CamShield: Securing Smart Cameras through Physical Replication and Isolation

Zhiwei Wang[1] Yihui Yan[1] Yueli Yan[1] Huangxun Chen[2] Zhice Yang[1] *
[1]*School of Information Science and Technology, ShanghaiTech University*
[2]*Huawei Theory Lab*

Abstract

Smart home devices, such as security cameras, are equipped with visual sensors, either for monitoring or improving user experience. Due to the sensitivity of the home environment, their visual sensing capabilities cause privacy and security concerns. In this paper, we design and implement the CamShield, a companion device to guarantee the privacy of smart security cameras, even if the whole camera system is fully compromised. At a high level, the CamShield is a shielding case that works by attaching it to the front of the security camera to blind it. Then, it uses its own camera for visual recording. The videos are first protected according to user-specified policies, and then transmitted to the security camera and hence to the Internet through a Visible Light Communication (VLC) channel. It ensures that only the authorized entities have full access to the protected videos. Since the CamShield is physically isolated from the shielded security camera and the Internet, it naturally resists many known attacks and can operate as it is expected to.

1 Introduction

Smart home represents the intelligentization trend of house appliances, accessories, and furniture. It brings us convenience but at the cost of our sound and video information being gathered and analyzed. More than 70% of surveyed consumers are very concerned about the risks of being spied on by their smart home devices [11].

Their concerns are not fiction. Visual sensors are ubiquitous in smart home devices. They reside in places like security cameras and doorbell cameras, and are in new devices like smart TVs [17], refrigerators [15], pet monitors [5], *etc.* They are essential to smart devices since they render visual capabilities, *e.g.*, remote monitoring, video analysis, *etc.* However, in many cases, networked smart devices are neither transparent nor secure. The recorded videos are subject to various vulnerabilities [27] and unauthorized leakages. It is reported that curious cloud might spy on the content [14]. Used Nest security cameras might leak videos about the former user [3]. When Xiaomi IoT cameras are connected to the Google Hub, the monitoring images might show scenes of other users [24].

The security and privacy issues of visual-capable smart devices have led to many discussions [50, 51, 64]. A common solution is to protect videos in the device before transferring them out, but it will fail if the device has been taken over by the attacker [1, 23], *i.e.*, it is fully compromised. This leads to the concept of trusted camera, which operates strictly in the way specified by its owner. One possible way to build a trusted camera is to make use of the trusted execution environment (TEE), which isolates the critical computing components so that they cannot be tampered with. However, a complete TEE requires secure hardware such as compatible processors [30] and visual sensors capable of hardware encryption [53], which cannot be satisfied by legacy devices already in use. Moreover, since some smart devices also need to perform local video processing, such as face detection, the TEE must also cover the processing pipelines, which is still an open problem due to the large code base size.

We believe a viable solution of trusted camera should be able to resist most threats, practically feasible, and preferably compatible with legacy devices. Our key insight is that the major complexities and vulnerabilities of trusted computing systems are rooted in their design philosophy - isolating trusted and untrusted computing components while leaving them *sharing the same hardware or even the software.* However, why can they not be physically isolated? This question motivates our approach.

1.1 CamShield Approach

CamShield is a hardware-software co-designed system to force ordinary visual sensing devices to become trusted cameras. Without loss of generality, this paper primarily concentrates on smart security cameras (or smart cameras in short). As shown in Figure 1, the CamShield system consists of the *CamShield device* and the *CamShield App*. At a high level, the

*Corresponding Author

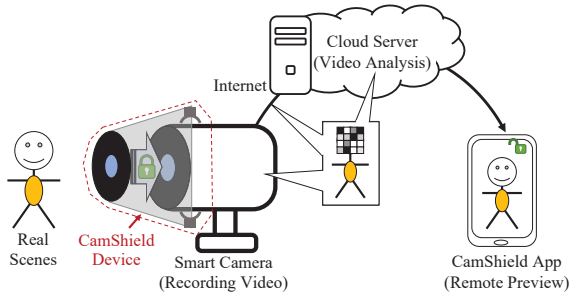


Figure 1: **Concept of CamShield**. The CamShield system consists of the CamShield device (red) and the CamShield App to secure the smart camera. It protects the privacy of the video content even if the smart camera is fully compromised. The CamShield device blinds the smart camera with its shield and replicates the visual sensing capability with its own camera sensor. It captures and delivers videos to the CamShield App through an encrypted channel. The CamShield device is physically and logically isolated from the network and the smart camera, and hence resists many known attacks.

CamShield device (or the CamShield in short) is a physical shield in front of the smart camera. When the CamShield is properly installed, the shielded camera can only capture the inside part of the CamShield, and hence is blinded in visually sensing the surrounding environment, *e.g.*, the living room. A blind camera is no longer subject to privacy leaks and attacks. To preserve the original functionalities of the smart camera, the CamShield has a visual sensor in front of the covered smart camera for visual sensing. The recorded video is encrypted, and then transferred to the companion app - the CamShield App, through the network connection of the smart camera. The owner uses the CamShield App to decrypt the video captured by the CamShield.

The trustworthiness of the CamShield system is based on two facts. First, the CamShield device is standalone. Its computing components are not only logically but also physically isolated from the smart camera and the network. Such isolation protects it against most practical attacks. Second, the only input interface of the CamShield is its visual sensor, which can hardly be abused to modify its software logic.

1.2 Contributions

To realize the idea of CamShield, we must address the following challenges: First, how can the CamShield transfer the video stream to the smart camera without violating the isolation? One solution is to use cables to wire their pins, but this needs hardware and firmware modifications on smart cameras. Our approach is to reuse the visual sensing ability of the smart camera to perform visible light communication (VLC). The CamShield contains a tiny screen inside its shield case. The captured video stream is encoded into a QR-code-like stream and displayed on it. Then, the smart camera captures and decodes the code stream to obtain the video. This one-way VLC data path keeps the CamShield isolated from the smart camera without requiring any modifications to the smart camera.

Second, CamShield uses end-to-end encryption between the CamShield and the App to guarantee the security of the videos. However, a fully-encrypted video eliminates the benefits from third-party services, *e.g.*, cloud video analysis. CamShield adopts a partially-encryption scheme, which encrypts only the sensitive areas or the region of interest (ROI) of the video, and thus allows the cloud servers to provide useful interference, *e.g.*, gestures, without knowing sensitive information, *e.g.*, the face in Figure 1. The problem is how can the user specify the ROIs according to his/her own concerns, as the CamShield is an isolated device, which lacks common input interfaces. We use another VLC channel from the owner's smartphone screen to the CamShield's camera to send configuration messages. It incorporates authentication protocols and hardware interfaces to enhance security and user awareness.

Third, as a key selling feature, most smart security cameras support motion detection, *e.g.*, to detect intruders. However, when using the CamShield, the smart camera captures VLC streams rather than the actual environment, hence the original motion detection module no longer works. We use compatibility designs to preserve this feature. Our solution is to replicate the motion detection algorithms at the CamShield, and then trigger the smart camera's motion detection by emulating the movement inside the shield.

We prototype a complete CamShield system. The hardware of the CamShield is assembled with commercial components and a 3D-printed case. The CamShield App is developed for smartphones to decrypt and render videos in real-time. Our evaluation covers five commercial off-the-shelf (COTS) smart cameras from different manufacturers. The CamShield system preserves most of their original features while making them trustworthy.

This work makes the following contributions:

- We propose and practice using physical isolation and function replication to achieve trusted computing. Its merit arises in situations where hardware replication cost is not the major issue.
- We design and implement the CamShield system. It is a bolt-on solution that turns a general camera into a trusted one without incurring hardware and software modifications.

2 Background and Related Work

2.1 ROI-based Visual Privacy Protection

Digital cameras are prevalent in both public and private places. People are concerned about whether their sensitive information, such as the face, behaviors, properties, *etc.*, would be captured and abused by the camera's owner or operator for

fun, profit, or censorship. One direct way to protect visual privacy is to blur the entire video stream, but it would cause information loss and reduce the utility value of the video. This is the trade-off between privacy and functionality. An elastic way is to use partial protection [50], where only the sensitive areas are protected and the rest are open for functional use.

The first step of partial protection is to identify the region of interest (ROI), which represents the sensitive areas of the video/image. An example in common is the face, but in general, ROIs are quite subjective. ROI detection usually relies on computer vision (CV) techniques, such as traditional handcrafted filters and recent deep neural network detectors. Some approaches utilize additional sensors to detect special ROIs. For example, thermal image sensor [67] and biological signal sensor [43] are used to differentiate the human face from the background.

The step after ROI detection is to protect the ROI areas. Blanking, blurring, pixelation, replacing to an object, *etc.*, are common methods used to reduce the information the ROI contains to protect privacy [50]. ROI areas can also be protected with encryption [33]. ROI encryption is lossless and thus retains raw information, which allows authorized entities to access the complete video.

We follow the above ROI-based protection framework to design CamShield. Related work can be classified into the following two categories.

2.1.1 Pre-capture Protection

The visual content is protected before being captured by the smart camera. This is usually achieved by intervening the protection system in the original imaging system.

One way is to use a computational visual sensor, whose pixels can be flexibly controlled. Fernández-Berni *et al.* [35] show that once ROIs are detected, the sensors can be configured to obfuscate the areas. Another way is to use optical filters. Tan *et al.* [54] design optics to blur the scenes to a level to destroy sensitive information but retain the detectability of the human face. Pittaluga *et al.* [48] attach a small lens in front of the camera to defocus and blur sensitive information. Anonymous camera [67] utilizes an additional thermal camera to detect the face and then uses a programmable optical filter to block the areas in the scene. Kitajima *et al.* [43] keep the camera defocused and use a pulse wave sensor to detect human faces based on the color variation due to heartbeats.

CamShield is similar to the above approaches in that it also performs ROI-protection prior to the digitization of the smart camera. However, CamShield pursues a trustworthy and complete system while they mainly focus on specific techniques and components. For the functional aspects, computational sensors are experimental and not suitable for commercial devices. Optical filters lack flexibility and cannot easily be modified for different ROIs. Further, their protection schemes irreversibly destroy ROIs or even the entire frame, while the ROIs of CamShield can be recovered after decryption.

2.1.2 Post-capture Protection

The visual content can also be protected after the camera's digitization. Many discussions assume the camera system is trustworthy and focus on other aspects of the privacy protection problem [50, 51, 64], *e.g.*, computational overhead [65], timing channel [45], *etc.*

Instead, we are concerned about the assumption and want to make smart cameras' digital world secure even if they are found in the hands of attackers. This is a challenging problem since smart cameras are subject to various attacks in practice. The security flaws lying in the camera's firmware, operating system (OS), processing software, and network protocols might all be made use of by attackers to access the raw video content or even compromise the whole camera system.

A potential solution is trusted computing. It is based on isolation and verification techniques to ensure that specific computing components cannot easily be tampered with. Early work takes advantage of the Trusted Platform Modular (TPM) chip to build cameras [62, 63]. TPM allows for secure boot and has a sealed key for hardware encryption, but it does not protect the running OS and video processing components. Hence it has a weaker threat model. Apart from the above, little work has been done to realize trusted cameras. This is because if the OS/driver managing the raw data from the visual sensor is compromised, little can be done in the software to protect the video content. In fact, achieving trusted peripherals I/O is a longstanding problem in trusted computing. Some approaches rely on virtualization for isolation but assume trusted hypervisors [29, 60]. Hardware TEE is free of the assumption but is limited to specific lightweight functions such as the control path [26, 44], the display UI [46], the keyboard [32], *etc.* The complexity comes from the large code base of the camera driver and the video processing stack, *e.g.*, OpenCV, TensorFlow, *etc.* It is challenging to cover all of them in the trusted computing base (TCB) [52].

An alternative approach is to use dedicated hardware, such as FPGA [37], embedded processors [28, 49, 61] as a separated and isolated middle layer to perform the video processing and protection tasks before delivering the content to the camera's host system. CamShield is similar in that they all perform ROI processing in independent hardware to isolate sensitive information from untrusted software. However, CamShield is different in that it replicates the imaging and processing components of the smart camera. It renders physical and hence likely stronger isolation. Importantly, CamShield is a bolt-on solution and compatible with legacy camera devices.

2.2 Screen-to-camera Channel

Screen-to-camera communication is a form of visible light communication (VLC). It transmits data from screens to cameras [42]. At a high level, one device displays QR-code-like visual content on the screen, and the other uses its camera to capture and decode the QR-code stream. It has become a popular

way to bridge devices that cannot be conveniently connected by traditional means [42]. Recent research [38, 40, 57, 58, 66] improved the screen-to-camera communication primarily on throughput and invisibility.

CamShield uses VLC to build the one-way data path between the CamShield device and the shielded smart camera. Its VLC design is inspired by existing work, but due to its unique scenario, it differs from them in many aspects. It does not pursue invisibility [57] and has a fixed field of view, which allows for simpler designs in some aspects. On the other hand, it needs to handle color dispersion, high compression noise, and at the same time achieve high and stable throughput with a low camera sampling rate. These technical problems are unique and have not been explored.

3 Threat Model

The overall goal of CamShield is to preserve the functionalities of the smart camera and protect its sensitive visual contents from being leaked.

As it has been reported, we assume a remote attacker may leverage some loopholes to compromise the network [24] and the cloud service [14] of the smart camera. For a powerful attacker, we assume it might take full control of the camera [1], including accessing and modifying the captured video.

As shown in Figure 1, we assume the CamShield device is correctly installed on the smart camera. We would like to use the CamShield to protect the information of the raw videos on the owner's demand. The raw videos are videos captured by the smart camera as if it is not shielded by the CamShield device. We want to ensure that the authorized parties, *e.g.*, the owner's CamShield App, can get full access to the raw videos, and the unauthorized parties, *e.g.*, the attackers, third-party cloud, will be regulated by the privacy policies issued by the device owner. We assume there will be a trusted initial setup phase where the CamShield device and CamShield App can exchange keys securely, *e.g.*, a user buys a new CamShield device and pairs it with the CamShield App.

We assume that the attacker does not have physical access to the CamShield device. A physical attacker can tamper, alter, and disable software and hardware components. We note that software attestation and solid manufacturing can increase the complexity of such attacks, but both are out of the scope of this paper. A physical attacker at close proximity might attempt to perform side-channel attacks to gain credentials, which is also out of the model. The attacker having direct sight on the smart camera might attempt to use optical tools, *e.g.*, laser projector, to interfere with the CamShield's operations. Optical shielding, *e.g.*, curtain, can overcome such attacks. We note that the CamShield can resist simple physical attackers such as copying video content via common data interfaces like USB, which it simply does not have.

We assume the way the owner consumes the raw content is secure. We assume that the attacker does not compromise the CamShield App. The CamShield App represents an application of a general trusted computer. It can be but does not have to be a smartphone app like our current implementation. For example, it can also be installed on a trusted desktop with strong security protection. A trusted computer might have different implementations. In the harshest scenario, where any network attacks might occur, a program can still be executed with hardware-based TEE [32]. We also assume the attacker cannot spy on the monitor to view the raw video, *e.g.*, via social engineering and optical reflections [25].

We do not handle denial of service (DoS) attacks, *e.g.*, the compromised smart camera refuses to stream videos, since they do not hurt privacy. We also do not handle side channels of the network traffic, *e.g.*, inferring activity through event timing [45], which can be eliminated by injecting dummy event notifications.

4 Overview

The hardware and software architecture of the CamShield device is shown in Figure 2 and Figure 3, respectively. The rightmost camera in Figure 2 is the smart camera to be secured. It is denoted as a sink camera, *i.e.*, there is no information flowing from the sink camera to the CamShield device.

As shown in Figure 2, the hardware of the CamShield device consists of five components: visual sensor, processing unit, screen, lens, and LEDs. The leftmost one is the visual sensor of the CamShield. It takes the place of the visual sensor of the sink camera and captures video frames to the memory of the processing unit. The processing unit manages the visual sensor to its left and the screen to its right. It is also responsible for executing image/video processing algorithms, encryption, and communication protocols. The screen is used as the output interface to convey encrypted video streams from the CamShield device to the sink camera. In front of the sink camera, we put a lens to focus the sink camera on the screen. There are two LEDs connected to the processing unit. The inner alarming LED is used to trigger the motion detection mechanism of the sink camera. The outer Msg (message) LED is used to inform the owner about the status of the CamShield. Finally, all of the above components are packed into a 3D-printed shielding case, which can be attached to a general smart camera through the adjustable seizers.

The overall software workflow of the CamShield device is shown in Figure 3. The CamShield first enforces typical ROI-based privacy protection on the captured video stream. We adopt standard CV modules to detect the ROI areas of the raw videos. While compressing the raw video, the encryption module encrypts the video content belonging to the ROI areas. Then, the CamShield encodes the encrypted and compressed video stream into VLC frames, which are filled into the frame buffer and displayed on the CamShield screen. The VLC frames are captured by the sink camera and transferred to the CamShield App over the network. The owner uses the

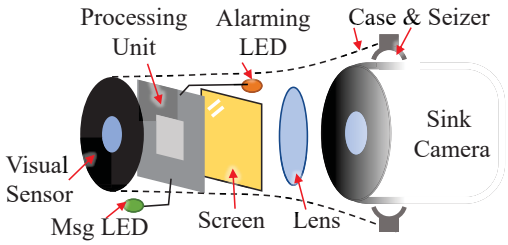


Figure 2: **Hardware Structure of the CamShield Device**.

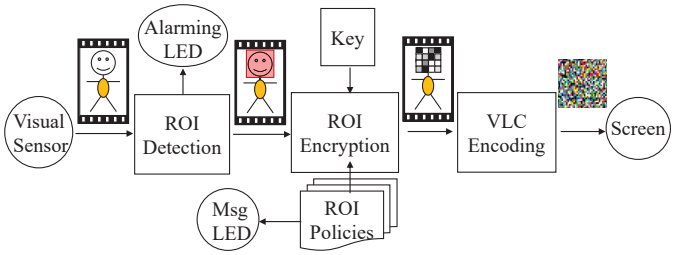


Figure 3: **Software Workflow of the CamShield Device**.

CamShield App with the key to retrieve the raw video streams captured by the CamShield, which basically follows a reverse process of encryption and VLC encoding.

In the following sections, we introduce the VLC data path in Section 5, the ROI-encryption and configuration in Section 6, and the compatibility designs in Section 7. We discuss other design choices in Section 8.

5 One-way VLC Data Path

The CamShield device is isolated from the network for security considerations, but there must be a way to transfer its video stream to the owner. From the security perspective, any two-way connections such as Bluetooth, USB cable might bring breaches to the isolation. Further, as we also want to secure smart cameras already in use, the installation must be simple enough, better without hardware and firmware modifications. CamShield takes advantage of the visual sensing capability of the sink camera to construct a one-way VLC data path that meets all these needs.

5.1 VLC Overview

At a high level, the VLC works like scanning a QR-code stream. An example of the VLC frame is shown in Figure 4 (a), which looks like a colored QR-code. The CamShield chops the captured video stream into sequential pieces and loads them into VLC frames (see Section 6). VLC frames are displayed sequentially on the CamShield's screen and captured by the sink camera. Through decoding the captured VLC frames, the loaded video stream can be extracted.

To formally describe the VLC design, we use the following notations. A VLC frame for transmission *i.e.*, txFR, is a matrix of pixels, with each matrix element representing the color of the pixel. Our VLC scheme uses different colors to represent bits. In practice, a single pixel is usually too small to be robust to noises, hence neighboring pixels are grouped to illustrate the same color. We use a k×k square block to group them. Pixels in the same block have the same color. A subscript n is used to denote the sequence of the VLC frames, *e.g.*, txFR_n is the n-th transmission frame. Similarly, rxFR denotes the VLC frame captured by the sink camera. rxFR is related to txFR, but is distorted and even merged from multiple txFRs. Extracting information from rxFRs is the major challenge of the VLC design. While there are plenty of existing VLC implementations, we face unique challenges specific to the CamShield's unique scenario.

5.2 Calibrating Distortions

To perform VLC decoding, the initial step is to determine the area of txRF in rxRF. Then, the location of blocks can be determined and their colors and hence bits can be extracted. In practice, this process is not straightforward due to various distortions. For example, most VLC systems are subject to perspective distortions [47], which transform the txFR to a trapezoid when viewing the screen from an angle. The CamShield's screen is fixed towards the sink camera and thus is free of them. However, it is subject to another two types of distortion, which are rooted in the compact form factor of the shield.

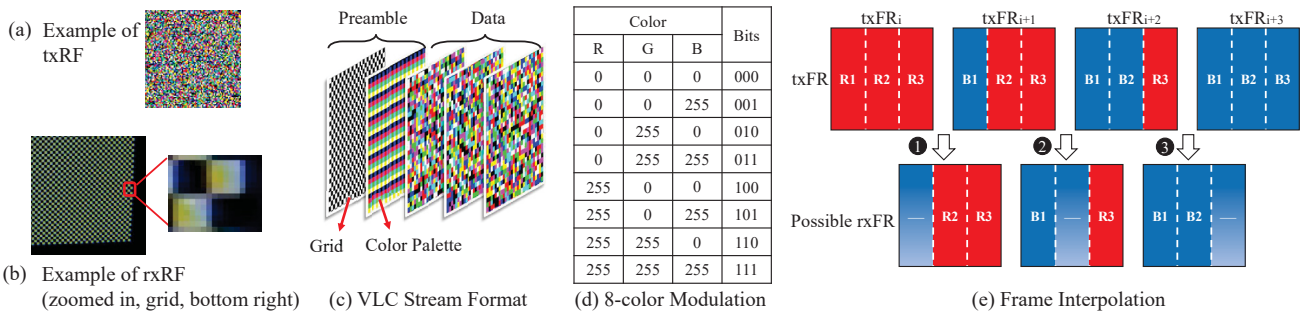
5.2.1 Lens Distortion

Smart cameras are designed to capture scenes centimeters away, but the screen of CamShield is placed only several centimeters in front of the sink camera. This distance is below the minimum object distance of most smart cameras [7], and thus leads to significant blur [1]. Similar to wearing glasses, we add a lens to refocus it on the screen, as shown in Figure 2. A side effect of adding a new lens to an imaging system is the lens distortion. As shown in Figure 4 (b), straight lines of pixels bend outward from the center of the image.

5.2.2 Chromatic Distortion

Another subtle problem of the new lens is that the distortions differ among colors. The reason is that the lens has different refraction ratios for different colors of light. As a consequence, the focal plane or the magnified size slightly differs among colors [4]. The more the incidence location from the center of the lens, the more color dispersion can be observed. As shown in the zoomed part of Figure 4 (b), the blue and red components are shifted off to the right and left of the original white square. Quite similar to the TikTok icon. This adds complexities in determining the precise locations of the blocks, as their locations are different when displaying different colors.

[1] like our eyes, the distance that they see clearly has a wide range but does not cover the nose.



Color			Bits
R	G	B	
0	0	0	000
0	0	255	001
0	255	0	010
0	255	255	011
255	0	0	100
255	0	255	101
255	255	0	110
255	255	255	111

Figure 4: **VLC Design**. The CamShield device uses the screen-to-camera VLC to convey encrypted videos to the sink camera. A VLC frame is shown in (a), which uses the 8-color modulation scheme in (d). The Grid and Color Palettes preamble frames in (c) are used to calibrate various distortions and biases. The VLC is subject to the rolling shutter effect, which merges transmission frames (txRF) if they are captured (rxRF) during screen refreshes. We use frame interpolation in (e) to minimize the difference between adjacent frames to amortize throughput penalty.

We note that the above two types of distortion are so unique that they have not been addressed in other VLC systems. Since the pixel blocks must be located at the pixel level, existing distortion adjustment methods cannot be used due to inefficient accuracy. For example, the anchor fields [12] used for adjusting the perspective distortion are not applicable, since pixels are distorted non-linearly in space. Model-based calibrations [68] are not accurate enough either, since the general mathematical distortion model does not perfectly fit the actual distortions.

Our calibration method is motivated by wireless communication, where distortions are measured by a known sequence - the preamble. The distortion is quantified by comparing the received preamble with its ideal values. We adopt a similar method by introducing a preamble frame prior to the VLC data frames. As shown in Figure 4 (c), the preamble frame is a black and white chessboard with a grid size equal to the block size. We use CV methods to accurately localize every black and white block in the rxFR and their locations are recorded. To handle chromatic distortion, the above process is performed in the Red, Green, and Blue channels independently. These locations are used to index the blocks in the data VLC frames to extract colors and thus the information that they convey. Note that since the CamShield is fixed to the sink camera, the block locations are stable, hence the preamble frames are played only in the initialization process after the device boot. Optionally, they can be inserted once for a while to increase stability.

5.3 Boosting Throughput via Color Modulation

Even if the color blocks are perfectly located, it remains difficult to convey High-definition (HD) videos, which has been the default resolution of current smart cameras. For streaming 720p HD videos, the minimum required data rate is 768 kbps when using the HEVC (H.256) compression [21]. As the sampling rate of the sink camera is fixed at 20 fps or 15 fps in most COTS smart cameras, a single rxFR must at least contain 38.4 kb. For a quick comparison, the capacity of the densest standard QR-code is 23.6 kb [12].

A typical way to increase the capacity is to use multiple colors in one block to represent information. If the color is chosen from N^c different colors, $\log(N^c)$ bits are contained in one block. The modulation scheme determines how the colors represent bits. An 8-color modulation scheme is shown in Figure 4 (d). However, N^c cannot be ultimately increased due to the following noise sources in rxRF: First, the sink camera adopts multiple non-configurable schemes to auto-adjust the contrast, brightness, saturation, *etc.*, of the captured video. These schemes work as blackboxes and bring color biases to rxFR. Second, some cameras we tested have the vignette effect [22], where the center area of rxFR is brighter than the periphery areas, which affects the block color spatially.

We note that while the auto-adjustment and the vignette effect are model-dependent, they can be calibrated with another preamble frame. As shown in Figure 4 (c). This frame is filled with identical color palettes, showing the colors used by txFR. These palettes are used as the reference for decoding. *i.e.*, choosing the closest color in the palette for determining the color of blocks in rxFR. Since the palettes are piled up on the entire screen, the bias caused by the vignette effect can be compensated as well.

5.4 Amortizing Rolling Shutter Effect

The rolling shutter effect is caused by unsynchronized line exposure of the CMOS sensor array [40]. When an rxFR is taken during the transition period between txFR_i and txFR_{i+1}, the rxRF will be a merge of txFR_i and txFR_{i+1}. Some of its portions are from txFR_i, as these lines expose first, while others are from txFR_{i+1}. Since the camera exposure and the screen refresh are not finished instantly, a large portion of rxRF is a merge from both txFR_i and txFR_{i+1}, which complicates the color extraction [66].

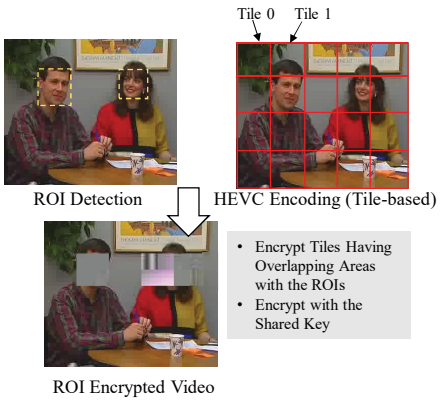


Figure 5: **Example of ROI-Based Video Encryption**. The frame is encrypted in units of tiles. The tiles overlapping with the ROI are encrypted and their visual contents are destroyed.

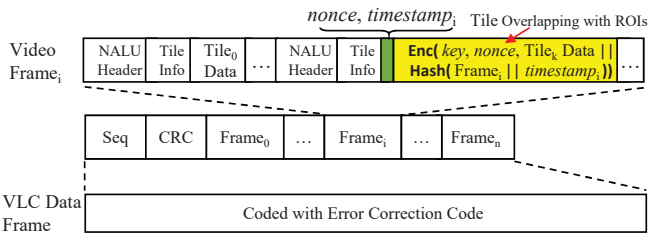


Figure 6: **VLC Frames for End-to-end ROI-Encryption and Transmission**. Video frames captured by the CamShield device are encrypted based on the ROI information and in units of a tile. Multiple encrypted video frames are packed into a VLC data frame for transmission. The VLC frame is coded with error correction codes to resist the noise of the VLC channel.

Exiting work proposed several solutions. A very recent work, ERSCC [66], achieved the best-reported results, *i.e.*, log(4.36) bits per block. It separates the colors in merged areas but its throughput cannot be further improved as the separation depends on a limited number of encoding colors. An alternative solution is to abandon the merged areas and use the error correction code to automatically make use of the information in the merge-free areas [58]. In this way, the merged frames contain less information while the merge-free frames contain more. When putting them together, all of them adaptively contribute to the overall capacity. However, due to the slight rate misalignment of rxFRs and txFRs, the merged frames occur in periodic bursts. Each burst might last for several seconds or even more. The actual capacity during such a worst-case period is very low, which results in video streaming latency and usability frictions.

The application scenario of CamShield has stringent requirements for both throughput and latency. To make use of VLC for such a purpose, we handle the rolling shutter effect in a distinct way: instead of splitting the merged areas in every frame or leaving them alone in a burst, we propose a hybrid scheme that intentionally distributes the merged areas evenly among all the rxFRs to amortize their negative impact, and at the same time, incorporates more colors in a block to maintain the capacity.

The key insight is that the merged areas occur only when the content of consecutive txFRs are different. If we keep the consecutive txRFs as similar as possible, then the merged area will be reduced. To do so, our approach takes advantage of the asymmetric frame rate of txFR and rxFR. As mentioned earlier, the rxFR rate is usually up to 20 fps, and the txFR rate of a typical LCD screen is around 60 fps. That is to say, the frame rate of txFR is at least 3 times that of rxRF. As shown in Figure 4 (e), instead of refreshing the entire txFR, which results in a large merged area if the rxRF encounters the refresh period, our method is to gradually change, or interpolate txRFs by making any adjacent txRFs differ by 1/3 parts. In this way, for any rxRF, at most 1/3 is affected by the rolling shutter effect, while the remaining 2/3 is unaffected. The overall capacity is stable at 2/3 of the merge-free case (it relies on an inter-frame coding scheme explained in Appendix A). We also note that the capacity can be further increased if high refresh rate screens are used, *e.g.*, 120 Hz LCDs are becoming prevalent in the latest smartphones and PC monitors.

6 ROI-based Video Encryption

CamShield adopts a typical ROI-based video encryption framework. It first uses standard CV methods to identify ROI regions, and then encrypts them. We refer the reader to the ROI-based video encryption schemes [33] for more technical details. This section describes the end-to-end video transmission protocol built upon the ROI encryption and the interface used to configure the ROI-based privacy policies.

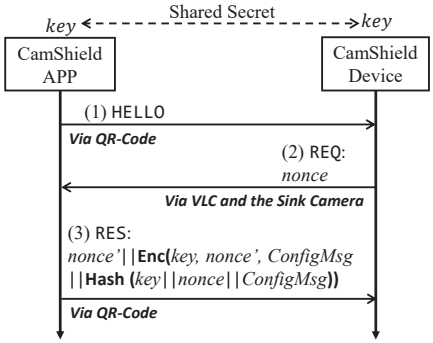
6.1 End-to-end Video Delivery

CamShield uses encryption to secure the information between the CamShield device and authorized parties, *e.g.*, the CamShield App. To fulfill our security goals, we adopt ROI-based encryption instead of encrypting the whole frame. ROI encryption allows the owner to flexibly specify the areas they would like to protect. It leaves room for the owner to trade off utility and privacy, *e.g.*, disclosing more information to the cloud might gain a better user experience.

The encryption is based on a pre-shared secret, *i.e.*, a symmetric *key*. The *key* is shared through a trusted initialization step. For example, the *key* of a CamShield device is sealed in its hardware and is printed on a disposable paper shipped with the new device. The device owner inputs the key into the CamShield App to authorize it to access the encrypted information.

Index	ROI	Invert	Alg.	Time	Enable
1	all	F	/	all	F
2	none	F	/	all	F
3	face	F	mobile net	all	T
4	body	F	opencv	all	F
5	text	F	opencv	work days	F

(a) Table of ROI Entries



(b) ROI Configuration Protocol

```
{
    "config": {
        "ROI Policy": {
            "Index": 3,
            "ROI": "all",
            "Invert": false,
            "Algorithm": "moiblenet",
            "Time": ["Mon","Fri"],
            "Enable": true
        }
    },
    "timestamp": {
        "Value": 1391141532000,
        "Time Zone": "GMT008:00"
    }
}
```

(c) ROI ConfigMsg

Figure 7: **ROI Configuration Interface**. The CamShield device allows the owner to flexibly configure the ROI policies in (a) with the configuration message *ConfigMsg* in (c). A challenge and response protocol in (b) is used to authenticate *ConfigMsg*.

Modern video coder-decodes, such as HEVC (or H.265), divide a video frame into uniform rectangles, *i.e.*, the tiles. HEVC provides a clean interface - the Network Abstraction Layer (NAL), to pack tiles into NAL units (NALU) for general network transmissions. CamShield's video encryption and transmission take advantage of this feature. The structure of VLC frames from the CamShield device is shown in Figure 6. We highlight several points: 1) Multiple video frames after HEVC compression are packed into one VLC frame. 2) Each video frame contains multiple encrypted and unencrypted tiles. Any tile overlapping with the ROI is encrypted with the *key* and a randomly-generated nonce. The nonce is inserted between the NALU header and the NALU data. The remaining tiles are left unencrypted for the third party to make use of. 3) An ordinary HEVC player can still render the encrypted tiles but the visual content is destroyed (see Figure 5). To identify the encrypted tiles in the CamShield App, we use a reserved bit in the NALU header as the encryption indicator, and set it to "1" for encrypted tiles. 4) The data of the whole frame and its timestamp are hashed and appended to the encrypted content for integrity and authenticity.

6.2 ROI Policy Configuration Interface

CamShield allows the owner to flexibly configure the privacy policies to specify how its ROI-based encryption works. An example of the policy entries is shown in Figure 7 (a), which looks similar to the Access Control List (ACL) rules of firewalls. Each row of the table defines the CamShield's protection behavior of the associated ROI.

The 'Index' and 'ROI' uniquely identify the policy of that ROI. The ROI detection can be conducted by multiple methods, which are specified by the 'Alg.' column. The 'Invert' column specifies the ROI region as the detected region or the corresponding complement region, *e.g.*, the body or the background other than the body. The 'Time' column and 'Enable' column determine when and whether this policy entry should be enabled. Multiple ROI policies can be enabled simultaneously, where the ROI region is the union of the regions of all the enabled ROIs.

To allow the owner to configure ROI policies, the CamShield preserves a convenient and secure input interface. It not only allows for configuring devices out of the cabling range but also retains physical isolation. The idea is to leverage the CamShield's visual sensor to scan the information displayed on the CamShield App. Specifically, the owner uses the app to generate a QR-code containing the configuration message, and shows it to the CamShield to configure it. An example of the configuration message for modifying the active period of the face ROI is shown in Figure 7 (c).

As the configuration interface is very sensitive, we use a typical challenge and response protocol shown in Figure 7 (b) to protect it against spoofing and replay attacks. To start the configuration, the CamShield App first displays a QR-code showing the HELLO message. Once the CamShield receives the HELLO, it sends a challenge message REQ containing a nonce to the app through the VLC data path. The app responses to the challenge with another QR-code called RES, which contains the hash of the nonce and the shared key. Since the key is only shared with the authorized CamShield App, the CamShield authenticates the app by verifying the hash. If the authentication is correct, the CamShield accepts the encrypted configuration message, *i.e.*, ConfigMsg, appended in the RES message.

In addition to the configuration interface, CamShield uses a MsgLED to physically notify the owner about its working status, *e.g.*, whether the Face ROI is enabled. This interface is used to avoid possible breaches due to the owner's subjective factors. For example, if the owner forgot to enable the ROI (for some reason it was disabled), then the CamShield can do nothing to protect privacy. Similar usability designs are essential to practical protection systems [39]. The CamShield prototype will allow us to continue the study on its usable privacy and security issues in future work.

7 Compatibility Designs

The major usability goal of CamShield is to preserve the functionalities of the shielded smart camera. The primary function of smart cameras is video previewing We further identify two other features that are valuable to users: motion detection and cloud video analysis. This section describes our designs to preserve them after introducing the CamShield.

Real-time Preview: Current COTS smart cameras achieve real-time video preview mainly in two ways: via direct IP connections or centralized servers. The majority of smart cameras in the market use the first solution. The smart camera stores videos locally and streams preview to the companion app via standard protocols such as RTSP. [2] With proper configurations [20], the CamShield App can directly receive their videos. The representative product of the centralized solution is the Google Nest camera. Its videos are encrypted and uploaded to the cloud storage. When the user requests, the cloud distributes the video to the native app. Since directly accessing these videos is hacky and ad-hoc [8], we propose to download and copy the video clips from the native app to the CamShield App to decode. We note that this process does not involve manual overhead since it can be automatized by tools such as Tasker [19].

Emulating Motion Detection: Usually, the smart camera detects motion locally, and then sends notifications (including a video clip or figure showing the detected event) to the companion app. The problem is the sink camera can only capture the VLC streams rather than the real scenes (see 1st and 2rd design choices in Section 8). Our approach follows the replication idea. The CamShield device detects the motion by analyzing the video stream, and then stimulates the sink camera to generate motion notifications to the Internet. To save the screen resource for VLC, it blinks an alarming LED inside the shielding case to emulate the motion event. As the VLC streams shown on the screen might also be treated as motions and cause false alarms, we leverage the virtual fencing feature of the sink camera to exclude the screen showing the VLC streams from the motion detection areas.

Cloud Analysis: In centralized solutions, the video streams from the sink camera are uploaded to the cloud storage, but the content is still the VLC streams. As the VLC decoding algorithm is publicly known and no encryption is applied on VLC frames, it is possible for the cloud servers to implement the VLC decoding to analyze the videos from the CamShield. Since current cloud servers have not supported this feature, we make a workaround: the CamShield App decodes the VLC streams and uploads the ROI encrypted videos to the cloud for video analysis.

[2] The solution might also have servers responsible for setting up connections and relaying streams if necessary [16].

8 Design Choices

This section discusses the design choices behind CamShield.

VLC v.s. Bare-metal Videos: Another way to deliver the video stream to the smart camera is to directly play the recorded video on the CamShield screen. This bare-metal approach has good compatibility since it directly delivers similar scenes as the ones not using the CamShield to the camera. However, due to noise, the video quality of the re-recorded video is much lower. Further, the re-recorded video does not preserve encryption, *i.e.*, if the video is ROI-encrypted, even the authorized entities cannot decrypt and recover the raw video, affecting the usability of the smart camera.

Decoding VLC at App v.s. at Smart Camera: The original functionalities of smart cameras can be retained by decoding the VLC streams locally to obtain the video content from the CamShield and then feeding the content into the camera's original processing pipeline. However, it requires firmware modification. This is possible but practically hard. To our knowledge, very few cameras have open firmware that can be modified [3].

Encrypting Video v.s. Encrypting VLC: The VLC data path is like an end-to-end tunnel from the CamShield device to the app. An alternative way to secure the video data is to encrypt the entire VLC payload. The two approaches are not exclusive and can be used simultaneously. The reason we choose unencrypted VLC streams is to leave an interface for cloud analysis (recall Cloud Analysis in Section 7).

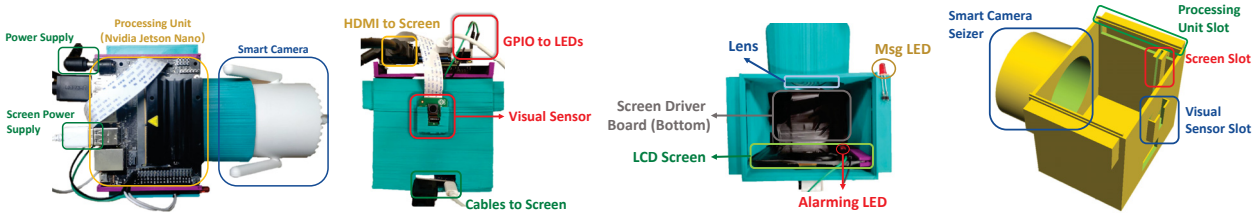
Motion Notification via Stimulation v.s. via VLC: Even though the CamShield device can send notifications to the CamShield App via the VLC data path, the app must always be active to decode the VLC streams in order to get notifications in time. This is not a good choice for devices with power constraints, *e.g.*, smartphones.

9 Security Analysis

Many reports suggest that the videos recorded by the smart cameras might be transparent to attackers due to unsecured network connections [24], malicious cloud overlays [14], or even compromised camera systems [1]. However, with CamShield, the private ROIs contained in the smart camera's videos are still confidential. This is because they are encrypted by the shield rather than the smart camera. The attacker cannot decrypt the content without the key. Since the CamShield device is isolated from the smart camera and the network, it operates as it is expected to and naturally resists many types of network-based attacks. In the following, we discuss several non-trivial attacks.

Video Manipulation Attack: The attacker with full control of the sink camera may first decode the VLC stream, alter some parts of the video, and regenerate the VLC stream to

[3] A hacked case is https://github.com/TheCrypt0/yi-hack-v4



(a) Top View with Installation (b) Front View (c) Top View Inside (d) CAD Model of the Shield Case

Figure 8: **Hardware Prototype of the CamShield Device**. (a)(b)(c) show different hardware components in Figure 2. The top view (a) shows how the CamShield device is installed on a COTS smart camera. (d) is the 3D-model of the case.

lead the owner to misconfig ROIs. For example, if the ROI policy was configured to 'none', *i.e.*, the raw video is not encrypted. The attacker can modify the video content to forge some ROIs as if they have been protected, hence fooling the owner that the camera has been protected. CamShield involves two mechanisms to defend against this attack. First, the CamShield device has a Msg LED to physically notify the protection status, *e.g.*, blinking to indicate the 'none' policy. Second, all the video frames from the CamShield have an integrity check whether there is ROI or not (Figure 6). Thus, the CamShield App can identify whether the video content is modified or not.

Video Replay Attack: Similar to the video manipulation attack, this attack assumes the ROI protection was turned off by the owner for some reason, and the attacker does not want the owner to be aware of it being turned off. The attacker may record the VLC stream when the ROI protection is enabled, and replay the record stream to the owner when the ROI protection is disabled [34]. Even though the recorded video is not tampered with, the replay can be detected since the timestamp is no longer valid.

Multi-device Colluding Attack: An advanced attacker might be able to control other co-located smart devices to compromise the CamShield. For example, if the attacker can control the co-located TV within the scope of the CamShield, it can initiate the CamShield configuration process by displaying or replaying QR-codes on the TV screen to modify privacy policies. Despite the high requirements, the nonce and timestamp in the RES message in Figure 7(b) are used to defend against unauthorized configurations.

ROI Failure Attack: The ROI-based encryption allows the owner to adjust the privacy policies. It relied on CV algorithms to identify ROIs, whose accuracy, while in many cases is comparable to the human cognitive system, is not 100%. A patient attacker can wait for the failures, *e.g.*, an ROI detection miss, to partially gain the sensitive information. The ROI-encryption cannot strictly eliminate this attack. Possible defense approaches are to keep CV algorithms up to date (see discussion in Section 12) or use full encryption (sacrificing functionality for security)

10 Implementation

We describe the implementation of the CamShield device and the CamShield App below.

10.1 CamShield Device

Hardware. We prototype the CamShield device with COTS components and a 3D printed case. Nvidia Jetson Nano board [10] is used as the processing unit. The visual sensor is Raspberry Camera V2.1, featuring 60 fps@720p. The visual sensor is directly connected to the CMOS Serial Interface of the Nano board. The screen is SHARP LS029B3SX02 1440×1440@60fps, which is driven by an HDMI driver board. We use a lens of 60 mm focal length to focus the sink camera to the screen. Two LEDs are connected to the GPIO pins of the Nano board as the alarming LED and Msg LED. As shown in Figure 8, the above components are attached to a 3D printed case. The CAD file of the case is shown in Figure 8(d). The LCD screen is placed in a movable slot so that it can be adjusted to fit the focal plane of the smart camera.

Software. OpenCV v4.1.0 and Tensorflow v1.14 are used to implement the ROI detection. For fast prototyping, we used shipped detectors. HaarCascade [56] and SSD+MobileNet [18] are face detectors. Histograms of Oriented Gradients [31] is used to detect pedestrians and Scene Text Detection [69] is used to extract regions containing text. The ROI detection module is also reused for motion detection (Background Subtractor MOG2 [70]) and decoding configuration message from QR-code decode (pyzbar 0.1.8 library [12]). The ROI encryption is based on an open-source HEVC codec [55], which outputs the encoded video tile as a Network Abstract Layer Unit (NALU). We adopt AES-GCM for encryption. For VLC encoding, we utilize the Real-time Transport Protocol (RTP) as an intermediate helper layer to pack the NALU stream first and then to the VLC data frame shown in Figure 6. To counter the errors in the VLC channel, Reed-Solomon (RS) codes are used for error correction (Python reedsolo library [13]). To avoid burst errors, we use Python commpy library for data interleaving before RS encoding. Bits are modulated by color modulation. To achieve real-time video rendering at 60 fps, the encoded frames are piped to GPU via the Gstreamer pipeline [9].

10.2 CamShield App

Currently, the CamShield App works with smart cameras supporting RTSP streaming protocol. The app is responsible for VLC decoding, ROI decryption, and video rendering. We implement it on Xiaomi Mi Mix 2S with Qualcomm Snapdragon 845 and Android 10.

The VLC decoding first locates all blocks in the preamble through the C++ robust corner detection algorithm [36] with the Java Native Interface (JNI). For data decoding, it calls the JAVA ReedSolomon library in zxing [6]. The ROI decryption is a reversed process of the ROI encryption. Firstly, we unpack the VLC frame to retrieve the NALU stream. Then, for each video frame in the NALU stream, we find out all encrypted NALUs through the flag in the NALU header. Next, we verify the integrity and time of the frame by checking the hash. Finally, we obtain the whole frame (ROI + non-ROI) through the HEVC decoder. To realize real-time video display on smartphones, we implement a pipeline scheme to speed up the whole workflow (see Appendix C).

11 Evaluation

This section evaluates the VLC performance, the ROI encryption, and the compatibility designs.

11.1 VLC Evaluation

This subsection evaluates the VLC performance and the effectiveness of the design components.

Overall Performance: We evaluate the performance of the VLC data path on different smart (sink) cameras shown in Table 1. Due to the impact of resolution, focal length, automatic image tuning, *etc.*, the properties of the VLC channel vary by camera model. Thus, we adjust the default VLC parameters (see Appendix B) for different camera models to achieve their best performance. Figure 9 (a) shows the highest throughput of each camera model. The results show that XiaoMi, Ezviz, and YI models could achieve a throughput of 300 to 500 kbps, meeting the requirement of streaming 720p@10 fps video. The Hikvison model is able to support 720p@20 fps, since it has higher resolution and allows manual configuration to disable the image auto-tuning, *e.g.*, white balance, saturation, exposure, *etc.* The results show that the VLC data path can fulfill the requirement of streaming HD videos.

Grid Preamble for Distortion Calibration: We evaluate the effectiveness of calibrating lens and chromatic distortions. We compare our calibration scheme with a simple calibration scheme that uses the same block locations for the three RGB channels. The results are depicted in Figure 9 (b). In the simple calibration scheme (combined location), more blocks are mislocated, which is obvious in the red and blue channels. Our calibration scheme (decoupled location) achieves better performance as the three channels have different distortion characteristics.

Brand	Model	Frame Rate (fps)	Resolution	Disable Auto-tuning
Hikvison	DS-2CD3T56FWDV2-I3	20	2560x1920	Support
XiaoMi	CMSXJ25A	20	1920x1080	No
Ezviz	CS-C6CN	15	1920x1080	No
YI	YYS.2919	15	1920x1080	No

Table 1: **COTS Smart Cameras Used in Evaluation**.

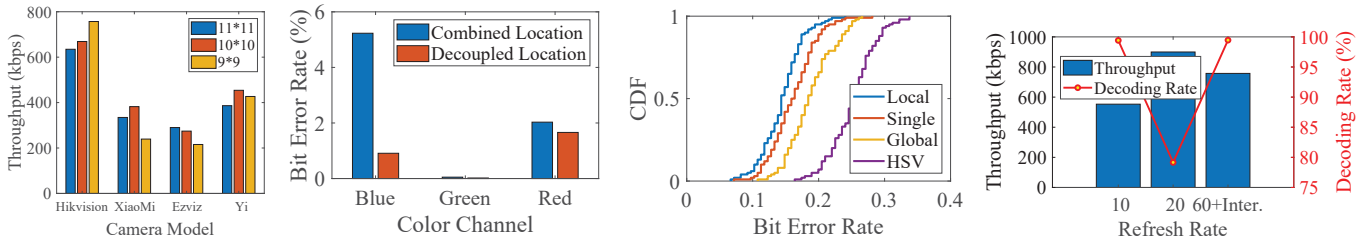
Color Palette Preamble for Color Bias Calibration: We test different schemes for mapping colors to bits. First, we choose HSV as a representative color space for directly classifying colors without the reference of the palette [59]. As shown in Figure 9 (c), its performance is not good enough. The main reason is that the hues of the colors have overlapping areas, resulting in demodulation errors. Second, as identical color palettes are piled up on the screen, we test three schemes to choose the decoding reference: 1) Single: use the closest palette; 2) Local: use the average of multiple single-palettes in a small area; global; 3) Global: the average of all palettes. As shown in Figure 9 (c), the local palette achieves the lowest error rate. It is better than the Single since it is less noisy.

Frame Interpolation: We use F_r to denote the rate at which the VLC transmitter refreshes its content (its maximum value is the CamShield screen refresh rate). To show the properties of the frame interpolation scheme, we compare it with two simple approaches: 1) F_r is half of the capture rate, *i.e.*, F_r=10 fps; and 2) F_r is the same as the capture rate, *i.e.*, F_r=20 fps. The screen refreshes at full rate in our scheme, *i.e.*, F_r=60 fps, but only 1/3 parts are refreshed in adjacent frames. Results are shown in Figure 9 (d). When F_r=10 fps, the sink camera can always capture a merge-free frame as its rate is 2 times of F_r. So its decoding rate (the percentage of correctly decoded bits) is high. However, as most rxRFs are duplicated, the overall throughput is low. When F_r=20 fps, the sink camera is almost synced with the screen, the ideal throughput would be two times of the F_r=10 fps case. However, as their actual refresh rates are slightly different, the rxRFs encounter screen refresh periodically, resulting in consecutive undecodable merged frames. As a result, the average throughput of this case is high but its throughput fluctuates, which is also reflected in the decoding rate curve. Our scheme can achieve 757 kbps, slightly less than the F_r=20 fps case, but its decoding rate is 99.5%, indicating stable performance.

11.2 Evaluation of ROI-based Encryption

This subsection shows the effectiveness of the ROI-based encryption in protecting the visual information.

Effectiveness of Content Protection: We adopt two image quality metrics: Peak Signal to Noise Ratio (PSNR) and Structural Similarity (SSIM). PSNR compares the noise level of the processed image and the original one. Larger PSNR means less noise. SSIM compares the similarity of two images. Large SSIM means two images are similar. The videos are collected



(a) Overall Throughput (b) Distortion Calibration (c) Color Bias Calibration (d) Frame Interpolation

Figure 9: **VLC Evaluation**.

Scenario	Description
meeting	Three men sit around the table for a meeting. There are two cans of drinks, a bottle of water and a set of keyboard and mouse on the table.
lecture	A man in a plaid shirt writes and draws on the white board. He occasionally looks at the camera.
office	A man in a knitted hat looks at the camera and talks to the camera, holding a whiteboard marker in his hand.

Table 2: **Scenarios of Captured Videos**.

in three public places [4]. The detailed scenario descriptions are shown in Table 2. Human face is selected as the ROI. The videos are compressed by the HEVC codec with and without the ROI-based encryption. Their ROI regions are compared with that of the original frames. Results in Table 3 show that the average PSNR and SSIM of all scenarios using ROI-based encryption decrease significantly over the No-encryption cases, meaning that the encrypted areas are quite different from the original ones.

Effectiveness of ROI Detection: The CamShield is based on CV algorithms to detect the ROI areas. For the legacy OpenCV detector (HaarCascad), the false-negative rate is around 3% in the meeting and office scenarios, and 10% in the lecture scenario. The false-positive rate is at the same level. The detention misses usually occur when the face moves. The neural network detector (SSD+MobileNet) is more robust. The false-negative rate is less than 1% and the false-positive rate is close to 0% in the three scenarios. Unlike cryptographic protection, ROI algorithm failure (false negative) can hardly be completely avoided with current CV algorithms. This is a common problem in ROI-based protection systems [41]. Our suggestion is to keep the algorithms up-to-date and also use stringent ROIs whenever possible, *e.g.*, always encrypt the full frame when the cloud video analysis is not active.

11.3 Evaluation of Compatibility Designs

This section evaluates the motion detection and cloud video analysis capabilities of the smart cameras in Table 1 after the installation of the CamShield.

[4]Collecting data in private places requires much more complex permissions, and there is no difference in justifying our performance.

Scenario	No-encryption		ROI-encryption	
	PSNR (dB)	SSIM	PSNR (dB)	SSIM
meeting	34.43	0.90	9.08	0.292
lecture	35.27	0.91	11.41	0.323
office	35.74	0.91	9.02	0.324

Table 3: **Video Quality: Original v.s. ROI-encrypted**.

11.3.1 Motion Detection

We evaluate whether the smart cameras can correctly issue motion detection notifications in time. The CamShield device deliberately generates the motion detection events for these smart cameras. The alarming LED is set to flash for 30 seconds every five minutes to stimulate the sink camera. We use the CamShield App to count the detected events and the timestamps. The test lasts 16 hours for each camera. The detection rate is calculated as the ratio of the number of received notifications to the actual number of motion events generated during the period. The results show that no camera has false alarms, and most smart cameras have 100% detection rate except that the YI model is 54%. We guess it is because the YI server may filter out too-frequent notifications.

11.3.2 Cloud Video Analysis

We evaluate the performance of cloud video analysis on ROI-encrypted videos. The scenes are described in Table 2. We utilize the CV interfaces (`describe_image` and `tag_image`) of Microsoft Azure [2] to see if the ROI-encrypted video can still be used by the cloud server to infer useful information for the owner. The CamShield App uploads the VLC decoded and ROI-encrypted videos to the server.

We first use the `describe_image` interface to generate descriptive captions together with confidence levels for the videos.Table 4 shows the results. We can see that the ROI-based encryption still preserves the context understanding of the cloud server. It just lowers the confidence level in the meeting and office scenarios by a little bit. In the lecture scene, the captions are quite different. The reason is that the proportion of the man in this scene is much smaller than the whiteboard. Thus, when the the man's face is not facing the camera, the analyzer pays more attention to the content of the whiteboard.

We further utilize the `tag_image` to analyze the fine-grained information. This interface infers multiple tags based on the input video. We use the number and content of the

Scenario	Generated caption	Confidence
meeting (w/o)	'a group of men sitting at a table'	55.71%
meeting (w/)	'a group of people sitting around a table with a computer'	45.93%
lecture (w/o)	'a man writing on a white board'	59.76%
lecture (w/)	'text'	65.61%
office (w/o)	'a person sitting at a desk'	55.09%
office (w/)	'a person holding a phone'	37.70%

Table 4:**Context Understanding with (w/) and without (w/o) ROI-encryption**.

tags to quantify how much information is preserved in the encrypted video. We first analyze the unencrypted raw videos and record all tags with confidence level higher than 50%, *i.e.*, true tags. Then, we apply two kinds of ROI-based encryption to the raw videos, 'face' and 'body'. Next, we apply tag analysis again on the encrypted videos and calculate the average detection rate, *i.e.*, the ratio of the detected true tags in the encrypted video to all the true tags.

The results are shown in Table 5. The 'Tag (All)' column indicates the ratio of detected true tags to all true tags. The 'Tag (Body)' column indicates the ratio of detected body-related true tags to all body-related true tags. Similarly, the 'Tag(Face)' column indicates the ratio of detected face-related true tags to all face-related true tags. The 0% detection rate of the two tags validates the effectiveness of ROI protection. However, as shown in the 'Tag (All)' column, the ROI-based encryption still preserves information for the cloud server. When the ROI is face, the 'Tag (Body)' column shows that some body-related true tags are still kept, *i.e.*, gesture recognition would still be possible when the face area is encrypted.

12 Discussion

We discuss the limitations of the current implementation.

Cost: The major cost of the CamShield prototype is the Nano board (100$) and the screen (50$). The Nano board can be replaced with DSP chips and a low-end processor. The price of screens will always decrease. Ideally, as the CamShield replicates the full functionalities of the smart camera, it will not be cheaper but should be on the same price level as a smart camera. We are also exploring low-cost alternatives, for example, it might be possible to reuse smartphones that are no longer in use as free CamShield devices.

Update: The CamShield relies on CV algorithms for ROI detection. When better algorithms are available, they can be updated via the policy configuration interface (Section 6.2). The binary of the CV module is first segmented and then transmitted by multiple configuration messages. We implemented a preliminary QR-code stream decoder, through which a legacy OpenCV detector (about 100 KB) can be updated in several mins. For larger modules such as neural network models (*e.g.*, a compressed `MobileNet` is about several MB), a carefully-designed high throughput scheme should be used to transmit the messages (*e.g.*, 100 kbps is achieved in [38]).

Scenario	ROI	Tags (All)	Tags (Body)	Tags (Face)
meeting	face	56.2%	15.0%	0%
	body	17.1%	0%	0%
lecture	face	89.3%	26.2%	0%
	body	61.4%	0%	0%
office	face	27.3%	3.1%	0%
	body	18.1%	0%	0%

Table 5: **Tags Detection Rate after ROI-encryption**.

For the remaining firmware (non-algorithm parts), many risks have been avoided by isolation. For vital firmware upgrade, which unlikely frequently occurs, one way is to return the device to the manufacturer/authorized parties to reload new firmware by proprietary interface and protocols.

Timestamp: Some of the security mechanisms rely on the timestamps of the Real Time Clock (RTC) time, *i.e.*, the absolute time. The CamShield's RTC is powered by the battery, which ticks even when the device is powered off. However, due to the frequency offset of the RTC's conciliator, its time keeps drifting from the accurate absolute time. As a result, a time calibration method is required. A potential solution is through the configuration interface, which already reserves the timestampe field.

HD Video Support: Higher throughput of the VLC data path is required for supporting 1080p or 2k videos. Currently, we use a square screen, but the CMOS of the sink cameras is a rectangle, hence we can adopt a rectangle screen to increase the VLC throughput. Further, high-refresh-rate screens are also beneficial in increasing the throughput.

Installation Overhead: As shown in Figure 8 (a), the CamShield device is put in front of the smart camera to fully cover it. To maximize the VLC performance, the installation must jointly adjust the lens of the CamShield and the lens of the camera to optimize the focus as well as to maximize the VLC areas. In practice, it takes us less than 5 mins to set up a new camera with a reasonable performance. For easier and faster installation, the manufacturer might provide pre-tuned space occupiers for different cameras models. Then, the user only needs to stack the CamShield, the occupier, the lens, another occupier, and the smart camera together.

13 Conclusion

Privacy and security issues of visual sensing devices have become a concern for consumers. This paper proposes an approach to use an isolated and fully functional copy of the original sensing device as a bolt-on accessory to protect sensing security and privacy. As an example, we design the CamShield system to secure COTS smart cameras while preserving their key smart features.

Acknowledgements

We thank anonymous reviewers for their valuable comments. We thank our shepherd Ardalan Amiri Sani. We thank Shauna Dalton for the proofreading. This work is supported by NSFC 62002224 and ShanghaiTech Startup Fund.

References

[1] 50,000 home cameras reportedly hacked, footage posted online. https://www.welivesecurity.com/2020/10/14/50000-home-cameras-reportedly-hacked-footage-posted-online/, 2022.

[2] Azure Cognitive Services. https://azure.microsoft.com/en-us/services/cognitive-services/#api, 2022.

[3] Buyer Beware: Used Nest Cams Can Let People Spy on You. https://www.nytimes.com/wirecutter/blog/used-nest-cams-can-let-people-spy-on-you/, 2022.

[4] Chromatic aberration. https://en.wikipedia.org/wiki/Chromatic_aberration, 2022.

[5] Furbo Dog Camera. https://shopcn.furbo.com/products/furbo, 2022.

[6] GitHub - zxing/zxing: ZXing ("Zebra Crossing") barcode. https://github.com/zxing/zxing, 2022.

[7] Minimum Object Distance. https://www.avsupply.com/MFG/rainbow-cctv-security/rainbow-cctv-security.htm, 2022.

[8] Nest Camera Recorder. https://github.com/dend/foggycam, 2022.

[9] NVIDIA Jetson Linux Driver Package Developer Guide. https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/, 2022.

[10] NVIDIA Jetson Nano Developer Kit | NVIDIA Developer. https://developer.nvidia.com/EMBEDDED/jetson-nano-developer-kit, 2022.

[11] People say they care about privacy but they continue to buy devices that can spy on them. https://www.vox.com/recode/2019/5/13/18547235/trust-smart-devices-privacy-security, 2022.

[12] QR Code Data Capacity. https://web.archive.org/web/20160326120122/http://blog.qr4.nl/page/QR-Code-Data-Capacity.aspx, 2022.

[13] Reedsolo PyPI. https://pypi.org/project/reedsolo/, 2022.

[14] Ring data can be view by staff. https://theintercept.com/2019/01/10/amazon-ring-security-camera/, 2022.

[15] Samsung's Family Hub Brings Food AI and Automation into the Kitchen at CES 2020. https://news.samsung.com/global/samsungs-family-hub-brings-food-ai-and-automation-into-the-kitchen-at-ces-2020, 2022.

[16] Security and Privacy of Yi Camera. https://www2.yitechnology.com/support/answer/id/17/tid/2/qid/1/oid/_, 2022.

[17] Smart TVs With Built-In Camera. https://window10forpc.com/smart-tvs-with-built-in-camera-and-how-much-they-worth, 2022.

[18] SSD MobileNet V2 by Google-Coral. https://raw.githubusercontent.com/google-coral/test_data/master/ssd_mobilenet_v2_face_quant_postprocess.tflite, 2022.

[19] Tasker. https://tasker.joaoapps.com/, 2022.

[20] TinyCAM. https://tinycammonitor.com/, 2022.

[21] Video bit Rate. https://helpdesk.spycameracctv.com/support/solutions/articles/43000480527-bit-rate-comparison-chart-for-hikvision-cameras, 2022.

[22] Vignetting. https://en.wikipedia.org/wiki/Vignetting, 2022.

[23] Warning issued over hackable security cameras. https://www.welivesecurity.com/2020/06/15/warning-issued-hackable-security-cameras/, 2022.

[24] Xiaomi IoT Cameras Leak Private Stills via Google Home Hub. https://securityboulevard.com/2020/01/xiaomi-iot-cameras-leak-private-stills-via-google-home-hub/, 2022.

[25] M. Backes, M. Dürmuth, and D. Unruh. Compromising reflections-or-how to read lcd monitors around the corner. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 158–169. IEEE, 2008.

[26] F. Brasser, D. Kim, C. Liebchen, V. Ganapathy, L. Iftode, and A.-R. Sadeghi. Regulating arm trustzone devices in restricted spaces. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 413–425, 2016.

[27] J. Bugeja, D. Jönsson, and A. Jacobsson. An investigation of vulnerabilities in smart connected cameras. In *2018 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*, pages 537–542. IEEE, 2018.

[28] A. Chattopadhyay and T. E. Boult. Privacycam: a privacy preserving camera using uclinux on the blackfin dsp. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[29] Y. Cheng, X. Ding, and R. H. Deng. Driverguard: a fine-grained protection on i/o flows. In *ESORICS'11 Proceedings of the 16th European conference on Research in computer security*, volume 6879, pages 227–244, 2011.

[30] V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

[31] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[32] S. Eskandarian, J. Cogan, S. Birnbaum, P. C. W. Brandon, D. Franke, F. Fraser, G. Garcia, E. Gong, H. T. Nguyen, T. K. Sethi, V. Subbiah, M. Backes, G. Pellegrino, and D. Boneh. Fidelius: Protecting user secrets from compromised browsers. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 264–280, 2019.

[33] M. Farajallah, W. Hamidouche, O. Déforges, and S. El Assad. Roi encryption for the hevc coded video contents. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 3096–3100. IEEE, 2015.

[34] X. Feng, M. Ye, V. Swaminathan, and S. Wei. Towards the security of motion detection-based video surveillance on iot devices. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, pages 228–235, 2017.

[35] J. Fernández-Berni, R. Carmona-Galán, R. Del Río, R. Kleihorst, W. Philips, and Á. Rodríguez-Vázquez. Focal-plane sensing-processing: A power-efficient approach for the implementation of privacy-aware networked visual sensors. *Sensors*, 14(8):15203–15226, 2014.

[36] A. Geiger, F. Moosmann, Ö. Car, and B. Schuster. Automatic camera and range sensor calibration using a single shot. In *2012 IEEE International Conference on Robotics and Automation*, pages 3936–3943. IEEE, 2012.

[37] D. Han, J. Choi, J.-I. Cho, and D. Kwak. Design and vlsi implementation of high-performance face-detection engine for mobile applications. In *2011 IEEE International Conference on Consumer Electronics (ICCE)*, pages 705–706, 2011.

[38] T. Hao, R. Zhou, and G. Xing. Cobra: color barcode streaming for smartphone systems. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 85–98, 2012.

[39] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer. Polisis: Automated analysis and presentation of privacy policies using deep learning. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 531–548, 2018.

[40] W. Hu, H. Gu, and Q. Pu. Lightsync: Unsynchronized visual communication over screen-camera links. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 15–26, 2013.

[41] S. Jana, D. Molnar, A. Moshchuk, A. Dunn, B. Livshits, H. J. Wang, and E. Ofek. Enabling {Fine-Grained} permissions for augmented reality applications with recognizers. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 415–430, 2013.

[42] X. Jin, F. Li, X. Li, W. Tian, B. Wang, L. Chen, and X. Wang. Visual-based data exchange system for internal and external networks in physical isolation. *Cognitive Robotics*, 1:134–144, 2021.

[43] T. Kitajima, E. A. Y. Murakami, S. Yoshimoto, Y. Kuroda, and O. Oshiro. Privacy-aware face detection using biological signals in camera images. *Electronics and Communications in Japan*, 101(6):67–79, 2018.

[44] M. Lentz, R. Sen, P. Druschel, and B. Bhattacharjee. Secloak: Arm trustzone-based mobile peripheral control. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 1–13, 2018.

[45] J. Li, Z. Li, G. Tyson, and G. Xie. Your privilege gives your privacy away: An analysis of a home security camera service. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 387–396. IEEE, 2020.

[46] W. Li, M. Ma, J. Han, Y. Xia, B. Zang, C.-K. Chu, and T. Li. Building trusted path on untrusted device drivers for mobile devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems*, page 8, 2014.

[47] S. D. Perli, N. Ahmed, and D. Katabi. Pixnet: interference-free wireless links using lcd-camera pairs. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 137–148, 2010.

[48] F. Pittaluga and S. J. Koppal. Pre-capture privacy for small vision sensors. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2215–2226, 2016.

[49] B. Rinner and T. Winkler. Privacy-protecting smart cameras. In *Proceedings of the International Conference on Distributed Smart Cameras*, pages 1–5, 2014.

[50] A. Senior, S. Pankanti, A. Hampapur, L. Brown, Y.-L. Tian, A. Ekin, J. Connell, C. F. Shu, and M. Lu. Enabling video privacy through computer vision. *IEEE Security & Privacy*, 3(3):50–57, 2005.

[51] D. N. Serpanos and A. Papalambrou. Security and privacy in distributed smart cameras. *Proceedings of the IEEE*, 96(10):1678–1687, 2008.

[52] L. Singaravelu, C. Pu, H. Härtig, and C. Helmuth. Reducing tcb complexity for security-sensitive applications: three case studies. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, volume 40, pages 161–174, 2006.

[53] P. Stifter, K. Eberhardt, A. Erni, and K. Hofmann. Image sensor for security applications with on-chip data authentication. In *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2006*, volume 6241, page 624109. International Society for Optics and Photonics, 2006.

[54] J. Tan, S. S. Khan, V. Boominathan, J. Byrne, R. Baraniuk, K. Mitra, and A. Veeraraghavan. Canopic: Predigital privacy-enhancing encodings for computer vision. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2020.

[55] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen. Kvazaar: Open-source hevc/h.265 encoder. In *Proceedings of the 24th ACM International Conference on Multimedia*, 2016.

[56] P. Viola, M. Jones, et al. Robust real-time object detection. *International journal of computer vision*, 4(34-47):4, 2001.

[57] A. Wang, Z. Li, C. Peng, G. Shen, G. Fang, and B. Zeng. Inframe++: Achieve simultaneous screen-human viewing and hidden screen-camera communication. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 181–195, 2015.

[58] A. Wang, S. Ma, C. Hu, J. Huai, C. Peng, and G. Shen. Enhancing reliability to boost the throughput over screen-camera links. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 41–52, 2014.

[59] Q. Wang, M. Zhou, K. Ren, T. Lei, J. Li, and Z. Wang. Rain bar: Robust application-driven visual communication using color barcodes. In *2015 IEEE 35th International Conference on Distributed Computing Systems*, pages 537–546. IEEE, 2015.

[60] S. Weiser and M. Werner. Sgxio: Generic trusted i/o path for intel sgx. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 261–268, 2017.

[61] T. Winkler, A. Erdélyi, and B. Rinner. Trusteye. m4: protecting the sensor—not the camera. In *2014 11th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 159–164. IEEE, 2014.

[62] T. Winkler and B. Rinner. A systematic approach towards user-centric privacy and security for smart camera networks. In *Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras*, pages 133–141, 2010.

[63] T. Winkler and B. Rinner. Trustcam: Security and privacy-protection for an embedded smart camera based on trusted computing. In *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 593–600, 2010.

[64] T. Winkler and B. Rinner. Security and privacy protection in visual sensor networks: A survey. *ACM Comput. Surv.*, 47(1), May 2014.

[65] H. Yu, J. Lim, K. Kim, and S.-B. Lee. Pinto: enabling video privacy for commodity iot cameras. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1089–1101, 2018.

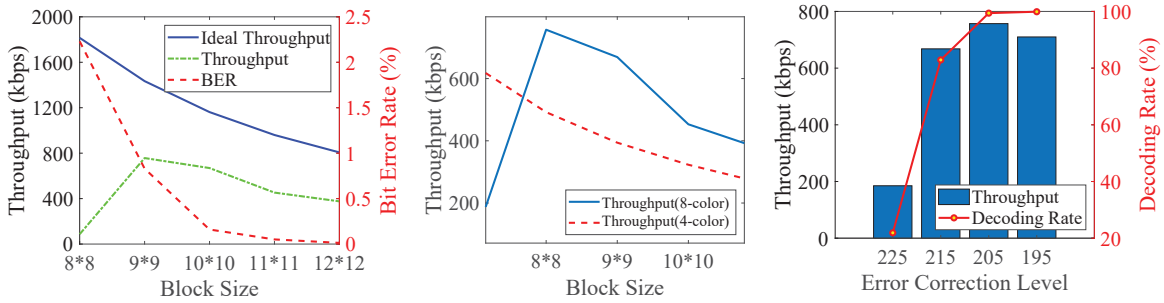
[66] O. Zhang, Z. Qian, Y. Mao, K. Srinivasan, and N. B. Shroff. Erscc: Enable efficient and reliable screen-camera communication. In *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 281–290, 2019.

[67] Y. Zhang, Y. Lu, H. Nagahara, and R.-i. Taniguchi. Anonymous camera for privacy protection. In *2014 22nd International Conference on Pattern Recognition*, pages 4170–4175. IEEE, 2014.

[68] Z. Zhang, Y. Matsushita, and Y. Ma. Camera calibration with lens distortion from low-rank textures. In *CVPR 2011*, pages 2321–2328, 2011.

[69] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang. East: an efficient and accurate scene text detector. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 5551–5560, 2017.

[70] Z. Zivkovic and F. Van Der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7):773–780, 2006.



(a) Color Block Size (b) Number of Colors Used (c) Error Correction Level

Figure 10: **VLC Parameter Selection**.

	txFR$_i$	txFR$_{i+1}$	txFR$_{i+2}$	txFR$_{i+3}$
txFR$_{x,1}$	A	C	C	C
txFR$_{x,2}$	B	B	D	D
txFR$_{x,3}$	A xor B	A xor B	A xor B	C xor D

Figure 11: **Inter-frame Coding Scheme**.

Appendix

A. VLC Inter-Frame Coding

Considering the possible sampling cases of the rxFRs in Figure 4 (e). As the two decodable rxFR pieces are random, to ensure they provide 2/3 capacity, the data of each piece must be coded. We denote the three pieces as subframes: txFR$_{i,1}$, txFR$_{i,2}$, txFR$_{i,3}$. A valid coding scheme is shown in Figure 11 to convey information pieces: A,B,C,D, *etc.*

We next show that, no matter when the rxFR samples, two distinct information pieces can be recovered. When the rxRF captures the static txRF frames, we surely can do so. So we consider the three cases affected by the roller shuttering effect in Figure 4 (e). In case ❶, B and A xor B can be decoded, so A can be decoded from A = B xor (A xor B). In case ❸, C and D can be directly decoded. In case ❷, C and A xor B can be decoded. In this case, the decoder needs to wait for the next rxFR, from which it obtains E and C xor D. With C xor D, D can be calculated.

B. VLC Parameter Selection

The performance of the VLC is affected by several parameters. This section describes how we choose them.

We adopt four metrics to quantify the performance. To define them clearly, suppose there are a data bits to transmit, and b bits are added for error correction. The VLC data path transmits $a+b$ bits out in total, of which c bits are flipped during the transmission. After error correction, the receiver recovers d data bits from the received $a+b$ bits. Then we have the following metrics: *ideal throughput* denotes the transmitted bits per second including the redundancy for error correction, *i.e.*, $(a+b)/t$; *throughput* denotes the actual correctly decoded data bits per second after error correction, *i.e.*, d/t; *decoding rate* denotes the ratio of correctly decoded data bits to all transmitted data bits after the error correction, *i.e.*, d/a; *bit error rate (BER)* denotes the ratio of all error bits over the total transmitted bits, *i.e.*, $c/(a+b)$. The four metrics are measured over 100 VLC frames.

Block Size: As shown in Figure 10 (a), the ideal throughput increases when block size decreases since more color blocks can be used to convey bits. However, decreasing the block size does not always turn into net capacity gain since the BER increases with smaller block size. This is because blocks with smaller sizes contribute less color intensity and are more fragile to the noise. The highest throughput 757 kbps is achieved at a block size of 9×9 pixels.

Number of Colors Used: We show two modulation schemes using different numbers of colors: 8-color and 4-color modulation. The colors the of 4-color modulation are black, white, green, and magenta. The 8-color modulation is shown in Figure 4 (d). Figure 10 (b) shows the performance of the two modulation schemes. When the block size is decreased, the throughput of the 4-color increases due to its smaller BER. However, a smaller block size is not possible due to the limitation of the block localization algorithm. As a result, 8-color modulation performs better with 9×9 block size. Thus, we use it as the default modulation scheme.

Error Correction Level: The RS code is described by RS(n,m), where n is output symbol rate and m is input data rate. n is larger than m, reflecting the redundancy used to recover errors. We fix n to 255 and vary m to change the error correction level. Smaller m means more redundancy and better error resistance. We test 4 levels as shown in Figure 10 (c). Smaller m brings better throughput. However, this gain has an upperbound since it needs redundancy. The optimal throughput of 757 kbps is achieved at the error correction level of RS(255, 205) with a 99.4% decoding rate. We use it as the default error correction level.

VLC rxFR Extraction	VLC Decoding		ROI Tiles Decryption	Pipelined Time Cost
	Data Decoding	RS Decoding		
18.9 ms	12.4 ms	3.1 ms	9.7 ms	19.4 ms
44.1 ms				

Table 6: **Per-frame Time Cost of the CamShield App**.

C. The CamShield App Run-time Efficiency

In order to achieve real-time video rendering, the total time budget for a VLC frame in CamShield App is 50 ms given the 20 fps capture rate of the sink camera. Our smartphone supports real-time 60 fps video decoding with an average time cost of 16.7 ms. Thus, the VLC decoding and ROI decryption must be finished within the remaining time budget, *i.e.*, $50 - 16.7 = 33.3$ ms.

We first measure the per-frame time cost of different components of the CamShield App, as shown in Table 6. If they work sequentially, the summed cost (44.1 ms) will exceed the time budgets. To speed up the decoding, we divide the process into three parts and use three threads for parallel decoding. As shown in the last column of Table 6, the average per-frame time cost of the pipeline design is 19.4 ms, within the 33.3 ms budget. The results show the feasibility of real-time video rendering on smartphones.