535, 2021.

[5] K. Prajwal, R. Mukhopadhyay, V. Namboodiri, and C. Jawahar. A Lip Sync Expert Is All You Need for Speech to Lip Generation In The Wild. https://arxiv.org/abs/2008.10010, 2020.

[6] L. Kant. Image-Background-Removal-And-Replacement-Using-Ml-And-Ai. https://github.com/laxmimerit/image-background-removal-and-replacement-using-ml-and-ai, 2023.

[7] Synthesia: Create Videos from Plain Text in Minutes. https://www.synthesia.io/, 2017.

[8] Imagen Video. https://imagen.research.google/video/, 2022.

[9] Make-A-Video. https://makeavideo.studio/?fbclid=IwAR0KjiiCmLKNhPs7S6RLCLG3U7_5yex0NHP8-6DfABUtf6SBpKgMkVykEcU, 2022.

[10] S. Cole. Hacked News Channel and Deepfake of Zelenskyy Surrendering Is Causing Chaos Online. https://www.vice.com/en/article/93bmda/hacked-news-channel-and-deepfake-of-zelenskyy-surrendering-is-causing-chaos-online.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] J. MacGillis. Fighting AI with AI: The Battle against Deepfakes. https://thewalrus.ca/fighting-ai-with-ai-the-battle-against-deepfakes/, 2022.

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

[21] 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.

[22] 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.

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

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

[25] 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.

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

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

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

[34] 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.

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

[36] CVE 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.

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

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

[39] 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.

[40] IC 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.

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

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

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

[48] 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.

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

[50] 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.

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

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

[53] 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.

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

[55] 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.

[56] 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.

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

[58] 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.

[59] 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.

[60] 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.

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

[62] 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.

[63] 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.

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

[65] 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.

[66] 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.

[67] 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.

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

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

[70] 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.

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

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

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

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

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

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

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

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

[79] 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.

[80] I. Shahid and N. Roy. "Is This My President Speaking?" Tamper-Proofing Speech in Live Recordings. MobiSys '23, 2023.

[81] Wang Z., Yan Y., Yan Y., Chen H., and Yang Z. CamShield: Securing Smart Cameras through Physical Replication and Isolation. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[82] 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.

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

[84] 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.