Using Visual Challenges to Verify the Integrity of Security Cameras

Junia Valente, Alvaro A. Cárdenas
Erik Jonsson School of Engineering & Computer Science
The University of Texas at Dallas
Richardson, TX
{juniavalente, alvaro.cardenas}@utdallas.edu

ABSTRACT

We propose a new way to verify the integrity and freshness of footage from security cameras by sending visual challenges to the area being monitored by the camera. We study the effectiveness of periodically updating plain text and QR code visual challenges, propose attack detection statistics for each of them, and study their performance under normal conditions (without attack) and against a variety of adversaries.

Our implementation results show that visual challenges are an effective method to add defense-in-depth mechanisms to improve the trustworthiness of security cameras.

1. INTRODUCTION

Security cameras are used in a variety of sensitive settings, including the monitoring of uranium enrichment facilities to verify that countries abide by the Nuclear Non-Proliferation Treaty [41], monitoring access to certificate vaults protecting secret keys [11], monitoring access to the computers generating random numbers for the lottery [12], and monitoring electricity substations [33]. As the sensitivity of usage scenarios and pervasiveness of security cameras increase, we need to ensure they are protected by defense-in-depth mechanisms to ensure captured images are fresh and authentic.

In a typical Hollywood bank heist film, attackers hack the security cameras and replay old footage so that security guards are not able to see the attack taking place. Unfortunately, this scenario is becoming more realistic as the interest for hacking cameras and the expertise of attackers continue to increase. In a recent example, the cameras that monitored access to computers generating random numbers for a lottery system recorded only one second per minute rather than running continuously like normal and prosecutors argue that the defendant tampered with the camera equipment to have an opportunity to insert a thumb-drive into the computers without detection [12]. During a Black Hat conference presentation, security expert Craig Heffner demonstrated how simple it is to exploit network surveillance cameras like a 'Hollywood-style' hacker [15] by freezing the current frame on the administrator panel.

These examples show the need for defense-in-depth mechanisms for security cameras. We argue that once an attacker has compromised the secret keys or the root-of-trust of an embedded device, the attacker can bypass most of the traditional integrity mechanisms that the receiver of the message can use to verify its authenticity. In addition, security mechanisms are not equipped to detect attacks like moving the cameras to point to a different place [26].

To mitigate these problems we propose an independent channel for verification that will detect an integrity violation of the video received even if the camera and its secret information has been compromised, or if it is physically attacked. Our idea is motivated by attestation and freshness protocols where a verifier sends a random challenge to the device and the device replies with a message to prove its freshness and authenticity; however, the novel aspect of our approach is that we do not send the attestation challenge to the device itself, instead we send a visual challenge to the physical area that the camera is monitoring, and verify that the desired changes are reflected in the video feed.

An intuitive idea for a visual challenge is to have a light turning on and off at the discretion of the verifier, and then use the video footage received to observe if the light is on or off. However, this approach is vulnerable to replay attacks. We propose and evaluate two types of visual challenges with enough randomness to prevent replay attacks: plain text—which can be recognized via optical character recognition (OCR) engines such as `tesseract` [42])—and QR codes—which can be decoded via popular barcode readers such as `zbar` [47]. In both cases, the verifier generates a random string of different sizes and either displays it in the display as-is (e.g., plain text image) or encodes the random string into a QR code and displays the QR code in the monitor. The verifier then receives the video feed from the camera, uses either the OCR engine or the QR code reader (depending on the type of visual challenge used) to extract the random string from the image, and then verifies whether the string found in the captured image is the same (or close enough in the case of OCR) as the one it sent as a challenge.

1.1 Contributions

We propose a new defense that leverages unique properties of cameras; in particular, we exploit their ability to capture visual challenges that we can then use to verify the integrity of the image captured. Our system will not replace traditional message integrity or attestation mechanisms, but rather, complement them with an additional defense-in-depth system whose trust is independent from the other components in the camera surveillance deployment, so that even when the camera feed is compromised (e.g., when the attacker obtains the secret keys of the camera) we are still able to detect problems with the camera footage.

Note that because we are not sending the challenge to the device but to the physical world, our approach works

ACSAC '15, December 07-11, 2015, Los Angeles, CA, USA
© 2015 ACM. ISBN 978-1-4503-3682-6/15/12...$15.00
DOI: http://dx.doi.org/10.1145/2818000.2818045

for legacy systems. The challenge is sent implicitly as an input to the prover (i.e., the camera) but the prover does not need to know about it, even though the response will show up in the output of the camera (i.e., even without the prover's knowledge). Therefore our proposal is not only well suited for legacy systems but also for systems where you do not want to (or cannot) change the software or hardware of the camera or the receiving server. Our proposal requires only two devices to be added, a display and a verifier, and it can work without the need to change any parameter in an already deployed system.

We propose a general system architecture with clearly defined trust assumptions and adversary models. We also propose two visual challenges and evaluate their performance with and without attacks. We believe our analysis of sending physical challenges is general enough to be applicable to other embedded sensors, and we will study extensions to other domains in future work.

2. SYSTEM DESIGN

Consider a security camera monitoring a given location (e.g., entrance to a bank safe or certificate vault, restricted areas of a nuclear power plant or the PRNG generator from a lottery company). Our goal is to detect a compromised camera in the presence of an attacker, i.e., a camera that is reporting incorrect information or old data. In order to do this we add two new devices to the currently deployed system: a display (for indoor environments we can use a digital signage and for outdoor environments an LED display) and a verifier which sends a continuous stream of random challenges to be captured by the security camera, as illustrated in Fig. 1.

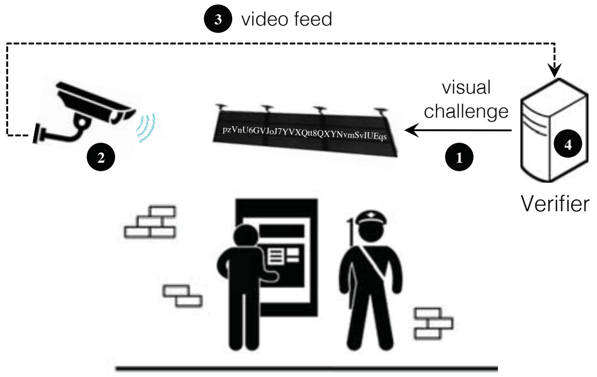


Figure 1: (1) A visual challenge is sent to a display, (2) the camera captures an image which includes the display, (3) the video feed is sent to the control center (where the verifier is located), (4) if the verifier confirms the challenge was captured in the video just received, it gains confidence that the camera is transmitting fresh and authentic footage.

The general design considerations are to (1) reliably detect the visual challenge with high accuracy (no false alarms), and (2) prevent attackers from bypassing our system. We now explain in more detail our design considerations.

2.1 Visual Challenges: QR Code and Plain Text

An intuitive and simple idea of how to verify if a camera is capturing what is really happening in its field of vision is to have an LED, laser or light bulb in a place the camera can see, and then turn the light on or off according to a desired pattern. If the camera captures the light when the light is supposed to be on, and does not capture when it is supposed to be off, then we can improve the trustworthiness of the video feed we are receiving. However, this intuitive visual challenge is very easy to defeat: the attacker needs to simply save one frame with the equivalent of an "on" state and another frame with the "off" state, and replay the frame whenever a challenge is repeated. Therefore, we need to use random challenges with enough entropy to prevent an attacker from recording all frames and then replaying whenever a challenge is repeated.

We propose that the verifier generates a random string and either displays it in the monitor as-is (*plain text challenge*) or encodes the random string into a QR code (*QR code challenge*). QR codes and plain text images have robust decoding algorithms in the form of QR code readers and OCR engines which can be used to efficiently detect the visual challenge. In addition, we need to send a continuous stream of visual challenges in order to guarantee freshness.

QR codes [6, 14] are ISO standards [1] to encode information in an image. Their main features are [6]: 1) high data capacity, 2) small printout size, 3) error correction capability, 4) omni-directional and high speed reading, and 5) structured appending feature.

There are three major considerations that differentiate the use of QR codes and plain text as visual challenges: (1) the amount of information (and hence randomness) that can be conveyed per pixel, (2) the ability to produce an error-free decoding, or a soft decoding with some errors, and (3) the type of display used.

The first consideration is the amount of information that can be displayed per pixel, and QR codes are an efficient way to display that information. Fig. 2 shows the advantages of QR codes in comparison to using plain text: we can encode up to 4,296 alphabetic characters (and 7k+ numeric characters) in the code. The accuracy of decoding the code is not affected as the number of characters encoded increase.

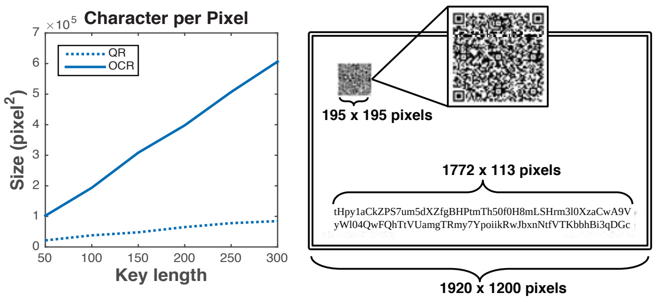


Figure 2: One of the advantages of using QR codes over plain text as a visual challenge is that QR codes occupy a smaller size when compared to plain text, therefore QR codes can communicate a larger capacity of information per pixel square than plain text.

The second consideration is the fact that QR codes use error correcting codes and as such, they produce a binary detection score: that is, they can either correctly detect the code, or produce a decoding error. OCR algorithms on the other hand can produce "soft" detection scores ranging between 0 to 100% detection accuracy. As we will see in our performance evaluation, the ability to work with partially correct decoded visual challenges might be an advantage from a security perspective.

The final consideration is the type of display used to show the challenge. For indoor environments, a digital signage can display both visual challenges, while for outdoor environments, only top of the line LED displays have enough resolution and are large enough to display QR codes; for relatively inexpensive LED displays plain text visual challenges might be the only option available.

2.2 Trust Assumptions

The architecture of our system is shown in Fig. 3. We assume that any device in our system is potentially compromised, and our goal is to detect when the camera is providing false information, or when the footage stored in the control center database has been modified.

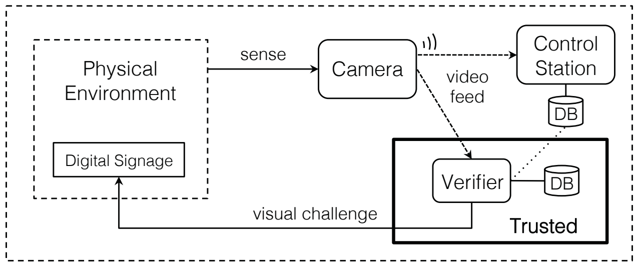


Figure 3: The only trusted entities in our system are the verifier, which sends the visual challenge to the display, and the database (if necessary) that stores the history of challenges and the time they were sent for forensics purposes.

Our system can provide security to *online* (attacking the camera) and *offline* (attacking the control center storage) attacks as long as the verifier is trusted. To detect offline attacks modifying the footage stored in the control center we also require a place to securely store the history of challenges (and the time) sent to the display. For offline attacks we also need to assume that the verifier obtains the time from a trusted source. It is not necessary for the database of the verifier to be secure (e.g., if we store digitally signed messages by the verifier).

2.3 Adversary Model

We want to minimize the chances the attacker can use previously recorded footage and present it as new. Therefore our system needs two requirements: (1) the random challenge needs to have enough entropy to prevent an attacker from recording all messages and then replaying whenever a challenge is repeated (we use alpha-numeric random strings of length l in our experiments), and (2) the verifier needs to refresh the random challenge periodically (in a matter of seconds) to prevent the attacker from recording the first time the challenge appears in the display and then replaying the first frame for the expected duration of that challenge. We discuss the importance of these two parameters in the section focusing on the security analysis of our implementation.

In our implementation section we also consider the performance of our system against three types of attackers who have compromised the camera. We first consider a naive attacker that does not know our system is in place and will launch a false data injection attack without taking into consideration the visual challenge, then we consider an attacker that knows our system is in place and will launch an attack as long as it is not detected, and finally, we consider an attacker who tries to impose the legitimate version of the visual challenge on top of a forged video feed.

2.4 Attack Detection

The ability to send a challenge continuously means that even if our detection mechanism does not recognize one particular correct response to the challenge (a false alarm), the accumulation of errors over a period of time will still be low if we are not under attack; while a consistently high mismatch over a period of time between the computer vision algorithm and the challenge should be an indication of an attack.

There are two possible errors in QR codes: either the decoding of a message is different from the one we sent or there is an error decoding the QR code. Because under normal (non-attack) circumstances we never decode a challenge different from the one we sent, then we can claim we are under attack whenever we do decode something other than what we sent. Similarly if we have too many decoding errors over a period of time, we can raise an alarm. A representation of this attack detection state machine is shown in Fig. 4.

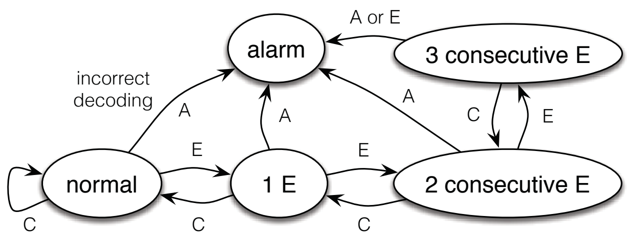


Figure 4: Anomaly detection for QR code visual challenges. Legend: C = correct decoding, A = decoding of a value different to the challenge (i.e., a high indicator of an attack), and E = cannot decode QR code in the current image frame.

In contrast to the all or nothing decoding from QR codes, OCR gives us the ability to work with decoded results that are partially correct. For our system we do not need to extract the correct message sent, as long as it is close enough to our challenge we can gain confidence that it is the correct footage from the camera. The extension of the anomaly detection state machine from Fig. 4 to the continuous case would be to keep a count of the anomaly score across successive challenges. This problem can be modeled with the non-parametric cumulative sum (CUSUM) algorithm to detect changes in our system. The non-parametric CUSUM statistic is updated using the following rules: start with an anomaly score of zero: $S_0 = 0$. For every consecutive visual challenge, compute the anomaly score ϵ_k (the error difference between the reconstructed challenge and the real challenge at step k) and then keep a cumulative count of the errors: $S_k = \max(0, S_{k-1} + \epsilon_k - \delta)$ (δ is a parameter chosen such that the expected value of $\epsilon_k - \delta$ is just smaller than zero when no attack is present).

To calculate ϵ_k we need a metric to tell us how far apart the extracted string is from the correct one. We can measure the distance between the two strings based on the number of edits necessary to convert one string to another, i.e., the minimum number of changes in the extracted string to convert to the correct one. The three possible types of edits are namely insertion, deletion, and substitution, where each of these string transform operations have a cost of 1 [39]. This metric, known as the *edit distance* (sometimes referred to as Levenshtein distance [29]), is widely used for *approximate string matching* in various applications, including the evaluation of the accuracy of an OCR system [39].

3. IMPLEMENTATION

This section describes an implementation of our proposal to evaluate both its performance and design considerations. Fig. 5 shows the setup of one of our laboratory experiments.

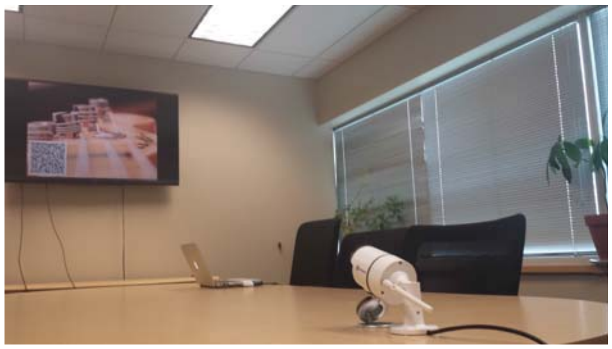


Figure 5: Laboratory setup for experiments. QR code can be seen in the lower left corner of display.

For our prototype we used a popular wireless IP surveillance camera purchased in 2015. The camera uses the Real-Time Streaming Protocol (RTSP) to send its feed to a Network Video Recorder (NVR). After repeated customer support calls we concluded that the vendor only provides applications to see the feed from Windows computers but does not offer support for accessing the raw feed programmatically in Linux systems. In order to access the raw feed we used `binwalk` to analyze the image of the NVR firmware. We found that the NVR device uses the *cramfs* file system, a Linux compressed read-only memory file system often used in embedded devices due to its simplicity and compression techniques to save disk space. As the firmware is not encrypted, we were able to extract the entire cramfs file system and unpacked it. We examined the `/etc/passwd` file and recovered the root password from its hash using popular password cracking tools. This password can then be used to access the system remotely and further retrieve the video feed directly from the camera (we submitted a vulnerability report, and we are expecting a public release from CERT by the end of November 2015). Unfortunately, this vulnerability is one of the ways attackers can gain full control of the camera feed.

3.1 QR Code Detection and OCR Accuracy

We experimented with different tools for decoding the visual challenges. Since our plain text visual challenge is represented as an image, we use an OCR engine to extract the text from the image for verification. There is a large body of research in OCR [3, 25] and it is used in various domains such as text retrieval from natural scenes [46], license plate recognition [2], and real-time translation of text in streets [43, 45]. We select two open-source tools for comparison: `tesseract` [42, 40] and `gocr` [36].

When using OCR engines we cannot rely on a dictionary to correct words because our visual challenges are random strings. But because we know what the correct string should be, then we can compare the recognized text against the expected string. For plain text we use a metric to tell us how far apart the extracted string is from the correct one. This metric, known as the edit distance, is widely used for approximate string matching: to check the difference between two programs, to search text with spelling errors, and to evaluate the accuracy of OCR systems [39]. We use the `python-Levenshtein` [13] library in our implementations.

For QR code challenges, we decode the visual challenge with a barcode reader tool. There are several open-source barcode readers available such as `zbar` [47] and `zxing` [49]. In our experiments, `zbar` outperformed `zxing` and thus `zbar` is used in our results. To check the correctness of `zbar`, we generated and decoded over 86,000 QR codes, and verified that the retrieved text was indeed the correct one. We found the decoding time for QR code challenges to range from an average of 92.39 to 171.36 milliseconds for random strings of length $l = 50$ and $l = 300$, respectively. To verify the text decoded from a QR code with the correct one, we formulate this verification as an *exact string matching* problem because partial errors are not tolerated in decoding QR codes.

Decoding rate of visual challenges: we use different metrics to calculate the *decoding rate* of our visual challenges. For plain text, we use:

$$accuracy_{OCR} = \frac{l - dist}{l} \in [0, 1],$$

where l is the length of the correct string and $dist$ is the edit distance, i.e., the number of errors, between the decoded and correct string. This was the metric used in the Fifth Annual Test of OCR Accuracy [30].

For QR codes, we use:

$$accuracy_{QR} = \begin{cases} 1, & \text{if } decodable \\ 0, & \text{otherwise} \end{cases}$$

where decodable means the barcode reader is able to decode the QR code and return a decoded string in plain text.

We calculate the average decoding rate as the average accuracy of plain text challenges or as the number of successfully decoded QR codes from the camera feed (as shown in Table 1). In addition, we calculate the *error rate* as $error\ rate = 1 - accuracy$.

3.1.1 Image Pre-processing

Instead of applying our decoding algorithms directly to the raw video feed received, we add an image processing step to help us achieve better decoding results. This is particularly true for QR code detection where decoding rates were poor with the raw image.

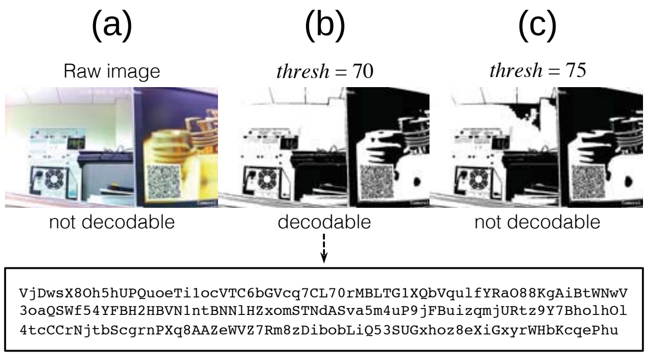
For both visual challenges, we convert the raw image into a binary one. We use the well-known binary *thresholding* operation to convert the raw image into black and white. This operation reassigns all pixels to either black or white values depending on the pixel intensity value $I(i, j)$:

$$O(i, j) = \begin{cases} maxValue, & \text{if } I(i, j) > thresh \\ minValue, & \text{otherwise} \end{cases}$$

We use $maxValue = 1$ (white) and $minValue = 0$ (black). There are several methods for automatic threshold selection [38] including the popular Otsu's method [27], Kapur, Sahoo, and Wong's Maximum Entropy method [18], and Li's Minimum Cross Entropy method [21]. While these methods can help select an optimal threshold, we found that using one method over the other was not enough for improving the recognition of visual challenges in our scenarios. Sometimes a method selects a threshold that will result in a decodable image whereas the same method might select a threshold for a different image that results in an image we cannot decode or accurately recognize its containing visual challenge.

Notice however that we are not restricted to trying only one method, in principle we can produce multiple images and feed them in parallel to the QR code decoder or OCR engine. Therefore we select multiple thresholds (for example, via Otsu's method or Kapur et al's method, and from a set of pre-calculated thresholds known to work well for our purposes) and create at least two binary versions of the raw image. At this step we attempt to decode the raw image and the binary versions in parallel. As we show in our results, it turns out that most of the time at least one of the images will become decodable in the case of QR codes, or will have higher accuracy rate in the case of plain texts. Thus, we are guaranteed with high probability to decode/recognize the visual challenge of the current frame. Once the visual challenge is decoded/recognized, the verifier can further verify it against the known expected challenge.

Fig. 6 illustrates our use of multiple threshold values. The original raw image, shown in Fig. 6(a), was captured during our experimental runs using a visual challenge that encodes a random alpha-numeric string of length $l = 200$. The string is encoded in the QR code shown in the lower right hand-corner of the image. For this particular frame the raw image cannot be directly decoded but becomes decodable after creating a binary representation using a threshold of $thresh = 70$, as shown in Fig. 6(b). It is worth noting that for this particular frame, the same way a threshold of $thresh = 75$ did not produce a decodable binary image (as shown in Fig. 6(c)), neither did the threshold auto selected by Otsu's method or Kapur et al's method help produce a decodable image.



```
VjDwsX8Oh5hUPQuoeTilocVTC6bGVcq7CL70rMBLTG1XQbVqulfYRaO88KgAiBtWNwV
3oaQSWf54YFBH2HBVN1ntBNNlHZxomSTNdASva5m4uP9jFBuizqmjURtz9Y7BholhOl
4tcCCrNjtbScgrnPXq8AAZeWVZ7Rm8zDibobLiQ53SUGxhoz8eXiGxyrWHbKcqePhu
```

Figure 6: A frame captured in our experimental results. Fig. (a) is the raw image and cannot be decoded as is. We convert the raw image to a binary one using different thresholds (Fig. (b) uses $thresh = 70$, Fig. (c) uses $thresh = 75$). We can then successfully decode at least one image, namely Fig. (b).

In Table 1, we show the results of using the raw image frame (either RGB or grayscale image) versus multiple versions of the image (raw image plus binary images with $thresh = 70$ and $thresh = 75$). Our results show two things: (1) It is possible for QR codes to be decoded with a high rate even in the case of no additional pre-processing. For example, the average decoding rate for QR codes is 99% and 94% for challenges of length $l = 100$ and $l = 150$, respectively. This is higher than the accuracy of plain text challenges which have at most 89.23% (for strings with $l = 300$). However, the lowest recognition rate for plain text challenges (83.14%) is higher than the lowest decoding rates for QR codes challenges (70%) without pre-processing. A possible explanation is that the percentage for the plain text reflects grayscaled images. (2) By using multiple thresholds in parallel, we are able to significantly increase the decoding/recognition rate of visual challenges. For instance, QR code challenges had an average increase of 20% for length $l = 50$ and plain text challenges had an average increase of 9.72% for length $l = 100$.

Table 1: The average decoding and accuracy rate of visual challenges are increased by parallel decoding of multiple pre-processed images.

	$l = 50$	$l = 100$	$l = 150$	$l = 200$	$l = 250$	$l = 300$
QR code						
RGB	79%	99%	94%	90%	70%	76%
B/W*	99%	99%	97%	100%	90%**	94%
Plain text						
Gray	86.00%	83.14%	86.13%	84.10%	87.79%	89.23%
B/W*	89.44%	92.86%	88.95%	88.96%	90.93%	93.09%

*parallel image processing (raw image, $thresh = 70$, $thresh = 75$)
**uses different thresholds from the rest ($thresh = 80$, $thresh = 85$)

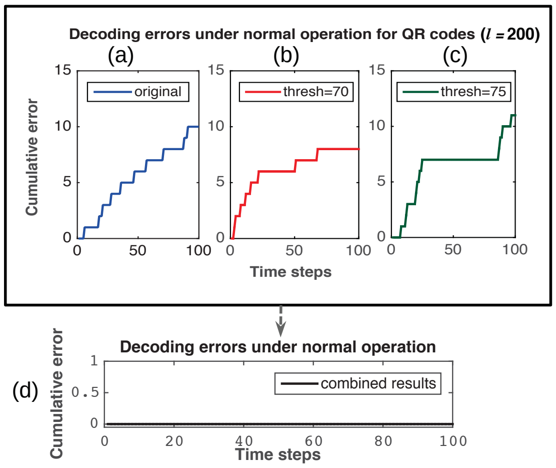


Figure 7: The cumulative error increases by one each time a QR code is not decodable. We decode different versions of the current frame as explained in Fig. 6. The frame is decodable if at least one version is decodable, which is the case shown in Fig. (d) for QR codes of $l = 200$.

Fig. 7 shows a successful case where pre-processing helped minimize the number of errors under normal operation for QR codes (i.e., high decoding rate) of $l = 200$. Fig. 7 (a) shows the cumulative number of decoding errors for raw images captured consecutively, and Figs. 7 (b)-(c) show the cumulative number of decoding errors for binary versions of raw images with $thresh = 70$ and $thresh = 75$. Finally, Fig. 7 (d) shows the cumulative number of detection errors when we are able to decode **at least one** of the different versions. In this case, the number of decoding errors is minimized to zero. We will use this *parallel* processing for selecting at least one decodable image in all our future results. Note that in other cases (e.g., $l = 250$) we still have a few decoding errors even after pre-processing QR code challenges.

3.1.2 Overall Decoding Results

Tesseract vs. GOCR accuracy: We experimented with both `tesseract` and `gocr` for OCR decoding. At the end we decided to focus on `tesseract` because it gave us better accuracy rates. For example, for visual challenges of length $l = 50$, the average accuracy rate for `gocr` is 64.10% whereas for `tesseract` it is 89.44%, and for length $l = 300$, the rate is 59.39% for `gocr` whereas it is 93.09% for `tesseract`. In addition, the performance of `tesseract` was better for recognizing larger strings. To recognize plain text challenges of lengths $l = 50$ and $l = 300$, `tesseract` takes an average of 0.46 to 0.68 milliseconds while `gocr` takes an average of 0.47 to 0.76 milliseconds.

QR vs. OCR accuracy: We evaluate the decoding rate for 100 *sample* visual challenges of 6 different lengths ($l = 50$, 100, 150, ..., $l = 300$). We test two scenarios, (a) decoding rates under ideal conditions (see Fig. 8(a)) where the challenge image is generated synthetically in the computer (to show the maximum performance that could be achieved under ideal image capture settings), and (b) the decoding rates using images captured by the video camera (see Fig. 8(b)). Under ideal settings QR codes achieve decoding rates of 100% for all challenge lengths (that is, all 600 QR codes are successfully decoded) while for plain text, the decoding rate is always above 98% but never 100% with a slight increase at $l \geq 200$. As expected, the rate drops for both visual challenges when using images from a real deployment due to the image errors introduced by the camera and the display. For the QR code, the rate remained high (between 97%-100%) for $l \leq 200$ but dropped for $l \geq 200$. For OCR, we see a drop from approximately 98% to 88% but a clear correlation to Fig. 8 (a). These results suggest that QR codes have better accuracy.

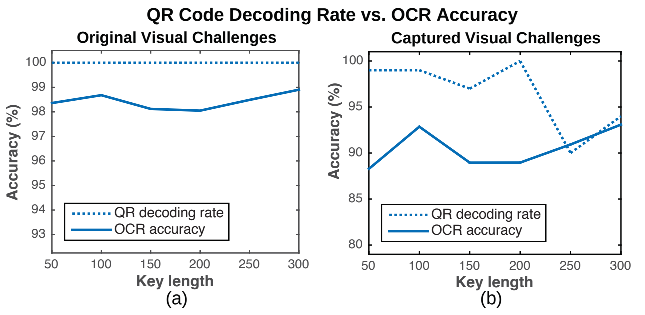


Figure 8: Accuracy of QR code readers and OCR algorithms as the visual challenge length increases. Fig. (a) shows the baseline achieved by analyzing the digital version of challenges sent to the display (i.e., ideal conditions for the image capture), and Fig. (b) shows the performance of visual challenge decoding algorithms after the image is displayed in a monitor and then captured by the camera (i.e., our deployment conditions in Fig. 5). As expected, the accuracy drops on the right image due to imperfect conditions in a real deployment.

Cumulative Errors: Having a QR code decoding error once in a while is not a problem (it is most likely a problem with the image captured), but when the QR code decoding errors are persistent (i.e., when we cannot decode a QR code for two or three consecutive image frames), we need to raise an alarm because this is an indicator of an attack. Therefore, we now study how our alarm state machine (as illustrated in Fig. 4) performs for QR code detection, and how the equivalent CUSUM statistic S_k performs for OCR code detection under normal conditions (i.e., under no attack).

Fig. 9 (a) shows the worst case example (for all other cases the QR code statistic had fewer errors) of the anomaly detection statistic (as introduced in Fig. 4) for QR codes when there is no attack. In all the cases we tested, there was never more than two consecutive QR decoding errors. Therefore we can select the threshold for raising an alarm as having three or four total errors (this ensures that in all our tests there will be no false alarms). Similarly, Fig. 9 (b) shows the worst case example of the non-parametric CUSUM anomaly detection statistic for OCR decoding when there is no attack. By selecting $\delta = 8$ and a threshold of 15 we guarantee that there are no false alarms in all our tests. In the next section we will test how well these thresholds, which do not raise any false alarm, perform under attack.

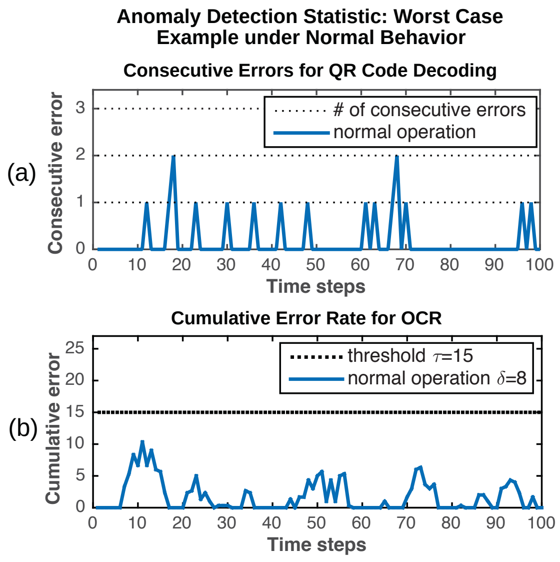


Figure 9: Anomaly detection statistic under normal behavior (no attacks). Fig. (a) shows consecutive QR decoding errors (as introduced in Fig. 4) and Fig. (b) shows CUSUM detection statistic S_k for recognizing plain text challenges.

4. SECURITY ANALYSIS

We study possible attacks and the robustness of our detection mechanism against them. In particular, we consider three different attacks: naive attacks, timing attacks, and forgery attacks. A **naive attacker** is completely oblivious to our detection mechanism and launches replay attacks (Hollywood style). A **timing attacker** knows our anomaly detection algorithms might not raise an alarm with one or two incorrectly decoded frames (because of the thresholds described in the previous paragraph) so it will try to send a false image frame and estimate how long it has before we raise an alarm. A **forgery attack** is the strongest attack against our system. Here, the attacker is both aware of the detection mechanism and has access to the visual challenge. The attacker then attempts to forge a fake image displaying the expected challenge (QR or text). We study each type of attack for both plain text and QR code visual challenges, as we show next.

4.1 Naive Attacks

Attack description: Since the attacker is unaware of the detection mechanism, it might attempt to use old data (i.e., video feed from a previous day) in a replay attack. In this case, our mechanism will immediately detect the attack as the old data will not contain the correct visual challenge. Consider a naive attacker that has access to old data (e.g., image frames from time $t = 1$ until $t = 30$). At a particular time t_i, the verifier sends v_i as a challenge to the monitor and expects to see something significantly similar to v_i in the next frame. During the attack, the attacker launches a replay attack at $t = 70$ and uses old image frames (frames starting at $t = 1$). The verifier processes and decodes the retrieved image frame. Then, it verifies whether the extracted text from the retrieved visual challenge is correct.

Attack against OCR: when the attacker launches an attack against OCR, the compromised frame at $t = 70$ will contain the visual challenge of v_1. Under normal operation the accuracy rate of decoding the visual challenge might have been approximately 89% and with an error rate of approximately 11%. When the attack is launched the accuracy rate drastically drops to 3.67% (and the error increases to 96.03%). Fig. 10 (a) shows the error rate when the attacker replays old frames starting at $t = 70$. As shown, the error rate immediately goes up at $t = 70$ and stays high for the duration of the attack. Fig. 10 (b) shows that as soon as the attack begins, the anomaly detection score grows quickly, exceeding the normal behavior of the system. An alarm will immediately be triggered.

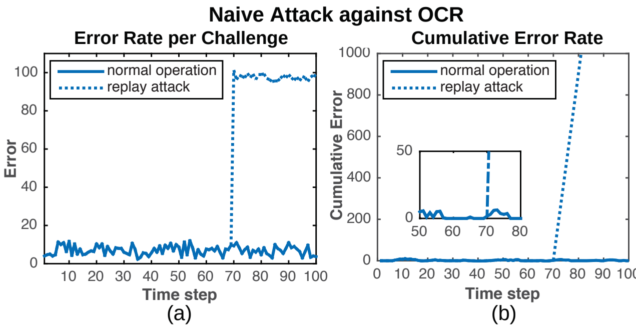


Figure 10: Naive Attack on OCR decoding of plain text challenges. Left: anomaly score per image. Right: cumulative anomaly score over time (i.e., the non-parametric CUSUM statistic).

Attack against QR code: QR codes are very robust for detecting replay attacks. In our experiments as soon as a replay attack is launched, a QR code with an incorrect challenge is decoded and as illustrated in Fig. 4, it will immediately raise an alarm. There is no need to show the anomaly statistic as in the previous case.

As we show, our proposed detection mechanism is secure to detecting this type of attack. Even when the footage might not look suspicious to security guards or automated analysis tools, our mechanism will detect whether the camera is not showing the correct footage.

4.2 Timing Attacks

Attack description: In this attack we try to see if an attacker can send a couple of incorrect frames back to the verifier without raising an alarm and then study how long the attacker can remain undetected. In particular the goal of this attacker is to launch attacks that block the QR code (e.g., blurring or blocking the place where the QR code would appear) while trying to stay undetectable by our detection mechanism (i.e., we assume the attacker knows the anomaly detection tests used by our proposal). We show this attack for both plain text (Fig. 11) and QR codes (Fig. 12).

Attack against OCR: although the goal of the attacker is to send bad frames while staying undetectable, we show that this attack is not possible for bypassing plain text challenges; that is, the attacker cannot send a single incorrect

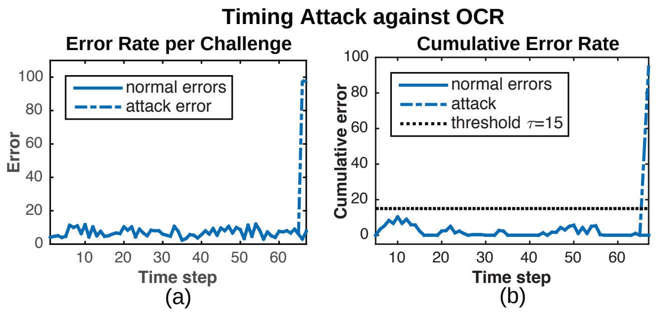


Figure 11: Timing Attack on OCR decoding of plain text challenges. The attacker cannot send a single incorrect frame without being detected. Left: anomaly score per image. Right: cumulative anomaly score over time.

frame without being detected! If the attacker launches its attack at $t = 70$ such that the visual challenge v_{70} in the compromised frame at $t = 70$ is an old visual challenge v_1, then just like in the naive attack case, the error rate will immediately raise above 85% (as shown in Fig. 11 (a)) and the anomaly detection score will go beyond the threshold (as shown in Fig. 11 (b)). The reason for this is that the error rate under normal operations is very low (e.g., about 10%) when compared to the error rate of approximating a bad visual challenge with the correct one (e.g., about 85%). It is worth noting that the threshold we selected ($\tau = 15$) does not raise any false alarm under normal operations and in addition, the replay attack is not able to stay undetected. So here the attacker cannot send any bad frame because it will be detected immediately. Even if the attacker tries to obfuscate or cover the place where the challenge would be, the OCR accuracy would be close to 0% and this would immediately raise an alarm.

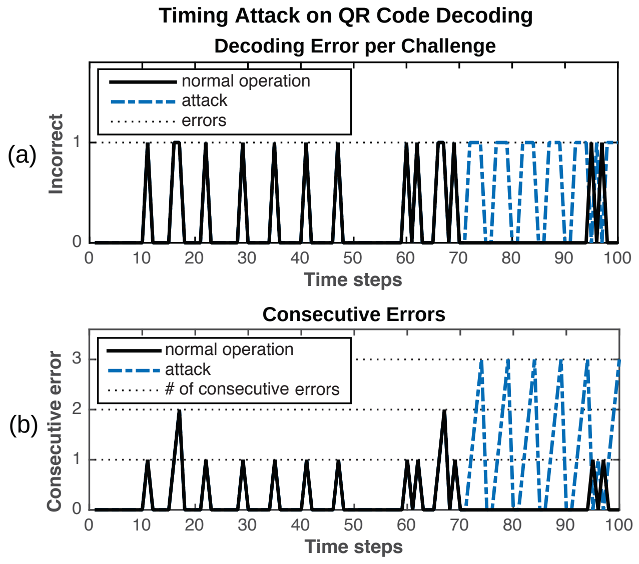


Figure 12: Timing Attack on QR codes. The attacker introduces errors at $t = 70$ for short periods of time to remain undetected. Top: decoding errors per image. Bottom: consecutive errors.

Attack against QR code: In contrast to OCR, here the attacker might try to leverage the fact that QR codes are sometimes not decodable, thus, it might purposefully replay old frames containing undecodable QR codes. The attack is again launched at $t = 70$ and replays frames that the attacker knows were not decodable (or simply obfuscates the QR code). Because the attacker would like to remain undetected, it will limit the attack duration to the maximum number of old frames (with undecodable QR codes) that do not raise the anomaly detection score beyond the alarm threshold. Fig. 12 (a) shows the errors introduced when the attacker replays old frames with an undecodable QR code starting at $t = 70$ for short periods of time. To maximize the attack time, the attacker stops the attack before hitting the threshold and waits until the cumulative error has come down to zero as shown in Fig. 12 (b). For example, the attacker launches a replay attack (with only undecodable QR codes) at $t = 70$, $t = 71$, and $t = 72$. It knows that a 4th consecutive error will raise an alarm, so it waits until the cumulative error goes down to zero. This happens after 3 consecutive correct challenges. Then the attacker launches another sequence of replay attacks. Since the goal of the attacker is to stay undetectable and maximize the attack duration, it waits until the anomaly detection becomes zero before launching another sequence of attacks.

Timing analysis: we showed that for QR codes, the attacker might be able to stay undetectable when its attack duration is no larger than 3-time steps. What this means is that for a total of 3-time steps the camera feed will report incorrect frames and not raise an alarm. The ability of the attacker to remain undetected in real-time will depend on how frequently we send new challenges to the display. In our particular implementation we send new visual challenges every 3 seconds. Because the time of decoding is small compared to that (approximately 171.36 milliseconds and could be as low as 92.39 milliseconds for QR codes encoding shorter strings) the attacker would get approximately 9 seconds (3 incorrect frames plus 3 seconds of challenge refresh rate) to perform a malicious act in the field of vision of the camera while sending undecodable frames back to the verifier. We can certainly improve on that by e.g., sending new challenges more frequently (1 per second) or selecting an alarm threshold of 3 (which will leave the attacker with the option of sending only two incorrect frames) that will give the attacker only 2 seconds of malicious activity. Our analysis is primarily a framework to evaluate the refresh rate of the QR code and the alarm threshold. In practice, the amount of time that an attacker might need to do something nefarious in the field of vision of the camera without being detected is highly context sensitive and will depend on the specific deployment.

4.3 Forgery Attacks

Attack description: In this case the attacker we consider will superpose the valid visual challenge on top of an old image. In particular, we assume that the attacker is both aware of the detection mechanism and has access to the visual challenge. The goal of this attacker is to fool our detection mechanism to give operators the confidence that the feed is legitimate (because it contains the correct visual challenge). Whereas, in reality the attacker has modified the feed (before it leaves the camera) to show old frames but superposing the correct visual challenge (a copy and paste attack). The attacker might use two different methods to do this: either the attacker has access to the communication channel between the verifier and the display and can recreate the valid visual challenge, or the attacker can 'cut' the visual challenge from the current frame (before it leaves the camera) and paste it to an old frame.

To deal with this attacker we can leverage image and video forensics tools [8, 17, 23, 48]. For example, video forensics can help us detect non-sequential frames, e.g., inconsistency in the lighting of sequential frames, discrepancy between objects' locations, or non-uniformity between resolution of the images. We used the Error Level Analysis algorithm (ELA) [20] to detect forgeries in frames against the attack that pastes the original visual challenge on top of a bad frame. This algorithm looks at the amount of error introduced when the image is re-saved at different quality levels and reveals the areas that contain different levels (manipulated areas on the image).

Fig. 13 shows a proof-of-concept for detecting this type of attack. We used the `fotoforensics` tool [19] which is based on error level analysis to detect a forged image. The image in Fig. 13 (a) is legitimate whereas the image in Fig. 13 (b) is forged; we pasted the real visual challenge image on top of an old frame. The images at the bottom show the error levels of the legitimate and forged images, respectively. It is clear that the only area with a different color tone in the image shown in Fig. 13 (d) is the area where the QR code was pasted. In the future, we plan to look more in depth at media forensics tools to deploy an automatic test that detects attempted forgeries.

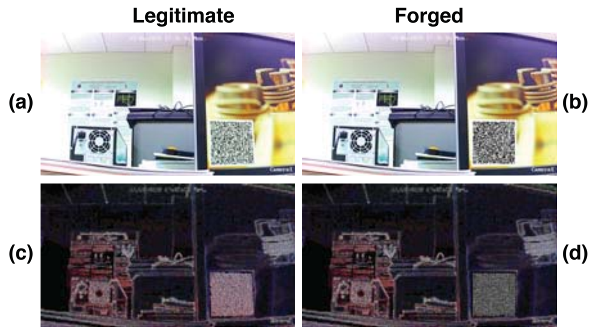


Figure 13: Forgery Attack on a QR code challenge. Top left: legitimate image. Top right: forged image where the original image of a QR code is superposed on an old image frame. Bottom images: error levels for legitimate and forged images.

Video and image forensics is still an early discipline. It is developing a cat-and-mouse game where a forensic tool used to detect image manipulation (such as cut and paste attacks) can be bypassed by *anti-forensic* tools. One advantage we have in our case to detect online attacks is that all anti-forensic tools are computationally intensive, and it is not clear if these tools will be able to finish a forged image that can bypass our forgery-detection algorithms in time to reply to our challenges (at the moment we wait one second before sending a challenge and checking the display, but we can certainly reduce that timing, which in turn will make undetected forgery attacks harder to launch).

5. RELATED WORK

Security of camera networks: A discussion on security issues for smart camera systems can be found in [37]. Overall, the main requirements for securing camera systems include the confidentiality, integrity, and freshness of data, as well as tamper-resistance and resilience against physical attacks. There has been attention on ensuring such systems secure camera streams in the cloud [16], to securing streams in transmission (by signing images before storage [9] or by encrypting the stream before sending off to a remote storage [16]). Watermark techniques have been proposed to ensure the integrity and freshness of digital video content [24]. However, these approaches assume that secret keys have not been compromised.

Research on the security of sensors includes those that look at the development of *trusted sensors* [34, 10, 7], and camera networks in particular [44], to ensure trustworthy sensor readings. The proposed approaches make it difficult for a sensor to lie about its readings or to maliciously fabricate the readings (e.g., modifying captured photos, falsifying photo location) because they assume the presence of trusted computing technology [4, 28]. The recent work on virtual-proofs [32] looks at the problem of proving physical claims (i.e., the temperature of a sensor) without using classical secret keys or tamper-resistant hardware, however, they require the active involvement of the prover during the verification process whereas our approach does not.

Our goal is not to replace such technologies but to complement them by providing another defense mechanism completely independent from the specific hardware or software available in the camera.

Using Cameras to Improve Security: Our work is also related to the literature that uses *visual side channels* involving camera phones and barcode scanning to exchange device information such as public keys [22, 35]. The visual channel used to exchange information needs to be trusted, and this trustworthiness rests on the premise that humans are present (i.e., they can see with their eyes the same image the camera is capturing) and will be able to spot any spoofs on the "visual channel." Our premise focuses precisely on the opposite assumption: we assume no human is present to verify the integrity of a video feed, and furthermore, our threat model assumes the attacker is able to compromise the visual channel to send false image frames and launch replay attacks so that the footage might not look suspicious even to security guards. Also, because we do not send the challenge to the camera, the camera does not need to know about the attestation protocol nor stop its normal operations to perform any attestation code (as in the case of McCune et al. [22]). Our proposal can be used to improve security protocols that rely on a trusted visual channel to be secure, but our proposal is also more general and applicable to other cases, as stated in the introduction.

Roesner et al. [31] also study the interaction of the physical world with cameras. In particular, they study displaying privacy policies in QR codes that can be automatically detected by continuous sensing devices (e.g., Google Glass). Our proposal can again improve this previous work by sending continuous changing visual challenges to guarantee the video being captured by these sensing devices is accurate.

6. CONCLUSIONS AND FUTURE WORK

We have proposed a new way to verify the integrity and freshness of footage from security cameras by sending visual challenges to the area being monitored by the camera. Our approach can provide an additional layer of security to traditional crypto authentication and message integrity codes.

There are many open research challenges and directions in which to extend our ideas. The most immediate direction is to study in-depth the forensics and anti-forensics tools to prevent forgery attacks. The field of video forensics and video anti-forensics is still in its early stages and we have the opportunity to contribute a new use-case for future research.

Another extension depends on whether or not the camera is deployed in a private or public space. If it is a private space, like the control room of a nuclear reactor, then perhaps we can have a dedicated display showing QR codes or random plain text strings. But, the same deployment might not be well suited for public spaces where QR codes or random strings would be seen with suspicion by the public.

There is some work on high quality visual QR codes which superimpose images and can be aesthetically more pleasing [5]. We can also have QR codes that decode to a saying, a word definition, or a phrase (instead of a random string) so if a person scans the code, they receive something intelligible.

Another extension is to consider implementing our approach to other sensors. Our research introduces a new kind of attestation tailored specifically for sensing devices. Here, the verifier does not send the *challenge* to the device itself. Instead, the challenge is implicitly sent to the entity that the device is sensing which can be sound sensors, or temperature sensors, among others.

Acknowledgments

We thank Sandy Clark and Jonathan McCune for their feedback and helpful suggestions on related work. This work was supported in part by NIST under award 70NANB14H236 from the U.S. Department of Commerce, and The Air Force Office of Scientific Research under award FA-9550-12-1-0077.

7. REFERENCES

[1] ISO/IEC 18004:2015, Information technology—Automatic identification and data capture techniques —QR Code bar code symbology specification.

[2] J. Barroso, E. Dagless, A. Rafael, and J. Bulas-Cruz. Number plate reading using computer vision. In *IEEE International Symposium on Industrial Electronics (ISIE)*, volume 3, pages 761–766, 1997.

[3] T. M. Breuel. The OCRopus Open Source OCR System. In *Proceedings of SPIE, the International Society for Optical Engineering*, pages 68150F–1, 2008.

[4] D. Challener, K. Yoder, R. Catherman, D. Safford, and L. Van Doorn. *A practical guide to trusted computing*. IBM Press, 2008.

[5] H. K. Chu, C. S. Chang, R. R. Lee, and N. J. Mitra. Halftone QR codes. *ACM Transactions on Graphics*, 32(6):217:1–217:8, 2013.

[6] Denso Wave. QR codes. http://www.qrcode.com/en/.

[7] A. Dua, N. Bulusu, W. Feng, and W. Hu. Towards trustworthy participatory sensing. In *Proceedings of the 4th USENIX conference on Hot topics in security (HotSec)*, 2009.

[8] J. Fridrich. Digital image forensics. *IEEE Signal Processing Magazine*, 26(2):26–37, 2009.

[9] G. Friedman. The trustworthy digital camera: restoring credibility to the photographic image. *IEEE Trans. Consum. Electron*, 39(4):905–910, 1993.

[10] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall. Toward trustworthy mobile sensing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications (HotMobile)*, pages 31–36, 2010.

[11] D. Goodin. A Fort Knox for Web crypto keys: Inside Symantec's SSL certificate vault. http://arstechnica.com/security/2012/11/inside-symantecs-ssl-certificate-vault, Nov. 2012.

[12] D. Goodin. Prosecutors suspect man hacked lottery computers to score winning ticket. http://arstechnica.com/tech-policy/2015/04/

prosecutors-suspect-man-hacked-lottery-computers-to-score-winning-ticket, Apr. 2015.

[13] A. Haapala. Levenshtein Python C. https://github.com/ztane/python-Levenshtein/.

[14] M. Hara, M. Watabe, T. Nojiri, T. Nagaya, and Y. Uchiyama. Optically readable two-dimensional code and method and apparatus using the same, 1998. US Patent 5,726,435.

[15] C. Heffner. Exploiting Network Surveillance Cameras Like a Hollywood Hacker. https://youtu.be/B8DjTcANBx0, Nov. 2013.

[16] R. Hummen, M. Henze, D. Catrein, and K. Wehrle. A cloud design for user-controlled storage and processing of sensor data. In *CloudCom*, pages 232–240, 2012.

[17] M. K. Johnson and H. Farid. Exposing digital forgeries by detecting inconsistencies in lighting. In *Proceedings of the 7th Workshop on Multimedia and security (MM&Sec)*, pages 1–10, 2005.

[18] J. N. Kapur, P. K. Sahoo, and A. K. Wong. A new method for gray-level picture thresholding using the entropy of the histogram. *Computer vision, graphics, and image processing*, 29(3):273–285, 1985.

[19] N. Krawetz. FotoForensics. http://fotoforensics.com/.

[20] N. Krawetz. A pictures worth digital image analysis and forensics. *Black Hat Briefings*, pages 1–31, 2007.

[21] C. Li and C. Lee. Minimum cross entropy thresholding. *Pattern Recognition*, 26:617–625, 1993.

[22] J. McCune, A. Perrig, and M. Reiter. Seeing-is-believing: using camera phones for human-verifiable authentication. In *IEEE Symposium on Security and Privacy (S&P)*, pages 110–124, 2005.

[23] S. Milani, M. Fontani, P. Bestagini, M. Barni, A. Piva, M. Tagliasacchi, and S. Tubaro. An overview on video forensics. *APSIPA Transactions on Signal and Information Processing*, Vol. 1, 2012.

[24] S. P. Mohanty. A secure digital camera architecture for integrated real-time digital rights management. *Journal of Syst. Architecture*, 55(10-12):468–480, 2009.

[25] S. Mori, H. Nishida, and H. Yamada. *Optical character recognition*. John Wiley & Sons, Inc., New York, 1999.

[26] M. Murphy. Man arrested for tampering with traffic cameras, posting 'how-to' videos on Facebook. http://pix11.com/2015/08/25/man-arrested-after-posting-videos-on-facebook-of-him-tampering-with-traffic-cameras, Aug. 2015.

[27] N. Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[28] S. Papa and W. Casper. Trusted computing. In *Encyclopedia of Cryptography and Security*, pages 1328–1331. Springer US, 2011.

[29] V. Pieterse and P. E. Black. Levenshtein distance. In *Dictionary of Algorithms & Data Structures*, 2015.

[30] S. V. Rice, F. R. Jenkins, and T. A. Nartker. *The fifth annual test of OCR accuracy*. Information Science Research Institute, 1996.

[31] F. Roesner, D. Molnar, A. Moshchuk, T. Kohno, and H. Wang. World-driven access control for continuous sensing. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, pages 1169–1181, 2014.

[32] U. Ruhrmair, J. Martinez-Hurtado, X. Xu, C. Kraeh, C. Hilgers, D. Kononchuk, J. Finley, and W. Burleson. Virtual proofs of reality and their physical implementation. In *IEEE Symposium on Security and Privacy (S&P)*, pages 70–85, 2015.

[33] Santa Clara County Sheriff. PG&E substation surveillance video. https://www.youtube.com/watch?v=RQzAbKdLfW8, Jun. 2013.

[34] S. Saroiu and A. Wolman. I am a sensor, and I approve this message. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications (HotMobile)*, pages 37–42, 2010.

[35] N. Saxena, J. Ekberg, K. Kostiainen, and N. Asokan. Secure device pairing based on a visual channel. In *IEEE Symposium on Security and Privacy (S&P)*, 2006.

[36] J. Schulenburg. GOCR, an open-source character recognition. http://jocr.sourceforge.net/.

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

[38] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, 13(1):146–165, 2004.

[39] S. S. Skiena. *The algorithm design manual*, volume 1. Springer Science & Business Media, 1998.

[40] R. Smith. An overview of the tesseract ocr engine. *International Conference on Document Analysis and Recognition (ICDAR)*, 2:629–633, 2007.

[41] A. Tabatabai. How much monitoring of Iranian nuclear facilities is enough? http://thebulletin.org/how-much-monitoring-iranian-nuclear-facilities-enough7923, Jan. 2015.

[42] Tesseract OCR. Tesseract Open Source OCR Engine. https://github.com/tesseract-ocr/tesseract.

[43] B. Turovsky. Hallo, hola, olá to the new, more powerful Google Translate app. http://googleblog.blogspot.com/2015/01/hallo-hola-ola-more-powerful-translate.html, Jan. 2015.

[44] T. Winkler and B. Rinner. Securing embedded smart cameras with trusted computing. *EURASIP Journal on Wireless Communications & Netw.*, 2011:8, 2011.

[45] J. Yang, J. Gao, Y. Zhang, and A. Waibel. Towards automatic sign translation. In *Proceedings of the First International Conference on Human Language Technology Research (HLT)*, pages 1–6, 2001.

[46] C. Yi and Y. Tian. Text string detection from natural scenes by structure-based partition and grouping. *IEEE Transactions on Image Processing*, 20(9):2594–2605, 2011.

[47] ZBar. ZBar bar code reader. http://zbar.sourceforge.net/.

[48] J. Zhang, Y. Su, and M. Zhang. Exposing digital video forgery by ghost shadow artifact. In *Proceedings of the ACM Workshop on Multimedia in Forensics (MiFor)*, pages 49–54, 2009.

[49] ZXing. ZXing barcode image processing library. https://github.com/zxing/zxing/.