Improving the Security of Visual Challenges

JUNIA VALENTE, KANCHAN BAHIRAT, and KELLY VENECHANOS, The University of Texas at Dallas
ALVARO A. CARDENAS, University of California, Santa Cruz
PRABHAKARAN BALAKRISHNAN, The University of Texas at Dallas

This article proposes new tools to detect the tampering of video feeds from surveillance cameras. Our proposal illustrates the unique *cyber-physical* properties that sensor devices can leverage for their cyber-security. While traditional attestation algorithms exchange *digital* challenges between devices authenticating each other, our work instead proposes challenges that manifest physically in the field of view of the camera (e.g., a QR code in a display). This physical (challenge) and cyber (verification) attestation mechanism can help protect systems even when the sensors (cameras) and actuators (a display, infrared LEDs, color light bulbs) are compromised. In this article, we consider skillful adversaries that can capture the correct challenges (our system is sending) and can re-create them in the response to try fooling our verification system, and we propose new algorithms to detect these powerful attackers. Also, we introduce new visual challenges that make harder for anti-forensics attackers to succeed, and we present experimental results showing how our system is robust against a variety of attacks ranging from naive attacks to more sophisticated anti-forensics attackers.

CCS Concepts: • **Security and privacy → Intrusion detection systems**; **Spoofing attacks**; • **Computing methodologies → Image manipulation**; • **Hardware → Physical verification**;

Additional Key Words and Phrases: Security of the Internet of Things (IoT), multimedia security, visual challenges, video forensics

ACM Reference format:
Junia Valente, Kanchan Bahirat, Kelly Venechanos, Alvaro A. Cardenas, and Prabhakaran Balakrishnan. 2019. Improving the Security of Visual Challenges. *ACM Trans. Cyber-Phys. Syst.* 3, 3, Article 34 (August 2019), 26 pages.
https://doi.org/10.1145/3331183

1 INTRODUCTION

The integrity of video cameras is critical in a variety of sensitive settings, including the surveillance of nuclear facilities to confirm countries are abiding by the Nuclear Non-Proliferation Treaty (Tabatabai 2015), monitoring certificate vaults protecting secret keys (Goodin 2012),

This work was supported by the U.S. Army Research Office (ARO) Grant No. W911NF-17-1-0299, NSF Grants No. CNS 1931573 and No. CNS 1929410, and by the Laboratory of Analytic Sciences. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ARO, LAS, and/or any agency or entity of the United States Government.
Authors' addresses: J. Valente, K. Bahirat, K. Venechanos, and P. Balakrishnan, Erik Jonsson School of Engineering and Computer Science, The University of Texas at Dallas, 800 W. Campbell Road, Richardson, Texas 75080; emails: {juniavalente, kanchan.bahirat, kelly.venechanos, bprabhakaran}@utdallas.edu; A. A. Cardenas, Jack Baskin School of Engineering, UC Santa Cruz, 1156 High Street, Santa Cruz, California 95064; email: alacarde@ucsc.edu.

2378-962X/2019/08-ART34 $15.00
https://doi.org/10.1145/3331183

monitoring computers generating random numbers for the lottery (Goodin 2015), and surveillance of electricity substations (Sheriff 2013). While cameras in these sensitive scenarios have cryptographic authentication and integrity, if an adversary obtains the secret key of the camera or compromises the device itself, then the adversary can send fake video footage. As the sensitivity of usage scenarios and pervasiveness of security cameras increase, we need to protect them by defense-in-depth mechanisms to ensure captured images are fresh and authentic.

In our earlier work, we introduced the idea of *visual challenges* (Valente and Cárdenas 2015) as a defense-in-depth solution for surveillance cameras: We send a visual challenge to the physical area under surveillance and verify that the desired changes reflect in the video feed. In short, a trusted verifier creates a new visual challenge. Then it sends the challenge to the environment that the camera is monitoring (i.e., to a display in the field-of-view of the camera) such that the camera captures the challenge in the video and relays it back to the verifier. The verifier—who knows the challenge—can then check if the captured video has the challenge and thus increase its confidence that the video is fresh and authentic. This approach works notably well in legacy systems and other systems where we cannot change the device (camera) software or hardware, nor receiving server (e.g., network video recorder, external video feed storage).

Visual challenges illustrate unique properties of cyber-physical systems: Physical challenges improve the security of a digital/cyber system. At a high-level, visual challenges are motivated by attestation (Parno 2008) and challenge-response protocols (Abadi and Needham 1994) where a *verifier* sends a random challenge to a *prover*, and the prover replies with a message showing its freshness and authenticity. Our approach differentiates in two ways: (1) our challenge is not digital but *physical*, and (2) it is not sent to the prover itself but to the physical environment the prover is monitoring.

One limitation in our previous work (Valente and Cárdenas 2015) was that while visual challenges help identify fake footage (when the prover is a camera), a skillful adversary—who knows the visual challenge mechanism is in place—can launch *copy-paste* forgery attacks to fool the detection mechanism. For example, the adversary can capture the correct visual challenge and then manipulates the video feed to insert the correct challenge onto fake image frames. In this article, we overcome this limitation and improve our previous proposal in several ways: (1) we consider a stronger adversary model with multimedia anti-forensics attacks, (2) we propose new defenses to counter this powerful adversary—like the continuous change of visual challenges (e.g., via a light bulb)—and we make harder for this powerful attacker to succeed, (3) we incorporate new types of *media* to transmit new visual challenges, (4) we present a prototype for hiding the existence of our system from the adversary (i.e., hiding visual challenges), and (5) we improve the detection rate of *replay attacks*. To incorporate these changes, we propose a modular and extensible architecture in this article.

Proposed Architecture. We present an overview of our proposed architecture in Figure 1.

Our architecture has three components: the physical environment we want to monitor, a surveillance camera, and a trusted verifier; the *physical environment* contains a device that constantly shows new visual challenges. In our previous works, we proposed using a digital monitor (e.g., a small tablet, digital signage) to display visual challenges such as `QR codes` and random letters (Valente and Cárdenas 2015) or `tweets` from verified users (Valente and Cardenas 2017). In this article, we expand the visual challenge options to include `full-image challenges` (using a multi-color smart light bulb) that can change the color of the scene under surveillance, and infrared (IR) light that is hidden from people but visible to cameras (e.g., with night vision). The *camera* visually perceives the environment and has typical features from surveillance cameras, such as the ability to change its resolution, frame rate, and the ability to select day or night vision.

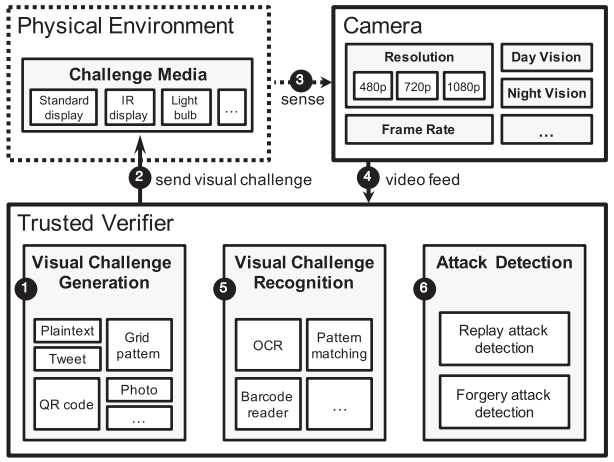


Fig. 1. Architecture overview: we extend our previous proposal (Valente and Cárdenas 2015) to support new visual challenges such as a grid pattern of IR LEDs and use multi-color smart light bulbs to change ambient lighting, to include more robust attack-detection algorithms (e.g., forgery attack detection) and more improvements. The verification happens as follows: (1) the verifier creates a new visual challenge and (2) sends to a display; (3) the camera captures video of a scenery (including the display); (4) the verifier retrieves the next video frame and (5) verifies the challenge in the frame just received: if the verifier confirms the challenge, then it gains confidence that the camera is transmitting fresh and authentic footage. Finally, (6) the verifier runs attack-detection algorithms.

(a) Verifier code (running on laptop) sending visual challenges to a display in the field-of-view of a camera.

(b) `QR code` visual challenge being displayed on a small tablet device (i.e., independent from camera and verification mechanism).

(c) Passphrase visual challenge being displayed on the field-of-view of a Swann 720p wireless IP camera with day/night vision and built-in cut IR filters.

Fig. 2. Setup for experiments using displays to show the visual challenge.

The *trusted verifier* has a visual challenge generation component (e.g., for creating a `QR code` based on some random value) and then a visual challenge recognition algorithm that receives the camera feed and detects if the visual challenge is present or not, and it uses image processing to retrieve the encoded data (e.g., uses OCR algorithms to recognize `tweet` challenges). Finally, the verifier also has attack-detection algorithms such as the ones we propose in this article. We show the setup for our experiments in Figure 2.

Contributions. Our contributions over previous works include: (1) a refined threat model against video tampering, (2) improved visual challenge algorithms—mainly in terms of performance (to achieve higher visual challenge recognition) and security (to detect attacks including replay attacks), (3) new attack detection algorithms that leverage correlations between consecutive video frames (i.e., *inter-frame* correlations) to detect forgery attacks, (4) new visual challenges

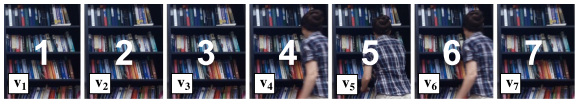


Fig. 3. Sequence of image frames to illustrate attack strategies. The goal of the attacker is to hide the activity on frames 4, 5, and 6 where a student removes a book from the bookshelf. v_1 to v_7 represent the visual challenge.

like *changing the environment color of the area under surveillance* to impede copy-paste attacks, and (5) new proposals and prototypes to hide the existence of visual challenges from adversaries. We present experimental results and show the performance of our detection system against a variety attacks (that we describe in Section 2).

To the best of our knowledge, previous works—that use physical challenges—do not consider the skillful adversaries we evaluate in this article, i.e., an attacker who knows the correct challenges and how to re-create them in the response. For example, Mo and Sinopoli (Mo et al. 2014) assume physical challenges (for control systems) are secret and hidden from the adversary (e.g., the adversary cannot reproduce watermarks introduced in the system), and Shoukry et al. (Shoukry et al. 2015) consider a weaker adversary that cannot *intercept* sensor signals and replace them with their own fabricated responses. In this article, we explicitly assume the adversary knows the watermarks and can fabricate false (but expected) responses to try fooling the verifier. The goal of our work is to minimize the chances these powerful attacks succeed in practice.

Outline. The article is organized as follows: in Section 2, we present our threat model and outline attacks against video feeds from surveillance cameras. We classify the proposed attacks as (1) naive or (2) forgery attacks. In Section 3, we address the naive attacks, and then in Sections 4 and 5, we present experimental results to detect forgery attacks. In particular, we propose using inter-frame correlations to detect copy-paste forgery attacks against our system (when the attacker knows we are using `QR codes`) in Section 4. In Section 5, we introduce `full-image challenges` to make forgery attacks harder to succeed against our system and to help detect stronger forgery attacks. In Section 6, we summarize related work. Finally, we conclude our work and discuss future research in Section 7.

2 OVERVIEW OF THREAT MODEL

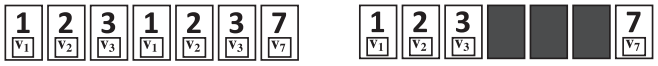
We assume an attacker that has compromised a surveillance camera (or the authentication material like secret keys used by the camera) and attempts to send fake video footage to receiving servers. In this article, we go beyond attackers that replay old footage of the camera without taking into account that there is a visual challenge present. We assume the attacker can see the visual challenge (or the communication between the verifier and the display) and then tries to launch attacks that use the correct visual challenge to fool the verifier.

We use Figure 3 to help us describe these attacks. The goal of the attacker is to hide the activity in frames 4–6. Notice that each video frame is tagged with a visual challenge v_1 to v_7. We can assume each tag represents a unique visual challenge that was sent by the verifier to a display in the field-of-view of the camera. Visual challenges can be `QR codes` or plain text as we proposed in (Valente and Cárdenas 2015) or social media feeds from public spaces (Valente and Cardenas 2017).

We consider the following five attacks:

(a) Attack 0: Replay attack -Figure 4(a)
(b) Attack 1: Timing attack -Figure 4(b)

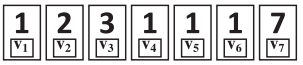
} **Naive Attacks—Section 3**



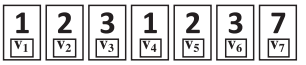
(a) Attack 0: replay attack (b) Attack 1: timing attack

Fig. 4. Naive attacks proposed and detected by our previous work (Valente and Cárdenas 2015).

(a) Attack 2: forgery using a *random* frame to paste the visual challenges.



(b) Attack 3: forgery using an *old* frame to paste the visual challenges.



(c) Attack 4: forgery using *sequence* of old frames (instead of repeating a single same frame).

Fig. 5. Copy-and-paste forgery attacks: The attacker copies the current visual challenge and pastes it to (a) a significantly different frame, (b) an old frame, or (c) a sequence of previous frames.

(c) Attack 2: Forged visual challenges using fake video -Figure 5(a)
(d) Attack 3: Forged visual challenges on old still frame -Figure 5(b)
(e) Attack 4: Forged visual challenges on sequence of old frames- Figure 5(c)

} **Forgery Attacks—Sections 4 and 5**

In the first attack, the attacker does not know about the visual challenge mechanism and therefore the replay attack contains the wrong visual challenges (e.g., v_1 instead of v_4); in the second attack, the attacker only knows the time the verifier takes to raise an alert (and blocks the feed for the maximum amount of time it can do so without raising the alert); and in the final three attacks, we consider a stronger attacker that knows the visual challenge and tries to insert it in a fake video footage to bypass the verifier (i.e., in attacks 2–4, the attacker inserts the correct visual challenges to the video feed). We now describe the attacks in more detail.

2.1 Naive Attacks

In our previous work (Valente and Cárdenas 2015), we study the effectiveness of QR code and random letter challenges to detect two naive attacks: (1) *replay attack* and (2) *timing attack*. We provide attack-detection statistics, and study their performance. In this article, we improve detecting these attacks, and use the discussion to provide background information.

Attack 0: Replay Attack. This attack represents the typical Hollywood-movie attack where a hacker taps into the video footage of a camera and replays old video footage, while the attackers complete a heist that is not shown in the monitors of the security guards. Figure 4(a) illustrates this attack.

We consider this attack solved by our previous work (Valente and Cárdenas 2015; Valente and Cardenas 2017), and therefore in this article, we focus on how to improve attack detection for attacks 1-4.

Attack 1: Timing Attack. In our previous work (Valente and Cárdenas 2015), we found that waiting for a small number of frames and collecting enough evidence before making a decision (to raise an alarm or not) gives more accurate results (reducing the false alarm rate effectively to zero). However, if the attacker knows there is a small buffer of frames where we do not raise alerts, then the attacker can abuse it and try to launch the attack during this buffer of frames without raising an alert. Figure 4(b) illustrates this attack where frames 4–6 indicate when the attacker blocks visual challenges (and use fake frames) for the (buffer) duration time we do not raise an alert to stay undetected.

In our previous work, the attacker could launch these undetected attacks for up to 9s. In this article, we improve the performance of the visual challenge recognition algorithms to tolerate fewer decoding errors and minimize the amount of time the attacker can remain undetected.

2.2 Forgery Attacks

In this article, we study *forgery attacks* against our visual challenge system. These attacks attempt to forge a fake image displaying the expected visual challenge. We consider three strategies to create forgeries: the adversary replaces current frame with a (1) random image or (2) old image frame (and *replays* them while superposing the expected challenge), or the adversary replaces current and subsequent frames with (3) image frames from an old sequence.

Attack 2: Forged Visual Challenges Using Fake Video. As suggested by Zhang et al. (2017), a skillful attacker could bypass visual challenges: one way is to physically move or rotate the camera (to keep it away from the target area) while at the same time adding the visual challenge in the correct location (e.g., moving the display with the `QR code`, or placing a new and different display where the verifier expect the code to appear).

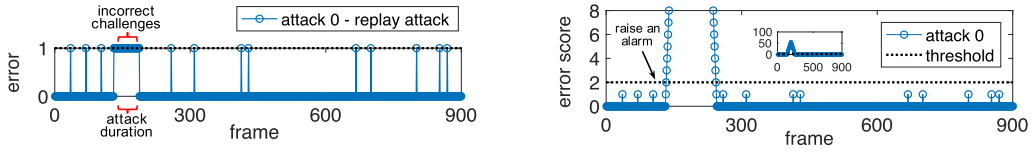
Another way involves the attacker sending a fake video while still including the expected visual challenge in the fake feed. The attacker can copy the `QR code` from the legitimate feed (captured from the original camera) and paste it over a fake video before sending the video feed to the verifier. This attack is well-known in the image/video forensics literature as a copy-paste forgery attack. In this article, we study copy-paste attacks in the context of visual challenges, which has not been considered before in previous related works.

These two strategies produce the same effect: a tampered video feed with the correct visual challenge. We illustrate this attack in Figure 5(a). In this article, we mitigate this attack by computing the correlation between inter-frames of the video feed and finding discrepancies in consecutive frames. We present a detection algorithm (using visual challenges and inter-frame correlations) and show our experimental results.

Attack 3: Forged Visual Challenges on Old Still Frame. The attacker launches a combination of replay and forgery attacks: the attacker maintains a copy of old frames, and during the attack, replays a perceptually similar (to the current) frame while injecting a forged visual challenge. Figure 5(b) illustrates this attack. If the attacker replaces the real footage with a still frame that is similar to the legitimate frames, then the previous mitigation—focusing on finding significant perceptible discrepancies in consecutive frames— will fail. Instead, we can successfully detect this attack if we focus on *constant* changes (or no changes) between consecutive frames.

Our experiments show that even when the area under surveillance remains unchanged, the inter-frame correlations between consecutive frames vary due to camera noise. Consequently, when the correlations remain constant, we can effectively conclude frames are being duplicated (i.e., attacker is replaying a still frame). We present experimental results to support this observation.

Attack 4: Forged Visual Challenges on Sequence of Old Frames. This is the strongest attack against our system: The attacker captures the correct visual challenges and inserts them into a *sequence* of fake frames that are significantly correlated to previous legitimate frames. Figure 5(c) illustrate this attack. The previous two mitigations fail, because we no longer can rely on significant discrepancies between consecutive frames nor on constant changes. To defend against this attack, we combine inter-frame correlations with a more robust visual challenge. Instead of changing only a small visual cue in the field-of-view of the camera (which is the case for `QR code` challenges), we propose `full-image challenges` that reflect in the entire scenery captured by the camera, and show our experimental results using these challenges.



(a) Attack 0: replaying old frames

(b) Attack 0: mismatch score increases during attack

Fig. 6. Detecting Attack 0: (a) when an attacker blindly replays old frames, the old frames will not contain the expected visual challenge—when there is a mismatch or error we denote as "1" otherwise as "0"—as a result the (b) *cumulative error score* will quickly increase beyond a threshold, raising an alarm.

Before we discuss our new attack-detection mechanisms in detail, we first present an overview of our proposed architecture and its architectural components.

3 DETECTION OF NAIVE ATTACKS

3.1 Detection of Attack 0: Replay Attack

Although we consider the problem of detecting attack 0 solved by our previous work (Valente and Cárdenas 2015; Valente and Cardenas 2017), we briefly revisit this attack to introduce background information about our proposal.

In our earlier work, we experimented using QR codes, random strings, or meaningful information (such as tweets) as visual challenges. Our previous results found limitations when using QR codes when compared to using random letters or tweets. In particular QR codes can be fully decoded or not, so even small errors in the display or noise in the camera can prevent us from decoding them; in contrast, we can obtain partial information from corrupted tweets or from random letter strings. Therefore, in our earlier work, we had more false alarms from incorrectly decoded QR codes than from strings of letters and as such, we had lower performance with them.

In this article, we use a new method to substantially increase the decodability rate of QR codes. We use the built-in error detection feature (Denso Wave Incorporated 2015). For instance, we use error detection Level H feature, which means that even when 30% of the QR code has been damaged, we can still restore the data encoded in the code. So even when the monitor displaying the code or the camera capturing the video add noise, the error detection level helps us successfully decode QR codes.

By applying this change, we saw few decodability errors in our experiments. In our experimental results shown in Figure 6, we had only 12 errors when trying to decode QR codes from 900 image frames with distinct challenges (when there is no attack), giving a 98.67% decodability rate. We denote errors as "1" and every time the decoded QR code matches the expected challenge as "0" (shown in Figure 6(a)).

Further, we show in Figure 6(a) that it is straight forward to detect replay attacks using visual challenges: when the replay attack starts, the verifier notices a consistent mismatch of visual challenges (because the old frames being replayed contains old visual challenges). The verifier either sees visual challenges that are not the expected ones (not the ones the verifier has been currently sending to the display) or occasionally (but never consistently) gets a decodability error. Every time there is a mismatch or error, the verifier increases a *cumulative error score* as shown in Figure 6(b). We raise an alarm only when the errors are persistent. In our case, we keep a zero false alarm rate when we select a threshold of $\tau = 2$, and this still allows us to have 100% attack-detection rates.

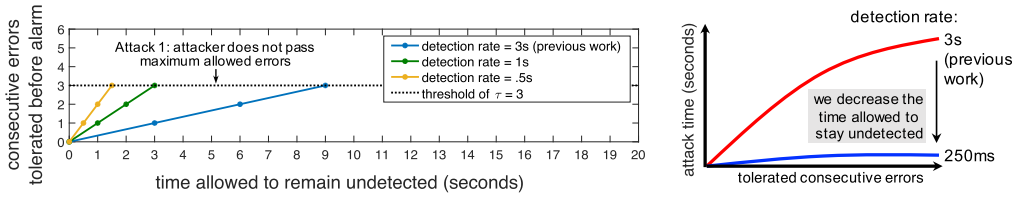


Fig. 7. Illustration of Attack 1: Attack duration depends on verification detection rate: higher rates decrease attack duration.

3.2 Detection of Attack 1: Timing Attack

The improvements we mentioned in the previous section can already help improve the mitigation of attack 1. Here, we provide additional insights to decrease the chances that this attack succeeds.

Recall that attack 1 takes advantage of the time the verifier takes to raise alerts. To avoid unnecessary false alarms, the verifier does not raise an alert every time there is a decoding error in QR codes, instead the verifier waits for subsequent verifications until it detects a consistent mismatch of challenges. We assume the attacker knows about this small buffer of tolerance and can abuse it by launching the attack during this particular buffer of frames, and stay undetected. Our goal here is to minimize the total amount of time the verification might accept frames containing undecodable QR codes.

In our previous work (Valente and Cárdenas 2015), our best performance tradeoff (100% detection accuracy and minimum time delay) allowed an attacker—launching attack 1—to remain undetected for 9s. In this article, we improve the *detection rate* and *camera capturing rate* to maintain 100% accuracy while significantly reducing the amount of time to raise an alarm.

The duration of the attack is correlated with the detection rate, which is the frequency a verifier runs the verification step (to check visual challenges in current frames). If the verifier only tries to decode QR codes at every 3s, then that is also the time it takes to increase the cumulative error score. So, when the verifier tolerates three consecutive errors to happen in the system, that sums up to a total of 9s (i.e., the system will tolerate three fake frames with each containing a QR code that is not decodable—on purpose by an attacker). After that, a next frame containing a bad QR code will trigger an alert. To remain undetected and to use the full 9s, the attacker can stop the attack before hitting the threshold (of tolerated consecutive errors) and wait until the cumulative error score reaches zero to start a new attack.

To mitigate this attack, we increase the verification detection rate. If the verifier performs the verification step at every second (illustrated in green in Figure 7(a)), then the verifier takes one second to increase the cumulative error score. As a result, the attacker takes 3s (versus 9s in the previous case) to arrive at the maximum allowed errors. If the verifier performs the verification at each half second (illustrated in yellow), then the tolerated errors translate to a total of 1.5s of attack time (which is substantially smaller than 9s).

In this article, we increase the detection rate to 250ms (a quarter of a second instead of 3s). Even if we still tolerate three consecutive errors, we already decrease the attack time—to use fake frames with no or bad QR codes and not get caught—to three fourths of a second. Since in this work we improve the decodability rate of QR codes (now we get few false alarms and even fewer consecutive errors), we can decrease the tolerated consecutive errors to *two*. As a result, we effectively decrease the attack time to half a second while still maintaining a zero false alarm rate. We illustrate our improvements in Figure 7(b).

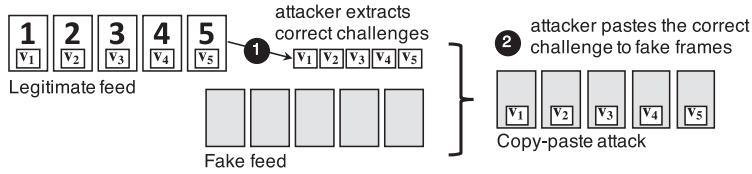


Fig. 8. Copy-paste forgery attack illustration.

(a) legitimate frame

(b) challenge v_4 (from legitimate frame) forged to a fake different frame

(c) challenge v_4 (from legitimate frame) forged to an old frame

Fig. 9. Sample forged frames from our experiments. These frames illustrate attacks 2–4.

We can also increase the camera capturing rate to minimize attack 1. Our previous work considered accessing the video feed at one frame per second (fps) and updating challenges every 3s—which explains the slower verification. In our experiments, we found it is more advantageous to use higher rates as it provides the verifier with additional frames during the verification step.

In the experiments we present in this article, our camera captures at 15fps. The verifier sends unique visual challenges continuously at every second and performs the verification step against each captured frame. Notice that this configuration allows multiple consecutive frames to contain the same visual challenge. This leads to an attack with similar effect as attack 3: an attacker can capture the first frame containing a new challenge and can replay that until a new challenge appears. We mitigate this attack by calculating the inter-frame correlation as we present next.

4 DETECTION OF FORGERY ATTACKS 2 AND 3

Visual challenges such as QR codes are fragile to copy-paste attacks. As we illustrate in Figure 8, it is trivial for an attacker—with access to the legitimate video feed—to spot where the challenge is because the location is fixed. A skillful attacker can then copy the correct challenge and paste it to a fake feed—before the verification step. In this section, we introduce inter-frame correlations including perceptual hash (Weinhaus 2014) and correlation coefficient (Wang and Farid 2007) metrics to detect this strongest attack against our detection system.

As we show later, it becomes difficult to achieve copy-paste attacks when the visual challenge spreads out in the physical environment and therefore appears in the entire image frame (e.g., by changing the scene lighting).

4.1 Attacks 2 and 3 Details

Figure 9 shows sample frames from our experiments. During the attack, the attacker copy-pastes the visual challenge from the legitimate frame (shown in Figure 9(a)) to a fake frame. Recall that attack 2 uses a fake feed that is different from the expected scene under surveillance (Figure 9(b)) and attack 3 uses an old frame (Figure 9(c)). During these attacks, the attacker keeps on repeating the same fake frame while pasting the legitimate challenge.

In practice, these attacks depend on the attacker's ability to create forgeries in real-time (as the camera is transmitting video feed to the verifier) and to possess fake or old frames. We assume the attacker meets these requirements.

Each forged frame is then passed to the verifier as if they were the current frame. The verifier checks the visual challenge (e.g., decodes the QR code) and finds the expected value. A naive verifier marks the frame as containing the correct challenge but in reality, the frame was forged to include the expected challenge. To deal with this, we complement the visual challenges verification with a new step to check discrepancies between consecutive frames.

4.2 Using Inter-frame Correlations to Detect Copy-Paste Attacks

The intuition for using either perceptual hash or correlation coefficient metrics includes: if the attacker uses a remarkably different video feed (in comparison to the legitimate one), then there is a *noticeable* change between frames when the attack starts and ends. We also see a sudden change in the correlation values. Further, we found that two frames are never exactly the same (because of noises introduced by the camera). So, the opposite is also true: when multiple frames correlate beyond an upper bound (close to classifying two frames as the same), we can detect duplicates frames, i.e., we can spot an attacker using a same still frame—to create forgeries—and we can detect such attack.

We classify the similarity between consecutive frames into either one of two classes depending on the metric. For the perceptual hash metric, we classify as follows:

$$\text{phash\_class} = \begin{cases} 1, & \Delta\text{h} > \hat{\tau}_{phash} \text{ or } \Delta\text{h} \leq \bar{\tau}_{phash} \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

and for the correlation coefficient metric as follows:

$$\text{corr\_class} = \begin{cases} 1, & \Delta\text{h} < \hat{\tau}_{corr} \text{ or } \Delta\text{h} \geq \bar{\tau}_{corr} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The first class represents when there is an attack: both when there is a sudden change (because the attacker starts using a completely different image frame—we use threshold $\hat{\tau}$) and when consecutive frames are duplicated or similar enough (because the attacker is forging legitimate challenges onto a still fake frame—we use threshold $\bar{\tau}$). We use Δh to denote the correlation between two consecutive frames. The second class represents when there is no attack.

Recall from our architecture that the verifier contains a visual challenge generation phase, a challenge recognition phase, and the attack-detection phase (to detect forgeries in addition to replay attacks). We refer the reader to Algorithm 1 for an overview of the steps. As we described before, the *generation phase* is straight forward: the verifier $\mathcal{V}$ generates random challenges (e.g., here, we use QR codes for illustration) and continuously sends them to a display in the field-of-view of the camera. Now, we incorporate inter-frame correlations during the *verification phase*. After verifier $\mathcal{V}$ certifies the current frame contains the correct visual challenge and that the response arrived within an expected response time (`line 6`), then $\mathcal{V}$ calculates the correlation between current frame F_j and previous frame F_{j-1} (`line 7`). If the correlation is outside the expected range (based on Equations (1) and (2)), then we raise an alarm (`lines 8-10`).

Although both perceptual hash and correlation coefficient metrics effectively detect anomalies in our system, they work slightly different: (1) correlation coefficient classifies more coarsely (images are either highly correlated or they are not) than perceptual hash, which classifies images as either very similar or with a score in between; (2) one calculates the similarity between two images while the other, the difference. We provide additional information in Appendix A.

ALGORITHM 1: Visual challenge: QR code

Verifier $\mathcal{V}$ uses `qr-code` visual challenges to verify video feed freshness of surveillance cameras.

Generation Phase

for $i \in \mathbb{N}_n$ **do**
  $r_i \leftarrow \{\text{alphanumeric characters}\}^{l_r}$ ▷ $\mathcal{V}$ generates a random string r_i of length l_r
  $v_i \leftarrow \texttt{encode}(r_i)$ ▷ $\mathcal{V}$ encodes r_i to `qr-code` visual challenge v_i
  store current time $t(v_i)$
  send v_i to digital display ▷ $\mathcal{V}$ sends v_i to a display in the field-of-view of the camera
  wait for random time $t_r(v_i)$
end for

Verification Phase

for $j \in \mathbb{N}_n$ **do**
  $F_j \leftarrow$ next image frame ▷ $\mathcal{V}$ receives an image frame F_j from surveillance camera
  $v_l \leftarrow$ retrieve `qr-code` from F_j ▷ $\mathcal{V}$ locates `qr-code` v_l from image frame
  $r_l \leftarrow \texttt{decode}(v_l)$ ▷ $\mathcal{V}$ attempts to decode `qr-code` v_l
  store current time t'
  if $\texttt{verify}(r_l)$ **and** $(t' - t(v_l) <= t_r(v_l) + \delta_t)$ **then** ▷ $\mathcal{V}$ verifies v_l and expected response time
    $\Delta\delta = \texttt{corr}(F_j, F_{j-1})$ ▷ $\mathcal{V}$ calculates the inter-frame correlation between F_j and F_{j-1}
    if $(\Delta\delta < \tau_{lo}$ **or** $\Delta\delta > \tau_{hi})$ **then**
      trigger an alert ▷ $\mathcal{V}$ triggers an alert if correlation is not within expected range
    end if
  else
    increase cumulative error score ▷ $\mathcal{V}$ keeps a cumulative error score
    if $\texttt{cumError} > \tau_{error}$ **then**
      trigger an alert ▷ $\mathcal{V}$ triggers an alert if cumulative error score passes a threshold
    end if
  end if
end for

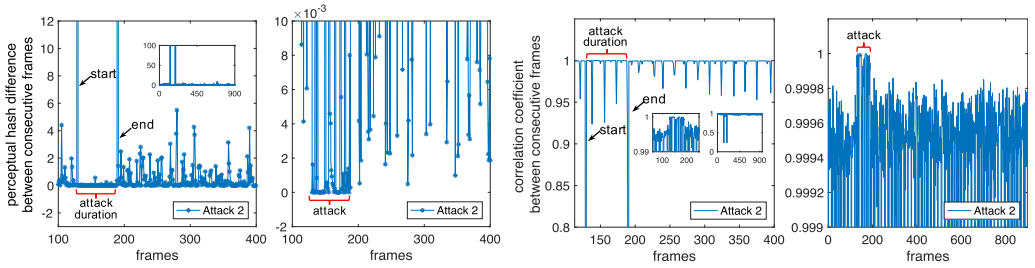
4.3 Experimental Results for Detecting Attacks 2 and 3

Experiment: During each try, we let the camera capture 900 frames. We then forge frames (e.g., F_a through F_{a+n}) to contain the expected visual challenge. We create our forgeries in `MATLAB`. First, we create two image arrays: one for the visual challenge (by deleting—in the legitimate frame—the regions outside the challenge); and another for the fake frame (we delete the region where the visual challenge will appear). Then, we perform the `plus` element-wise binary operation in `MATLAB` to combine the two image arrays and create the forged image.

Then, we run the verification phase: $\mathcal{V}$ calculates the correlation between consecutive frames. We run our algorithm using the perceptual hash difference and then the correlation coefficient. First for attack 2, and then for attack 3. We plot the outcome values, and show our results next.

Attack 2: Recall that attack 2 uses a completely different image frame (during the attack). So, when we use either perceptual hash or the correlation coefficient, we can notice a dramatic change in the values.

Figure 10(a) shows our results when using the perceptual hash metric. Figure 10(a) (left) shows a clear spike when the attack starts and when it ends. So, we can raise an alarm every time $\Delta\delta$ is larger than, for instance, 10. Notice that if we zoom-out the plot (the little box), the perceptual hash difference values exceed 100 when the attacker starts and stops using a fake frame. We can



(a) Attack 2 - perceptual hash difference (b) Attack 2 - correlation coefficient

Fig. 10. Detecting attack 2: We calculate the perceptual hash (left) and correlation coefficient (right) between consecutive frames.

correctly classify these sudden changes if we use threshold $\hat{\tau}_{phash}$=149.64. This threshold does not raise any false alarms. Now for the attack duration, we can notice the perceptual hash values stay close to 0. We can see this more clearly in Figure 10(a) (right). When we zoom-in via the y-axis (using ranges $[-.002, .01]$), we notice that the correlations have higher precision. Then when there is no attack, these values vary more and are farther from 0.

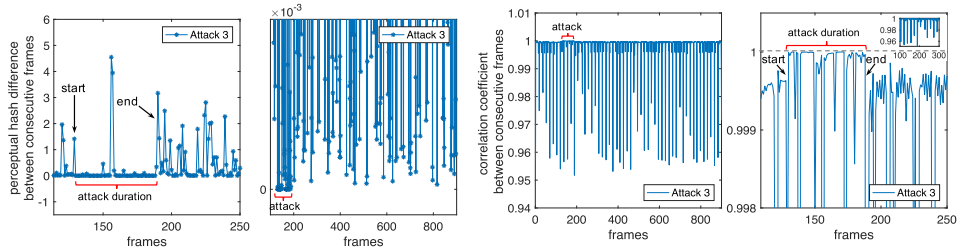
Figure 10(b) shows our results when using the correlation coefficient metrics. Our results are similar to when using the perceptual hash metric: We can detect when the attack starts and ends, and the duration of the attack. We again see a sudden change in the correlations. As we show in Figure 10(b) (left), while the correlation between two legitimate frames is never below 0.9, the correlation when the attack starts (and ends) is below 0.2. We classify these sudden changes correctly without raising any false alarm when we use threshold $\hat{\tau}_{corr}$=.2377. Also, when we zoom-in this plot, we clearly see a change in the correlations during the attack. We show this in Figure 10(b) (right). These values remain close to 1, which is different than what they were before and after the attack.

Notice that during the verification phase, $\mathcal{V}$ will immediately raise an alarm when attack 2 begins. Since $\mathcal{V}$ constantly checks the correlation between the current and previous frame, to see if they are below or above a threshold using $\left(\Delta\delta < \tau_{lo} \textbf{ or } \Delta\delta > \tau_{hi}\right)$, then $\mathcal{V}$ keeps on triggering an alarm until the attack ends.

Attack 3: Recall that attack 3 forges the visual challenge to an *old* frame. This means the correlation between a forged frame and a legitimate one may be similar to the correlation between two legitimate frames. So unlike attack 2, we may not see a sudden change when the attack starts and ends.

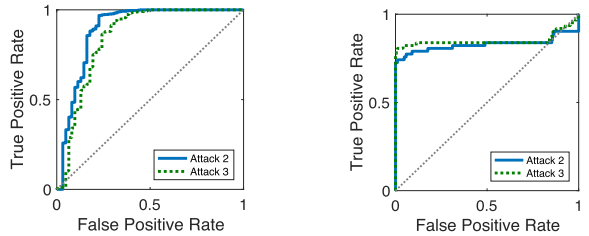
Figure 11(a) shows our results when using the perceptual hash metric. Now, we do not see two large spikes in the perceptual hash difference value. However, we can see when the attack is taking place: notice the correlations in Figure 11(a) (left) where we labeled as attack duration. These values stay close to 0 and are consistent (except for a small spike, perhaps indicating when $\mathcal{V}$ sends a new challenge). When we zoom in this plot via the y-axis, we see more clearly the inter-frame correlations during the attack stay much closer to 0 than when there is no attack. We show this in Figure 11(a) (right). The correlations are also constant and have higher precision than before/after attack. We can effectively classify the attack using $\Delta h \leq \bar{\tau}_{phash}$ from Equation (1).

Figure 11(b) shows our results when using the correlation coefficient metrics. In Figure 11(b) (left), although we can see several spikes, we cannot clearly notice where the attack is taking place, unless we zoom in this plot. We show that in Figure 11(b) (right). Like perceptual hash, we can still notice a change in the correlation values during the attack. We classify the attack with $\Delta h \geq \bar{\tau}_{corr}$ from Equation (2).



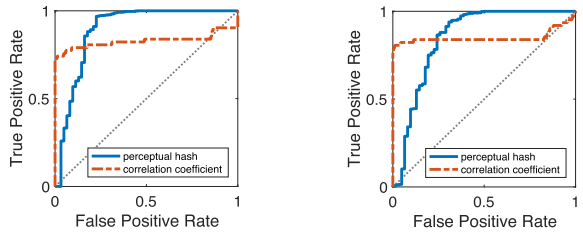
(a) Attack 3 - perceptual hash difference (b) Attack 3 - correlation coefficient

Fig. 11. Detecting attack 3: We calculate the perceptual hash (left) and correlation coefficient (right) between consecutive frames.



(a) perceptual hash metric (b) correlation coefficient

Fig. 12. ROC curve for attacks 2 and 3: using (a) perceptual hash metric and (b) correlation coefficient.



(a) Attack 2 (b) Attack 3

Fig. 13. Results from Figure 12 presented differently: (a) ROC for attack 2 and (b) for attack 3 (using either correlation metric).

To evaluate our binary classifier system, we show receiver operating characteristic (ROC) curves in Figure 12. The ROC curves show the performance of our binary classifier system against different thresholds for $\bar{\tau}_{phash}$ and $\bar{\tau}_{corr}$. These thresholds help us notice correlations that are constant (indicating when attack 2 or 3 is taking place). In Figure 13, we show the same results, but we present based on attack type. Based on our experimental results, we conclude that both the perceptual hash and the correlation coefficient metrics are effective to detect attack 2 and 3.

5 DETECTION OF FORGERY ATTACK 4

The mitigations we proposed so far cannot detect attack 4. Since the attack pastes visual challenges to a *sequence of old* frames, then, we can no longer rely on (1) significant discrepancies nor on (2) persistent constant/or no changes between consecutive frames. Unlike attacks 2 or 3, there are no sudden changes when attack 4 starts nor the attack uses repeated fake frames. In this section, we

(a) top view (b) camera perspective

Fig. 14. Setup for experiments using a tablet displaying visual challenges (in this case a QR code) and a smart light bulb to constantly change the scenery ambient color. Here the color is cyan. [Note: images contain color.]

Fig. 15. Illustration of changing the ambient lighting using a smart light bulb with a color palette of over 16 million colors. The attacker would need to copy the QR code challenge from the current frame and paste to an old frame that matches the *expected* color challenge. Simply matching the color is not enough, because the frame must also be *highly* correlated. [Note: image contains color.]

combine inter-frame correlations with a more robust visual challenge—that reflects in the whole image frame and not only on a small portion—to defend against attack 4.

5.1 Introducing Full-Image Visual Challenges

We propose adding a visual challenge in the form of changing the ambient color lighting (of the area under surveillance). The verifier controls the physical environment color lighting and verifies whether the changes reflect in the video feed. In our experiments, we use a smart light bulb that can transmit over 16 million colors. We show our setup in Figure 14. In Figure 14(a), we show the smart light bulb reflecting the color cyan, and in Figure 14(b), we show the perspective from the camera. As we can see in these RGB figures, the color cyan reflects in the scenery the camera is monitoring.

The color lighting also reflects in the video feed. Unlike QR codes that appear in a small area, color lighting overlays the entire image frame. For the remainder of the article, we refer to this new visual challenge as full-image challenge. We show sample image frames from our experiments in Figure 15. Notice the colors in each frame are due to changes in the physical environment, not by editing each frame with a color filter. We label the frames with respective color names for clarification. As we show in our experiments later, we found it is not trivial for an attacker to recreate these "color filters" into a fake frame to fool the verification.

Algorithm 2 outlines the steps to create full-image challenges. The generation phase is straight forward: $\mathcal{V}$ chooses a random color (e.g., RGB value for cyan: (0, 255, 255)) as the next visual challenge (line 2), and changes the smart light bulb to that color (line 4). $\mathcal{V}$ continuously repeats this process.

Assumptions and Limitations. We can either use full-image challenges by themselves or to complement other challenges. In our experiments, we use them to complement QR codes and mitigate attack 4.

Changing the lighting color of a physical environment can be intrusive when compared to other proposed challenges such as QR codes, random text, or tweets likely present in public spaces. Under certain scenarios (e.g., entrance to a nuclear enrichment facility that is empty most of the time), it might be acceptable to change the environment color, whereas it will not be suitable for

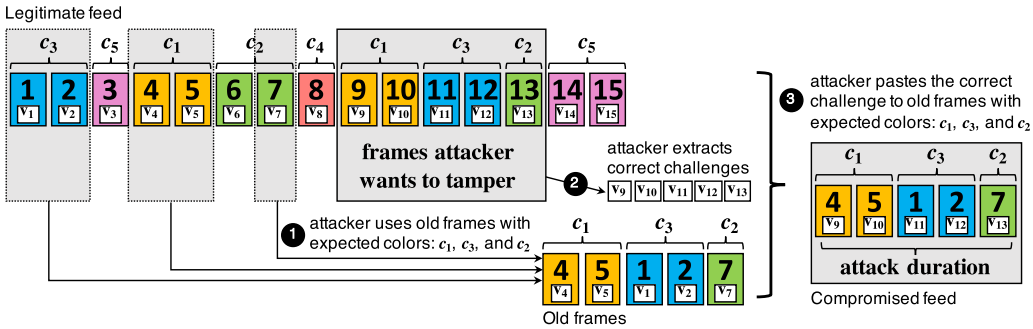


Fig. 16. Attack 4: Illustration of attack 4 using ambient color lighting. When the attack starts, the attacker (1) selects a sequence of old frames with the same color as the current legitimate frames, (2) extracts the current correct challenges, and (3) pastes the correct challenges to the old frames. [Note: image contains color.]

ALGORITHM 2: Full-Image Challenge

Verifier $\mathcal{V}$ modifies ambient lighting (in the area under surveillance) and thwarts copy-paste attacks on the video feed.

Generation Phase

1: **for** $i \in \mathbb{N}_n$ **do**
2: $c_i \leftarrow \{c_r \in \text{Color}\}$ ▷ $\mathcal{V}$ picks a random color $c_r = \{R_r, G_r, B_r\}$ and assigns to c_i
3: store current time t_i
4: send c_i to light bulb ▷ $\mathcal{V}$ sends c_i to a smart light bulb near the camera
5: wait for random time t_i^r
6: **end for**

other settings such as the entrance of a crowded apartment building. To make color challenges more discreet it is possible to quickly alternate between *opposite colors* to simulate the lack of color in the environment and cancel out the colors—the camera would still capture the colors, but it would be less noticeable in the physical environment. Another possibility is to completely hide the visual challenge as we describe later.

5.2 Attack 4 Implementation Details

Recall attack 4 captures the correct visual challenges (e.g., `QR code`) and inserts them to a sequence of fake frames—significantly correlated to the legitimate frames—such as old frames. The lighting introduced in the physical environment forces the attacker to choose a sequence that contains the expected color. To make the attack harder to succeed, the verifier constantly changes the light bulb color. This forces the attacker to use a different sequence of old frames every time the color changes.

We illustrate this attack in Figure 16: The attacker forges visual challenges to a sequence of old frames with color `yellow` (e.g., c_1), then of color `blue` (c_3) and `green` (c_2). Notice that a less resourceful attacker may attempt to reuse a single frame until the color changes instead of a *sequence* (e.g., reuse a single `yellow` frame until the color changes, then reuse one `blue` frame, etc.). Such attack gives us similar effect as attacks 2 and 3, and we consider them solved.

The attacker cannot trivially copy-paste `full-image` challenges as they would with `QR codes`. The attacker needs to craft their attack more intelligently: first, detect the overall color of the legitimate frame (the attacker wants to replace) and then, find an old sequence matching the expected color.

Algorithm 3 describes generic steps for the attack. During each step, attacker $\mathcal{A}$ detects the color of the current frame F_c (`line 3`) using their own trained classifier (`line 4`). Then, attacker $\mathcal{A}$ finds an old frame F_x most correlated to F_c and with expected color (`line 5`) to create the forged frame (`line 6`).

Recall the attacker wants to also select frames with *acceptable* inter-frame correlation values (so our previous mitigations fail). To ensure this, once the attacker finds a good old frame F_x, the attacker can use subsequent frames (e.g., F_{x+1}, F_{x+2}) for the next fake frames. This strategy works while subsequent frames contain the expected color, otherwise the verifier detects a color mismatch. Recall verifier $\mathcal{V}$ constantly selects a new random color—unknown to attacker—every random time t_i^r. This forces the attacker to keep history of *multiple* sequences of old frames: a sequence for each expected color. When we revisit Figure 16, we notice that although the attacker picks an old frame with (1) matching color and then (2) subsequent frames, we can see interruptions in the sequence of frames. For example, between legitimate and/or fake frames tagged with visual challenge pairs: (v_8, v_9), (v_{10}, v_{11}), (v_{12}, v_{13}), and (v_{13}, v_{14}). Their inter-frame correlations reveal breaks in the sequence. As we show later in our results, this enables us to detect the attack. To maximize detection and minimize the attack, verifier $\mathcal{V}$ can update color challenges more frequently.

ALGORITHM 3: Attack: Ambient Lighting

Attacker $\mathcal{A}$ modifies video feed with same color lighting and high correlated frames.

Generic Attack

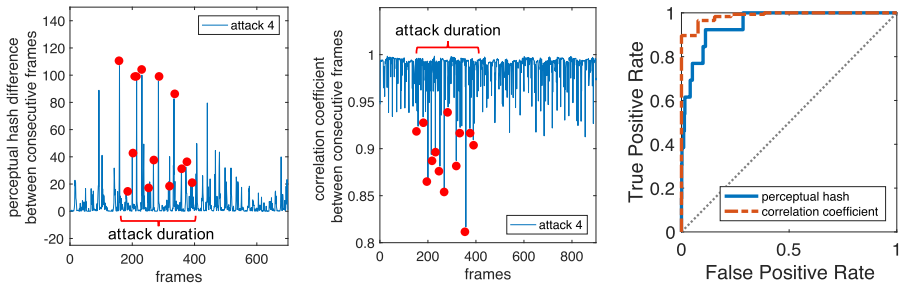
1: **while** copy-paste attack **do**
2: $F_c \leftarrow$ next current image frame ▷ $\mathcal{A}$ intercepts the current frame F_c from surveillance camera
3: $\{R, G, B\} \leftarrow$ `attackDetectColor`(F_c) ▷ $\mathcal{A}$ runs own algorithm to detect overall frame color
4: $c_a \leftarrow$ `attackPredictLabel`$(\{R, G, B\})$ ▷ $\mathcal{A}$ uses own trained classifier and predicts a color label c
5: $F_x \leftarrow \text{argmax}_{F_x \in \texttt{OldFrames}}\, f(x) = \texttt{corr}(F_c, F_x)$ ▷ $\mathcal{A}$ finds an old frame of color c_a most correlated to F_c
6: $F_a \leftarrow$ `forge`(F_x, F_c) ▷ $\mathcal{A}$ copies visual challenge from F_c and pastes to F_x
7: replace current frame with forged frame F_a ▷ $\mathcal{A}$ replaces current frame with forged frame F_a
8: **end while**

In our experiments, we show that even when the attacker chooses old sequence of frames with the same color lighting, their correlation with the previous frame is significant enough to reveal the attack.

5.3 Using Inter-frame Correlations to Detect Attack 4

We now describe our verification for when we use `full-image challenges`. We refer the reader to Algorithm 4, for the overall steps. Once verifier $\mathcal{V}$ receives the next image frame, $\mathcal{V}$ recognizes the overall color in the frame and predicts a color label (`lines 3-4`), and then verifies whether the predicted color matches the expected one (`line 6`). Once $\mathcal{V}$ verifies the frame contains the correct color and that the response arrived within an expected response time, then $\mathcal{V}$ calculates the correlation between current frame F_j and previous frame F_{j-1} (`line 7`). As we did in the previous mitigations, we use either perceptual hash difference or correlation coefficient. If the correlation is outside an expected range, then we raise an alarm (`line 9`).

Since we assume the attacker does not repeat fake frames, then we cannot rely on constant changes in the correlation value. So, we use only parts of Equations (1) and (2): namely, $\Delta \text{h} > \hat{\tau}_{phash}$ or $\Delta \text{h} < \hat{\tau}_{corr}$, depending on the metric.



(a) Attack 4 - perceptual hash (b) Attack 4 - correlation coefficient (c) ROC curve for attack 4

Fig. 17. Detecting attack 4 using: (a) perceptual hash and (b) correlation coefficient metrics. We show their performance in panel (c).

ALGORITHM 4: Full-Image Challenge

Verifier $\mathcal{V}$ modifies ambient lighting (in the area under surveillance) and thwarts copy-paste attacks on the video feed.

Verification Phase

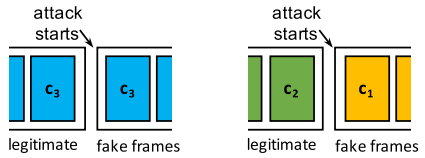
for $j \in \mathbb{N}_n$ **do**
  $F_j \leftarrow$ next image frame ▷ $\mathcal{V}$ receives an image frame F_j from surveillance camera
  $\{R_j, G_j, B_j\} \leftarrow$ `detectColor`(F_j) ▷ $\mathcal{V}$ detects the overall color (i.e., RGB values) from frame F_j
  $c_l \leftarrow$ `predictLabel`$(\{R_j, G_j, B_j\})$ ▷ $\mathcal{V}$ uses a classifier (previously trained) to predict color label c_l
  store current time t'
  if `verify`(c_l, t') **then** ▷ $\mathcal{V}$ verifies perceived color c_l and if response is within expected time
    $\Delta\delta =$ `corr`(F_j, F_{j-1}) ▷ $\mathcal{V}$ calculates the inter-frame correlation between F_j and F_{j-1}
    if ($\Delta\delta$ is not within expected value) **then**
      trigger an alert ▷ $\mathcal{V}$ alerts if correlation is outside range (for expected color)
    end if
  else
    trigger an alert ▷ $\mathcal{V}$ triggers an alert if classified color (from perceived frame) is incorrect
  end if
end for

5.4 Experimental Results for Detecting Attack 4

Experiment: During each try, we capture frames while our verifier $\mathcal{V}$ code updates `full-image challenges`. This enables us to have old frames in varied colors to later create the attacks. In our experiments we use: `red`, `green`, `blue`, `cyan`, `magenta`, and `yellow`. We then let the camera capture 900 frames, and we forge frames F_a through F_{a+n}. We create our forgeries in `MATLAB`: first, we recognize the legitimate frame color (we train a multiclass Naive Bayes classifier using `fitcnb` and `predict` a color label), and select an old fake frame F_x containing the same color. Then, we replace subsequent legitimate frames with frames succeeding F_x such as F_{x+1}, F_{x+2}, and so on. When the color challenge changes, we select a new sequence of old fake frames matching the new color.

Then, we run the verification phase from Algorithm 4. First, using the perceptual hash difference, and then the correlation coefficient. We plot each $\Delta\delta$ outcome value, and show our results next.

Figure 17(a) shows our results when using perceptual hash difference. Recall the attacker is forced to pick a new sequence of old frames each time $\mathcal{V}$ changes the color challenge. The red



(a) Strategy 1 (b) Strategy 2

Fig. 18. Attack 4: We study two strategies for the attack. (1) Both the fake frame and previous legitimate frame have the same color challenge, and (2) the attack starts when there is a color change such that the legitimate and fake frames do not share the same color.

dots denote the perceptual hash difference during these changes in color challenge. Figure 17(b) shows our results when using the correlation coefficient metric. Again, the red dots denote when the attacker picks a new sequence of old frames. Differently from perceptual hash, we can see a consistent change in the correlation values every time $\mathcal{V}$ updates the color challenge—the correlation values are smaller than (1) before/after the attack and (2) while the attacker leverages consecutive frames of the same color. To maximize detection, $\mathcal{V}$ can update color challenges more frequently, so we see more of these red dots. In Figure 17(b), we show the performance of our binary classifier system against different thresholds for $\Delta h > \hat{\tau}_{phash}$ and $\Delta h < \hat{\tau}_{corr}$. Notice the correlation coefficient metric performs better than the perceptual hash difference metric to detect attack 4.

Additional Experiments. To conclude our experiments, we study in depth the correlation between a legitimate frame (the attacker wants to forge) and a fake old frame. We show it is difficult for an attacker to find a good fake old frame (of the expected color) to bypass our verification while our detection actively sends `full-image challenges`.

We consider two strategies to start the attack: (1) the fake old frame and previous (legitimate) frame have the same color challenge, and (2) the attack starts when there is a color change (i.e., the legitimate and fake frames do not share the same color). Figure 18 illustrates these two strategies. Our results show it is not trivial for an attacker to find old frames that are correlated *enough* with the expected current frame—even when the fake frame shares the same color and similar scenery with the previous legitimate frame—to successfully bypass our verification. Notice these strategies also describe the end of the attack. The last forged frame may be of the same or distinct color as the next legitimate frame.

Strategy 1 to start/end attack 4. For our experiments, we let the camera capture 900 frames. We then select frames F_a through F_{a+i} to forge. Instead of simply selecting an old frame for the forgery, we calculate the correlation between F_{a-1} (frame before attack starts) and each old frames of matching color. During each run, we use 15–20 old frames that we captured 15min prior while the scenery stayed unchanged. So, we have a total of 15–20 correlation values. In these experiments, we use the correlation coefficient metric, because as we saw it performs better for attack 4.

We keep record of the biggest correlation found (best-case scenario for the attacker), the smallest correlation (worst case scenario), the average between all computed correlations, and the expected correlation (when there is no attack).

Then, we select F_a as the attack start, and repeat this process until F_{a+i}. As a result, we keep on selecting a new place to start the attack, while calculating the correlation between the new position and old frames. Figure 19 shows our results for color challenge `blue`. The blue (top) dots represent each new attack start. Notice the expected correlations (solid lines) are significantly larger than best-case correlations for the attacker (to find a good old frame to start/end attack 4).

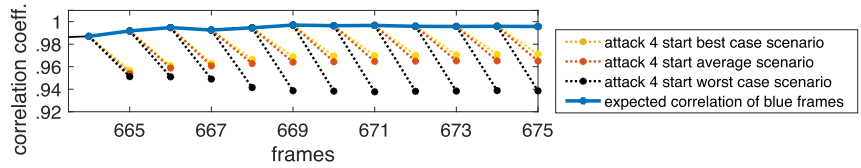
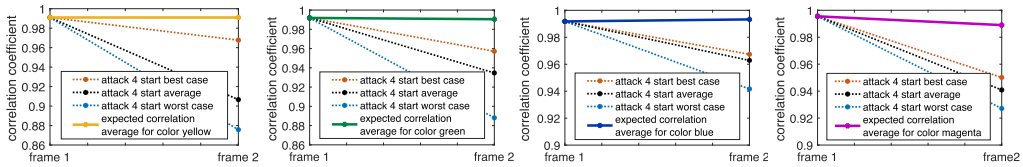


Fig. 19. Attack 4: We show the best-case/average/worst case scenarios to start attack 4 in various time periods.



(a) Yellow ambient light (b) Green ambient light (c) Blue ambient light (d) Magenta ambient light

Fig. 20. Attack 4—Attacker needs to find an old frame that has the same ambient lighting and is highly correlated with the previous frame (accepted by the verifier). Our results show that this is not trivial, since best-case scenarios are lower than expected values.

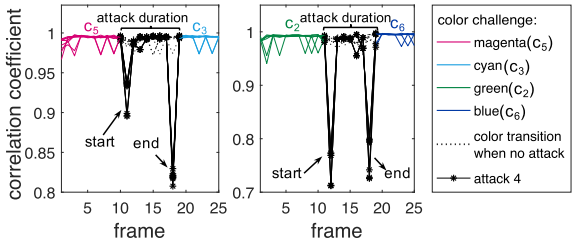


Fig. 21. Attack 4—In our experiments, we analyzed the correlations during color transition, and simulated attacks were the attacker uses the corresponding transition pattern (i.e., magenta to cyan) to launch the attack. We found that it is possible to see spikes when the attack begins and ends, even when the attacker chooses old frames that follow the same transition.

Figure 20 shows our results for color challenges: `yellow`, `green`, `blue`, and `magenta`. We calculate the average for the best case/average/worst case scenarios—per color—to start attack 4. Although the average correlations may vary according to the color challenge, our results show that the best-case scenario (for the attacker) is smaller than the expected correlation and not good enough to remain undetected. This means we can notice a decrease in the correlation value when the attack starts (even when they pick the most correlated frame), and detect the attack.

Strategy 2 to start/end attack 4. Now, we focus on when the attack starts during a transition between two color challenges. For example, the verifier sends challenges of color: `magenta` and then `cyan`, or `green` and then `blue`. We assume the attacker has captured sequences of these same color transitions. When the attacker sees a `blue` color challenge changing to `green`, the attacker will select an old `green` frame that was transitioned from `blue`. We consider transitions because intermediate frame (between two color challenges) undergo different color transformation. In Figure 21, we show what happens when the attacker tries to use old frames to launch the attack. Here both figures show the transition between two colors: `magenta` to `cyan`, and `green` to `blue`. We capture multiple sequences for each color transition, and graph what those transitions look like with no attack. Then, we select old frames containing the same color transitions and start the

attack (after the last frame of the first color appears). We compare the correlation between the last magenta frame and first cyan frame (from an old transition sequence), and we keep calculating the correlation between consecutive frames from the old sequence. Then, we calculate the correlation between last fake cyan frame and first cyan frame after attack (like strategy 1). We clearly see a drop in the correlation values. As a result, we can detect when the attack starts and ends.

Our results show that full-image challenges difficult a skillful attacker to be successful. It forces the attacker to use frames with expected "color filter." Moreover, it is not trivial for the attacker to find an old frame with matching color challenge good enough to bypass our verification. As a result, we can mitigate attack 4 using full-image challenges.

Discussions. A common practice in security includes: to select the appropriate security mechanism to protect a system will depend on a trade-off between the sensitivity of the application we want to protect *versus* the added costs, challenges associated with the security protection. If we want to protect the video feed of cameras in sensitive settings (e.g., surveillance cameras monitoring computers that generate cryptographic keys) and the room is an area that has very few people entering the room, then we can go for the more *invasive* security protections against the strongest adversary (e.g., our proposal in Section 5); in contrast, if we have lower sensitivity applications, then we can assume weaker adversaries and deploy security protections that minimize any drawback (e.g., our proposal in Section 3). To consider this trade-off, we deliberately proposed a modular and extensible architecture (recall architecture in Figure 1) that allows customizable deployment: e.g., we can (1) deploy our system in settings of various levels of sensitivity, (2) defend against increasingly levels of adversaries, and also (3) take in consideration usability concerns. Note that this architecture allows us to deploy separate types of *media* to the physical environment (to transmit visual challenges), so, the visual challenges we can use in our defenses will depend on each particular deployment. For example, a deployment using a standard digital monitor may use QR code and/or random text (or tweet) challenges; a deployment containing a smart light bulb may use full-image challenges; and a deployment with infrared LEDs integrated in the environment may use IR-challenges. Although we consider these decisions to be implementation details, in this article, we present experimental results showing how we can use various visual challenges to protect against naive and forgery attacks.

6 RELATED WORK

Video Forensics. Recent efforts in video forensic analysis can be broadly classified in two categories: (a) methods exposing naive attacks, and (b) methods revealing anti-forensic attacks. Most previous forensic attack-detection tools are designed against naive attacks such as deleting/inserting frames, duplicating complete frames, and replacing frames with different frames captured using other camera.

Assuming Naive Attacks. We categorize existing forensic approaches to expose naive attacks in four classes:

Class 1: Detecting deletion/insertion of frames. Wang and Farid (2006) propose one of the initial video forensic approaches. Their approach relies on group-of-pictures (GOPs)[1] structure variation in MPEG recompressions: when video is re-compressed after inserting or deleting a set of video frames, the GOP structure gets distorted. In an unaltered video, $\mathcal{P}$-frames in a GOP are generally

[1] GOP is a collection of successive image frames and contains image types such as an $\mathcal{I}$-frame (a complete image frame and first frame of a GOP sequence) and $\mathcal{P}$-frames (an *incomplete* frame containing only the changes from the previous frame, e.g., color changes and content modifications). Image frames are grouped in GOPs of a particular size, and a compressed video stream consists of successive GOPs. We refer the reader to Haskell et al. (1996); Marshall (2011) for additional background information.

correlated with the initial $\mathcal{I}$-frame in the same GOP. However, when frames are deleted or inserted, some $\mathcal{P}$-frames move from one GOP to another GOP after re-compression. Hence, their correlation with the initial $\mathcal{I}$-frame of the current GOP is smaller, resulting in larger prediction errors. Since Wang and Farid assume a fixed GOP size, deleting frames result in a constant shift of $\mathcal{P}$-frames throughout the video and $\mathcal{P}$-frames prediction errors exhibiting a periodic behavior—which we observe through Fourier analysis. *Limitations*: this approach requires human inspection (of $\mathcal{P}$-frames prediction errors), and is susceptible to human errors (Stamm et al. 2012).

Gironi et al. (2014) also suggest a video forensic technique to detect frame insertion and deletion. Their approach focuses on video encoding compression. An encoder divides a frame in macroblocks (MBs) and encodes them separately as: (1) $\mathcal{I}$-MBs (encodes without temporal prediction), (2) $\mathcal{P}$-MBs (encodes in reference to other frames), or (3) $\mathcal{S}$-MBs (copies directly from a previous frame). Gironi et al. then define a variation of prediction footprint (VPF) based on the number of $\mathcal{I}$-MBs and $\mathcal{S}$-MBs. The variation is denoted by the $v(n)$ signal. The authors rely on the autocorrelation of $v(n)$ to identify periodicity in the signal and expose forgeries.

In our work, since we constantly send visual challenges to the physical environment the camera is monitoring, if we do not see a sequence of challenges we sent—because frames were deleted—then we can detect the deletion. To insert extra frames, the attacker needs to fabricate visual challenges or reuse previously sent challenges. As we show in this article, we can detect when an attacker adds an incorrect challenge or copy-paste a legitimate challenge to a fake frame.

Class 2: Detecting duplication of frames. Wang and Farid (2007) propose a forensic technique to identify frame duplication (also referred as "inter-frame copy-move attack") in a stored video offline. This method works in two stages: first, it divides a video sequence into a number of overlapping sub-sequences with a shift of one frame. Their approach computes: (1) the temporal correlation between each frame to form a correlation matrix T_i for each sub-sequence, and (2) the correlation between all possible sub-sequence combinations. In the second stage, they divide each video frame to m non-overlapping blocks. Last, they compute the block-wise correlation between frames—of the same index—in different sequences (e.g., tenth frame from two sub-sequences). If the correlation values are higher than a threshold, then they classify the video sequence as having duplicates frames.

Yang et al. (2016) also propose a method for detecting frame duplication in videos offline. Their approach is based on feature similarity. It divides a video into over-lapping (by one frame) subsequences. Next, it computes the singular value decomposition (SVD) features for each frame and computes their distance to the SVD features of the first frame. Their approach then computes the correlation between the distance and each sub-sequence combination to form a correlation matrix. Two sub-sequences with a high correlation are classified as duplicates.

Our work is similar in the sense that we also calculate the correlation between successive image frames. However, it is different because our verification is *active* and takes place *online* while the camera streams the current video. Further, we do not calculate the correlation between a current frame and all old frames from a subset (to detect duplicates). Instead, we focus only on the correlation between consecutive frames, because (1) we assume the attacker repeatably uses duplicate frames (not sporadically) during attacks 2 and 3 while copying-pasting the expected challenge and (2) we can detect attack 4—which duplicates old sequences of frames—by constantly changing color challenges and noticing correlation discrepancies. Last, calculating the correlation of consecutive frames is less computational expensive than calculating the combination of multiple frames.

Class 3: Detecting camera source. Mondaini et al. (2007) propose a method to identify whether all frames in a video stream were captured with the same camera. They also propose to use photo response non-uniformity (PRNU) noise to detect forgeries in the video. First, they use a few of the video frames to determine a characteristic PRNU pattern that corresponds to the camera. Then,

they calculate three correlation coefficients: (1) correlation between PRNU noise pattern of each frame with the reference frame, (2) correlation between PRNU noise pattern of consecutive frames, and (3) correlation between consecutive frames. Then, they use a combination of the correlation coefficient values to determine video doctoring. Their approach identifies when a frame is taken with a different camera and is malevolently inserted to a video feed. Their method continues to work for compressed videos (e.g., MPEG compression).

Class 4: Detecting double quantization. Wang and Farid (2009) propose using *double quantization* to expose forgeries offline. Typically, when we decode, edit, or re-compress a video sequence, we introduce special artifacts in the $\mathcal{I}$-frame coefficients—which can be used to detect forgeries. Wang and Farid first model single compressed de-quantized coefficients as a standard Gaussian distribution in $\mathcal{I}$-frames. Then, they use the expectation-maximization (EM) algorithm to estimate and compare distribution probability for doubly quantized macroblock (MB) with the original distribution of the coefficients. Finally, their approach reveals localized tampering in regions as small as 16×16 pixels. Their detection is highly effective when the ratio between the first and second quantization is greater than a threshold.

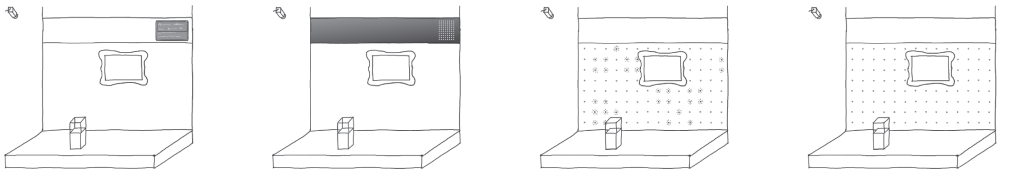
Assuming Anti-forensic attacks. Most works in video forensics focus on detecting naive forgery attacks. They do not consider that the attacker may use anti-forensic operations to remove evidence of the forgeries. There are a few works in the literature that study anti-forensic methods. We present them next:

Among the initial efforts, Stamm and Liu (2011) propose an anti-forensic approach to deceive the forensic algorithm proposed by (Wang and Farid 2006)—which uses motion errors (introduced by changes in the GOP structure) to expose forgery. Stamm and Liu suggest that after an attacker performs the forgery, then (1) we can compute a maximum prediction error, and (2) during re-encoding, we can select a motion vector so the prediction error increases for all frames. As a result, the forgery is indistinguishable for the algorithm proposed in Wang and Farid (2006).

In their later work, Stamm et al. (2012) propose a method to expose traces of the anti-forensic attack described earlier. In general, a video encoder tries to minimize the prediction error. So, the computed motion vector reflects the motion in the scene. However, the anti-forensic approach described before selects motion vectors such that the prediction error increases. Stamm et al. point out that the mismatch between the motion vectors—used by the anti-forensic approach and in the scene—expose the anti-forensic attack.

Kang et al. (2016) improve previous approaches based on periodicity of $\mathcal{P}$-frame prediction error (e.g., improve false positive rates). Further, they propose a counter anti-forensics attack against the work by Stamm and Liu (2011). Kang et al. show that when a video is re-compressed with (1) standard motion estimate and (2) motion compensation, then they can restore footprints removed by an anti-forensics attacker. As a result, they can expose the anti-forensic attack proposed by Stamm and Liu (2011).

Our proposal is complementary to all other proposals on video forensics. It can be used in combination with them to help us identify attacks, but in addition, our proposal is the only one providing strong *freshness* guarantees. One advantage of using visual challenges (as opposed to previous approaches in video forensics that might want to detect duplicate frames or copy-move attacks in video files), is that we can increase our confidence about the freshness and authenticity of a video feed. Using visual challenges is an *active* effort, and thus we can use them to verify *liveness* properties in video feeds, whereas *passive* approaches cannot introduce changes in a scenery while the video is being captured—they usually only have the end product (i.e., video file) when checking for video tampering.



(a) Standard display (b) Hidden infrared display (c) Integrated IR LEDs (on) (d) Integrated IR LEDs (off)

Fig. 22. Options for integrating challenge devices.

7 DISCUSSION AND CONCLUSION

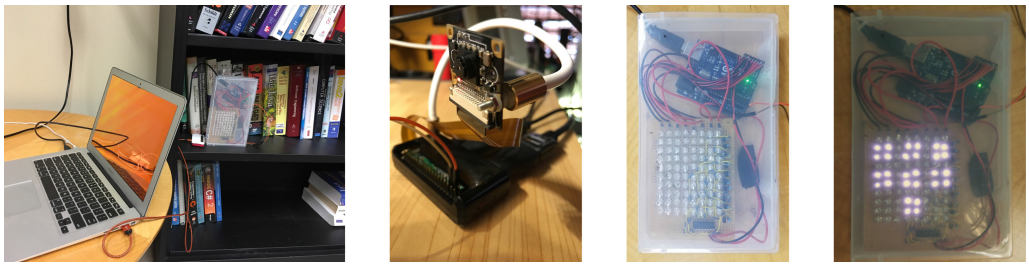
Our work in this article focuses on the very powerful anti-forensic attacker that has been shown to have success in previous work on video forensics. Our approaches using `full-image challenges` can also provide stronger guarantees for detecting video tampering by attackers and make launching successful attacks much harder than the current state-of-the-art in video forensics.

The attacks we consider in this article assume the attacker knows that the defender is using visual challenges. Sending `QR codes` or random letters to a display as previously proposed (or even changing the color of the environment as we propose in this article) is a fairly conspicuous way to send visual challenges. Not only they might annoy people nearby, but they also can be noticeable to an attacker doing reconnaissance—of a potential target—to identify whether a camera is leveraging visual challenges to protect its video footage.

In future work, we plan to explore approaches to minimize the chances the adversary will know that the defender is using visual challenges. We illustrate this concept in Figure 22. One approach is to use infrared (IR) light to create and display the visual challenges (Figure 22(a)). We refer to these challenges as `IR-challenges`. We can also place the device (displaying `IR-challenges`) behind a filter—which allows infrared light to pass through while blocking visible light. This conceals the challenge from human eyes. We can devise to integrate the filter and display in the environment as if they were part of it. For example, we can run a strip of opaque glass along a portion of a wall while behind one section, we have a matrix of infrared `LEDs` displaying `IR-challenges` (Figure 22(b)). Moreover, we can expand the matrix of infrared `LEDs` in size to integrate a larger area in the scene under surveillance—to assist preventing copy-paste attacks. Ultimately, we can expand the matrix to cover an entire wall, as needed. The `LED` matrix can be further recessed behind holes in the wall and placed behind glass panes (Figures 22(c) and 22(d)). In this way, the background of the scene (under monitoring) can become the device displaying the challenge.

To initiate research in this direction, we constructed a `LED` matrix (Patel 2015). Instead of using visible light `LEDs`, we used infrared `LEDs` operating at 940nm. This wavelength renders `LEDs` invisible to the human eye while still maintaining visibility with infrared sensitive cameras. We show our prototype in Figure 23.

We use an Arduino UNO R3 and Raspberry Pi Zero W to control our infrared `LED` matrix. The Arduino controls the matrix directly while the Raspberry Pi sends the `IR-challenges` to the Arduino. The Raspberry device sends the challenges as an array of bytes corresponding to a raw binary pattern to display in the IR `LED` matrix. Meanwhile, the verification code is running on a laptop, as we show in Figure 23(a). We use a Raspberry Pi camera—with no IR filter—for our experiments with `IR-challenges` (Figure 23(b)). Figure 23(c) shows what people see when looking at our infrared `LED` display, and Figure 23(d)—captured by our Raspberry Pi camera—shows what the camera sees. In our prototype, the infrared `LEDs` are still fairly noticeable: people can see the `LED` matrix (although they cannot see that it is on and displaying visual challenges). Nonetheless, recall that a real-world implementation of our idea (as we illustrate in Figure 22) can hide the infrared



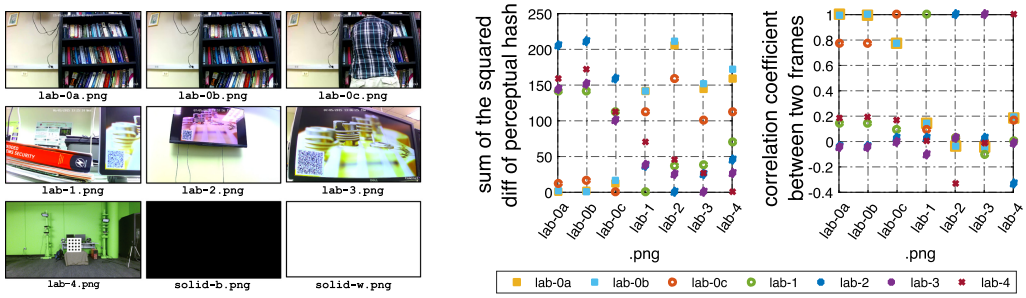
(a) Verifier sends IR- challenges to our LED matrix. Night-vision cameras can perceive the IR-challenges.

(b) Raspberry Pi camera with no IR filter (i.e., night vision module).

(c) People cannot see the challenge in the IR LED board.

(d) Cameras sensitive to infrared can see IR visual challenges.

Fig. 23. Our prototype for hiding visual challenges from human eyes.



(a) (b)

Fig. 24. (a) Sample image frames. (b) We use phash (left) and corr (right) to calculate difference/similarity among sample frames.

emitters from the view of people nearby. We plan to continue this line of work in the future—to balance the convenience of unobtrusive visual challenges, with the security guarantees that they provide.

A APPENDIX

Perceptual hash vs. correlation coefficient comparison. To compare the performance between perceptual hash difference (phash) and correlation coefficient (corr) metrics, we calculated the phash and corr values among nine sample images. We show our sample images in Figure 24(a). We considered the sample images under various resolutions (e.g., 480p, 720p, 1080p). Since the results are similar, we show our results for when using sample images of 720p.

We show our results in Figure 24(b). The x-axis represents each sample image, as we label with the image name (e.g., lab-0a.png, lab-0b.png). The y-axis shows the phash (left) and corr values (right) between the image labeled in the x-axis against the sample images (shown by each legend symbol).

These plots show that similar images (e.g., lab-0a and lab-0b) have a low phash difference score (e.g., phash $<$ 20) and high corr similarity score (e.g., corr $\sim=$ 1). Further, these plots show that images that are different (e.g., lab-0a and lab-2) have a higher difference score (e.g., phash $>$ 200) and extremely low correlation score (e.g., corr $\sim=$ 0). We omit our results comparing sample images with images with all black (solid-b.png) and white pixels (solid-w.png). Their phash values were significantly larger (phash $>$ 2,000) while corr values were equal to constant NaN (not

a number)—since there is no variation in all-black/white frames, the calculation simplified to zero divide by zero (i.e., NaN).

Our results show us that: (1) the correlation coefficient metric classifies more coarsely than the perceptual hash difference metric—similar images have a high correlation value, while everything else has corr close to 0. Unlike phash, (2) correlation coefficients reveal information about the frame rate (fps) used when capturing the video feed. By plotting the corr values, we can see periodic spikes revealing what frame rate the camera uses (e.g., whether the camera uses 15, 30, or 90fps). Notice that we can incorporate this information to detect further discrepancies in the sequence of frames to identify forgeries. Finally, (3) corr is faster than phash. It takes on average 29.04 ms to calculate corr between two images while 1.22s to compute the phash value between two images. We recommend corr over phash.

REFERENCES

Martín Abadi and Roger Needham. 1994. Prudent engineering practice for cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE, 122–136.

Denso Wave Incorporated. 2015. QR code error correction feature. Retrieved from http://www.qrcode.com/en/about/error_correction.html.

Alessandra Gironi, Marco Fontani, Tiziano Bianchi, Alessandro Piva, and Mauro Barni. 2014. A video forensic technique for detecting frame deletion and insertion. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'14)*. IEEE, 6226–6230.

Dan Goodin. 2012. A Fort Knox for Web crypto keys: Inside Symantec's SSL certificate vault. Retrieved from https://arstechnica.com/security/2012/11/inside-symantecs-ssl-certificate-vault/.

Dan Goodin. 2015. Prosecutors suspect man hacked lottery computers to score winning ticket. Retrieved from https://arstechnica.com/tech-policy/2015/04/prosecutors-suspect-man-hacked-lottery-computers-to-score-winning-ticket/.

Barry G. Haskell, Atul Puri, and Arun N. Netravali. 1996. *Digital Video: An Introduction to MPEG-2*. Springer Science & Business Media.

Xiangui Kang, Jingxian Liu, Hongmei Liu, and Z. Jane Wang. 2016. Forensics and counter anti-forensics of video inter-frame forgery. *Multimedia Tools Appl.* 75, 21 (2016), 13833–13853.

Dave Marshall. 2011. MPEG Compression. Retrieved from https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node255.html.

Yilin Mo, Rohan Chabukswar, and Bruno Sinopoli. 2014. Detecting integrity attacks on SCADA systems. *IEEE Trans. Control Syst. Technol.* 22, 4 (2014), 1396–1407.

Nicola Mondaini, Roberto Caldelli, Alessandro Piva, Mauro Barni, and Vito Cappellini. 2007. Detection of malevolent changes in digital video for forensic applications. In *Proceedings of the SPIE Conference on Security, Steganography, and Watermarking of Multimedia Contents IX*. 6505, 65050T.

Bryan Parno. 2008. Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd Conference on Hot Topics in Security (HotSec'08)*. USENIX, 9:1–9:6.

Sohil Patel. 2015. How to Build an Arduino LED Matrix in 3 Simple Steps. Retrieved from https://diyhacking.com/arduino-led-matrix/.

Santa Clara County Sheriff. 2013. PG&E Substation Surveillance Video. Retrieved from https://www.youtube.com/watch?v=RQzAbKdLfW8.

Yasser Shoukry, Paul Martin, Yair Yona, Suhas Diggavi, and Mani Srivastava. 2015. PyCRA: Physical challenge-response authentication for active sensors under spoofing attacks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1004–1015.

Matthew C. Stamm, W. Sabrina Lin, and K. J. Ray Liu. 2012. Temporal forensics and anti-forensics for motion compensated video. *IEEE Trans. Info. Forensics Secur.* 7, 4 (2012), 1315–1329.

Matthew C. Stamm and K. J. Ray Liu. 2011. Anti-forensics for frame deletion/addition in MPEG video. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'11)*. IEEE, 1876–1879.

Ariane Tabatabai. 2015. How much monitoring of Iranian nuclear facilities is enough? Retrieved from https://thebulletin.org/how-much-monitoring-iranian-nuclear-facilities-enough7923/.

Junia Valente and Alvaro A. Cárdenas. 2015. Using visual challenges to verify the integrity of security cameras. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC'15)*. ACM, 141–150.

Junia Valente and Alvaro A. Cardenas. 2017. Remote proofs of video freshness for public spaces. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical Systems Security & Privacy (CPS-SPC'17)*. ACM, 111–122.

Weihong Wang and Hany Farid. 2006. Exposing digital forgeries in video by detecting double MPEG compression. In *Proceedings of the 8th Workshop on Multimedia and Security*. ACM, 37–47.

Weihong Wang and Hany Farid. 2007. Exposing digital forgeries in video by detecting duplication. In *Proceedings of the 9th Workshop on Multimedia & Security*. ACM, 35–42.

Weihong Wang and Hany Farid. 2009. Exposing digital forgeries in video by detecting double quantization. In *Proceedings of the 11th ACM Workshop on Multimedia and Security*. ACM, 39–48.

Fred Weinhaus. 2014. Tests of Perceptual Hash Compare Metric. Retrieved from http://www.fmwconcepts.com/misc_tests/perceptual_hash_test_results_510/.

Jianmei Yang, Tianqiang Huang, and Lichao Su. 2016. Using similarity analysis to detect frame duplication forgery in videos. *Multimedia Tools Appl.* 75, 4 (2016), 1793–1811.

Nan Zhang, Soteris Demetriou, Xianghang Mi, Wenrui Diao, Kan Yuan, Peiyuan Zong, Feng Qian, XiaoFeng Wang, Kai Chen, Yuan Tian et al. 2017. Understanding IoT security through the data crystal ball: Where we are now and where we are going to be. arXiv preprint arXiv:1703.09809.

Received March 2018; revised December 2018; accepted May 2019