2016 IEEE Symposium on Security and Privacy

PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations

Assa Naveh* and Eran Tromer†
Blavatnik School of Computer Science, Tel Aviv University
Tel Aviv, Israel
Email: *assa.naveh@cs.tau.ac.il, †tromer@cs.tau.ac.il

***Abstract*—Since the invention of the camera, photos have been used to document reality and to supply proof of events. Yet today it is easy to fabricate realistic images depicting events that never happened. Thus, dozens of papers strive to develop methods for authenticating images. While some commercial cameras already attach digital signatures to photographs, the images often undergo subsequent transformations (cropping, rotation, compression, and so forth), which do not detract from their authenticity, but do change the image data and thus invalidate the signature. Existing methods address this by signing derived image properties that are invariant to some set of transformations. However, these are limited in the supported transformations, and often offer weak security guarantees.**

We present `PhotoProof`, a novel approach to image authentication based on cryptographic proofs. It can be configured, according to application requirements, to allow *any* permissible set of (efficiently computable) transformations. Starting with a signed image, our scheme attaches, to each legitimately derived image, a succinct proof of computational integrity attesting that the transformation was permissible. Anyone can verify these proofs, and generate updated proofs when applying further permissible transformations. Moreover, the proofs are zero-knowledge so that, for example, an authenticated cropped image reveals nothing about the cropped-out regions.

`PhotoProof` is based on Proof-Carrying Data (PCD), a cryptographic primitive for secure execution of distributed computations. We describe the new construction, prove its security, and demonstrate a working prototype supporting a variety of permissible transformations.

I. Introduction

A. Background

Photography is one of the most prevalent media in the modern age, and digital cameras are nowadays ubiquitous and integrated into mobile phones and portable computers. Photos are usually considered reliable and convincing, and are relied upon in personal, commercial and legal contexts, and in forming public opinion.

Digital image editing tools are also very common, and with the ability to improve image quality and add artistic flavor, they also help creating fake photos of scenes that are essentially altered, or wholly fictional, yet appear realistic. Many such forgery examples are well known, in propaganda [19], [41], photojournalism [26], [54], pranks, extortion attempts, attempts at personal embarrassment, and falsified legal evidence. Tools are needed to detect fake images and help the photographic medium maintain its credibility.

Image Authentication (IA) is, loosely speaking, the ability to prove that an image faithfully represents some original photograph that was captured in a given class of physical image acquisition devices (e.g., a camera model). Distinguishing a genuine image from a fake just by inspection can be very hard.[1] Forensic experts can seek anomalies in content, such as shadow/illumination direction [29], in file metadata, e.g. thumbnails embedded in the image file header [30], or in digital artifacts (see [16] for a survey), but in general this can be time-consuming and unreliable.[2] An alternative approach, pursued in this paper, is to associate additional data (a signature *or* proof) with the final image, in order to detect forgery reliably and robustly.

B. Prior work

A popular approach to image authentication is to use an authentication mechanism to append proofs to authentic images. These proofs can readily be verified by any viewer. One example is for the camera to digitally sign the image when it is captured, as suggested by Friedman [21]. A secret signing key can be securely embedded inside the camera's Image Signal Processor (ISP). Using the corresponding public key, viewers can verify the signature and thus be convinced that the image is authentic.

The limitation of this solution is that digital signatures are, by design, sensitive to even the smallest change in the signed data. When the signature is calculated on the image (or, as is often done for efficiency, a collision-resistant hash thereof), changing even a single bit of the image will result in a mismatching signature. This restriction is incompatible with many applications where it is legitimate and desirable to alter images, as long as they are kept "authentic" (in some application-specific sense). For example, operations such as rotation, rescaling, lossy compression, brightness/contrast adjustments and cropping may be considered permissible, as the resulting image still faithfully represents a captured physical scene.[3]

[1] For example, Schetinger et al. [44] found that nonprofessional human inspectors identified forgeries in digital images with only about 58% accuracy.

[2] A recent Broad Agency Announcement (BAA) soliciting research proposals in the area of visual media forensics by the American agency DARPA [14] points out the imbalance between thousands of available image manipulation programs and very few forgery detection tools in the market, and the relatively low work capacity of human analysts.

[3] For example, New York Times guidelines on integrity permit rectangular cropping [53].

© 2016, Assa Naveh. Under license to IEEE.
DOI 10.1109/SP.2016.23

IEEE computer society

Developing an image authentication mechanism that supports some set of permissible transformations is an ongoing research area. Generally speaking, the existing solutions can be categorized into two main approaches.

Semi-fragile watermarking. The first category is *semi-fragile watermarking* (e.g., [32], [52], [27]). A watermark is a signal or pattern embedded into an image in a perceptually and statistically invisible fashion. Embedding the watermark ensures that performing one of the allowed transformations on the image will not destroy the watermark, but (ideally) any other digital manipulation will. Sun et al. [52] suggested embedding a semi-fragile watermark in coefficients of SVD decomposition of image blocks in a way resilient to JPEG compression. In [32], two JPEG compression invariants are used to embed an authentication string in the image DCT coefficients, making the watermark robust to JPEG compression.

Robust hashing. The second category of image authentication mechanisms is *robust hashing* or *feature extraction* (e.g., [46], [20], [55], [33], [48], [17], [59], [34], [60]). Here, a specially designed hash function for images is defined, such that different images yield different results, but images that are essentially the same (i.e., modified by some permissible transformation) give identical (or close) hash digests. When an image is captured, its digest is signed using a private signing key and attached to the image. A verifier that receives a copy of the image computes its digest, verifies the signature on the attached digest using a verification key, and compares the two digests. If these two digests are close enough, by some distance measure, they are accepted by the verifier.

Venkatesan et al. [55] construct an image hash function robust to JPEG compression and limited geometric modifications. Their authentication method relies on a secret key and is thus restricted to the private verification setting. Lin and Chang [33] use differences between corresponding DCT coefficients from different blocks to create authentication data which is invariant to JPEG compression. Lin et al. [34] present a robust hash based on running a pseudorandom feature vector of the image through a Slepian-Wolf coding (where the pseudorandom seed is known by the verifier). They show this technique to be resilient to JPEG compression, affine warping, and contrast and brightness adjustments.

All existing authentication methods have at least one of the following drawbacks:

Fixed set of permissible transformations. Different applications may consider different transformations as permissible, but most existing techniques are specific to a fixed set of supported transformations (e.g., [46], [20], [33], [48], [40], [34]). Usually the set is also relatively small and includes transformations of the same "nature", e.g., only certain geometric transformations or only image compression, which are preserved by an invariant of their watermarking or underlying hash. One exception is the method in [17], which allows some degree of freedom in the choice of allowed transformations, though at the expense of accuracy.

Non-negligible error probability. Most existing image authentication techniques with permissible transformations have non-negligible probabilities for false alarms or false acceptance. This is mainly due to the statistical nature of verification, usually in the form of comparison of some quantified property to an (empirically chosen) threshold (as in [46], [33], [17], [59], [34]).

Vulnerability to adversaries. Many of the methods are insecure against an adaptive attacker who is familiar with the authentication method. Such an attacker may devise an image that will fool the verifier with very high probability. In [46] for example, the authors propose robust hash image authentication that works by splitting an image into blocks (of different sizes) and taking their intensity histograms. An attacker who is aware of this method can generate an image that (a) gives the same vector of authentication data (i.e., block histograms) and (b) is not a result of a JPEG compression. To overcome this weakness, some have suggested incorporating pseudo-random elements into the authentication process. However, in the public verification setup, it is often the case that the random seed (i.e., the key) must be known. The attacker might then compose a manipulated image that fools the verifier with high probability. The method proposed in [34], for example, is vulnerable in this way, and other methods are similarly vulnerable.

Lack of succinctness on the verifier side. In most authentication methods, the verification time and the size of the authentication data grow with the image size or with the number of transformations that were applied on it. While for robust hash methods this results in larger image data, in watermarking based authentication, longer embedded data results in a larger decrease in image quality.

CertiPics. A different approach to image authentication is taken by Walsh [57], who implemented CertiPics, an image authentication software over the Nexus operating system [45], [49] using a secure co-processor (TPM). First, a policy is defined, stating the allowed transformations and editing rules. Users use specially written and authorized applications on a Nexus machine for editing. CertiPics keeps a certified and unforgeable log of the transformations performed. This log is then used when viewing the edited image on a Nexus machine, to verify that the image is authentic according to the policy.

CertiPics, too, lacks succinctness on the verifier side: the size of the log and the effort to verify it grow with the length of the history. It also does not provide zero-knowledge: the log leaks the editing history of the image, even when this is undesirable and not required by the policy.

Trusting cameras. Most approaches to image authentication, including ours, rely on a signing camera as root of trust. Critique and justification of this assumption is not the main focus of our contribution, but since it affects the overall security of the system, we discuss it in Appendix A.

Table I
COMPARISON BETWEEN EXISTING AUTHENTICATION METHODS AND PhotoProof

technique	type	claimed transformations							flexible specification	negl. error probability	size overhead
		JPEG	rotate	crop	scale	brightness, contrast	flip	trans-pose			
digital signature									×	✓	$O(1)$
[32]	SW	✓				✓			×	×	$O(n)$
[33]	RH	✓							×	×	$O(n)$
[52]	SW	✓							×	×	$O(n)$
[17]	RH	✓			✓				restricted	×	unspecified
[34]	RH	✓	✓		✓	✓			×	×	$O(n)$
[55]	RH	✓	$< 2^\circ$	$< 10\%$	$< 10\%$				×	×	$O(n)$
[60]	RH	✓	$< 5^\circ$	$< 2\%$	✓				×	×	$O(1)$
[20]	RH	✓				✓			×	×	$O(1)$
PhotoProof	CP	(✓)	✓	✓	(✓)	✓	✓	✓	any efficient transformation	✓	$O(1)$

Type can be Robust Hash (RH), Semi-fragile Watermarking (SW), or Cryptographic Proofs (CP). **Flexible specification** denotes whether the set of permissible transformations is configurable. **Error probability** denotes whether the probability of false alarms and misdetections is negligible. **Size overhead** is the length of the authentication data as a function of image size. (✓) refers to capabilities supported in our scheme but not yet implemented in our current prototype.

C. Our contribution

We present PhotoProof, a novel approach to image authentication that is based on cryptographic proofs of computational integrity. Our construction yields a public-proving and public-verifying authentication system which is secure and reliable. We also describe our implementation of a working proof-of-concept prototype, including a collection of permissible image transformations.

Our construction realizes IA scheme, a new cryptographic primitive defined in this work, which does not possess any of the aforementioned shortcomings of the existing solutions and has other desirable properties. Table I summarizes the comparison of our construction to existing techniques.

Consider the following IA workflow. A system administrator first decides on a set of permissible transformations. Any editor can apply a permissible transformation on an image and generate a new proof, provided the image's authentication data is available either as a digital signature computed by the camera or as a previous proof. Any viewer can then verify that the proof is valid for this image.

PhotoProof fulfills the above IA workflow, with additional important properties, some of which have not been achieved by any other technique:

1) Proofs are unforgeable. Not only does our method have negligible error probability, either for falsely rejecting or falsely accepting an image as authentic, it is also provably secure against (computationally bounded) adversaries that might try to pass manipulated images as authentic.
2) Proofs are zero-knowledge, which ensures that no information about the image, other then it being authentic, can be learned from the proof. For example, one might want to crop out or black out an embarrassing portion of an image. The zero-knowledge property guarantees that information edited out of the image (in an authenticity-preserving way), and even the choice of transformations that were applied to the image, remain secret.
3) Verification is fast and proofs are of constant-size.
4) Authentication can include additional metadata, e.g, to prevent change of author information or geographic data.
5) Finally, PhotoProof is unique in that it can naturally be adjusted to include any set of image transformations.

In terms of performance, proof verification takes mere milliseconds, but at present the creation of proofs is too slow for many applications and common image sizes. We discuss possible performance enhancements in Section V.

D. Approach

Our approach is an application of the Proof-Carrying Data paradigm, defined by Chiesa and Tromer [11].

Consider the (informal) scenario of a distributed computation between multiple parties, with parties receiving inputs from others, performing their own computation and outputting their results to the next parties in the computation. PCD transforms this computation into a secure one, by enabling each party to attach to its output a proof. The PCD proof certifies not only the correctness of the last computation, but also of its entire history. This means that in order to accept the result of such a distributed computation as correct, only the last proof is required (see Figure 1).

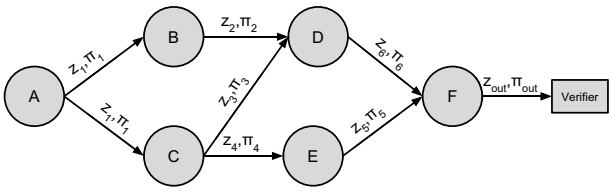

Figure 1. A PCD computation transcript. Each node receives input messages z_i and proofs π_i from previous nodes, and then computes its output and generates a proof for it. Only the final proof is needed to certify that the entire computation was executed correctly.

PCD is known to be possible under various reasonable assumptions [9], and was recently implemented, in the general

case [7], and (more efficiently) in the special case of a single hop, where it is called a SNARK [6]. We build on these prior works and adapt them to the image authentication setting. This work is also the first application-specific use of PCD (prior prototypes were either proofs-of-concept for generic computation [7], [13] or used the special case of SNARKs [3]).

We begin with the following high-level description of how to use PCD to build an image authentication mechanism (see Figure 2). Initially, an image is captured with a digital camera. A user who wishes to edit the image can apply a permissible transformation on it, and — using the PCD proving algorithm — generate a new proof for its authenticity. Any number of such steps can follow, and may be performed by different users. At any time, any viewer of the image may check its proof by using the PCD verification algorithm. The proof guarantees that this latest step *and all prior steps* were complied with the specification (encoded via PCD) of the predefined set of permissible transformations. We propose a digital signature produced inside the camera (as also proposed in [21]) as the root of trust that specifies the beginning of the chain of permissible transformation steps, and with the required precautions we discuss in Appendix A.

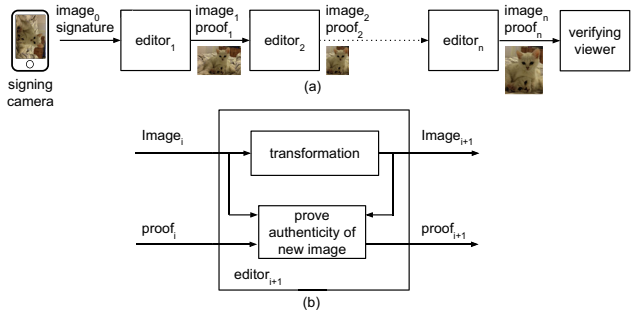

Figure 2. The history of an image as a distributed computation (a) where in each step a PCD-generated proof is appended (b).

In this work we present the theoretical construction of `PhotoProof` from PCD for any set of permissible transformations, as well as an implementation of a `PhotoProof` prototype, supporting small images and a diverse set of permissible transformations. Our prototype builds on the PCD implementation of [7].

The rest of this paper is organized as follows. We begin by formally defining image authentication schemes in Section II. In Section III we give some required background on PCD. We present our theoretical construction and its proof of security in Section IV and describe our prototype, challenges in instantiation, and possible extensions in Section V. We summarize and give future research directions in Section VI.

II. Defining Image Authentication

We consider the following image authentication workflow. First, the IA scheme is constructed and implemented, along with a library of implemented image transformations. A *system administrator* who wishes to integrate this IA scheme into his or her system, chooses a set of permissible transformations from the available collection. This set reflects what changes are allowed to be made to the images. The system administrator then runs the IA scheme's generator algorithm to produce the system keys: two public keys for proving and verification, which are encoded in editing and viewing applications respectively, and a secret signing key, securely embedded in the cameras.[4] *Photographers* can then use the cameras to produce digitally signed images. Any *editor* can use the editing application to edit the image, by performing a permissible transformation on it, and to generate a proof of authenticity for the new image. The viewing application enables any *viewer* to present images while verifying their authenticity using their attached proofs.

Our goal is to construct such an IA scheme that applies for *any* defined set of permissible transformations. The proofs must be succinct, as otherwise simply attaching a signed copy of the image prior to the transformation to an altered one (and letting the verifier compare the two) would suffice. We also want the proofs to be recursive, i.e., it must be possible to prove the authenticity of an image using another authenticated copy, without having access to the original camera-signed image. Moreover, the proofs must be cryptographically hard to forge and must not reveal data about the image other than its authenticity. We begin by defining some basic terms, and proceed to the formal definition of an IA scheme.

A. Basic definitions

Image. An *image* I is a pair $I = (B, M)$ containing *a pixel matrix* B and *metadata* M. The pixel matrix $B \in \{0, 1, ..., 255\}^{3\times N\times N}$, for some integer N, represents an $N \times N$ matrix of pixels, each containing 3 bytes for the red, green and blue components.[5] N is not the actual size of the image, but merely the allocated size. The height and width of the image are specified in the metadata and can be any integer smaller or equal to N. Pixel values outside the real image area should be 0.

The *metadata* for an image, denoted M, is a list of key-value pairs, where the keys are unique and taken from some predetermined set. We assume M is represented as a binary string by some reasonable encoding (the precise encoding is inconsequential for the ensuing discussion), and constrain the length of this string to a fixed length m. Let $I[\mathsf{key}]$ denote the value in M with the key key, which we also call *the* key *field*. M must contain values for width and height keys that specify an image's real width and height.

Let $\mathcal{I}$ denote the set of all images, and let $\mathcal{I}_N$ denote the set of all N-*images*, i.e., with an $N \times N$ pixel matrix.

[4]For simplicity, we describe the scenario where the signing keys to be embedded in cameras are created from scratch by the IA scheme. In reality, existing signing keys embedded by camera manufacturers would probably be used; see Section V-G for this natural extension.

[5]We consider here only square matrices and 8-bit RGB pixels for clarity. This is trivially generalized.

Remark 1. We distinguish the notion of image as defined here from the notion of image *files* (e.g., PNG or JPEG). When we refer to image files we say so explicitly.

Transformation. An *image transformation* t is a function $t : \mathcal{I} \times \{0,1\}^\star \to \mathcal{I}$ such that for every N, if $I \in \mathcal{I}_N$, then $t(I,\gamma) \in \mathcal{I}_N$ for any γ. That is, an image transformation takes an N-image and some parameters as input, and outputs an N-image. We also require that for every N there is an upper bound K_N such that for all $I \in \mathcal{I}_N, \gamma \in \{0,1\}^\star$ there is a $\gamma' \in \{0,1\}^{K_N}$ such that $t(I,\gamma) = t(I,\gamma')$. In other words, the parameter string length is bounded for every N. Note that an image transformation may work on both the image's pixel matrix and its metadata (e.g., a crop transformation may change the width and height fields of an image). As a default, a transformation leaves the metadata as is, unless otherwise defined. It is also possible to define transformations that change *only* the metadata (e.g., edit an author field).

Permissible transformation. Let $\mathcal{T} = (t_1, ..., t_n)$ denote the set of *permissible transformations*, a set of n image transformations that were defined by the system administrator as authenticity preserving.[6]

Provenance. For some N-image I, a *provenance*[7] of I is an N-image J, a finite series of transformations $(u_1, ..., u_m)$ and a series of parameter strings $\Gamma = (\gamma_1, ..., \gamma_m)$ such that if we take $I_1 = J$ and $I_{i+1} = u_i(I_i, \gamma_i)$ for $1 \le i \le n$, then $I_{m+1} = I$. In this case we say that I has a provenance originating in J. We call a provenance of I a *permissible provenance* if all of the u_i are permissible transformations.

Original image. Let $\mathcal{S} = (G_\mathsf{s}, S_\mathsf{s}, V_\mathsf{s})$ be an existentially unforgeable digital signature scheme [24]. For an image I, a signature verification key p_s and a signature σ, we say that I is *original with respect to* p_s *according to* σ if $V_\mathsf{s}(p_\mathsf{s}, I, \sigma) = 1$. When it is clear what key is relevant, we only say that I is original according to σ, and if σ is specified in that context too, we may just say that I is original.[8]

Authentic image. Given p_s and $\mathcal{T}$ as before, let $e = (O, (u_1, .., u_m), \Gamma)$ be a provenance and σ a signature. We say an image I is *authentic with respect to* $(\mathcal{T}, p_\mathsf{s})$ *according to* (e, σ) if e is a permissible provenance of I *and* O is original with respect to p_s according to σ. (Note that every original image is authentic, according to the provenance of length 0.)

B. Image authentication

Given a set of permissible transformations $\mathcal{T}$ and a security parameter λ, and given an existentially unforgeable signature scheme $\mathcal{S} = (G_\mathsf{s}, S_\mathsf{s}, V_\mathsf{s})$, an *Image Authentication (IA)* scheme for $\mathcal{T}$ is a tuple $\mathbf{IA} = (\mathcal{S}, G_\mathsf{IA}, P_\mathsf{IA}, V_\mathsf{IA})$, where G_IA is called the *generator,* P_IA is called the *prover,* and V_IA is called the *verifier*.

The generator $G_\mathsf{IA}\left(1^N, 1^\lambda\right) \to (pk_\mathsf{IA}, vk_\mathsf{IA}, sk_\mathsf{IA})$, given a maximal image size N and security parameter λ, probabilistically generates a secret signing key sk_IA, a public verification key vk_IA, which also contains a public key that matches sk_IA, i.e. $vk_\mathsf{IA} = vk'_\mathsf{IA} || p_\mathsf{s}$ (here and everywhere else, $||$ denotes string concatenation), and a public proving key pk_IA. G_IA is a preprocessing step and assumed to be executed once, in advance, by a trusted party (e.g., the camera manufacturer).

The prover $P_\mathsf{IA}(pk_\mathsf{IA}, I_{in}, \pi_{in}, t, \gamma) \to (I_{out}, \pi_{out})$, receives a proving key pk_IA, an N-image I_{in}, a proof π_{in} and a parameter string γ, and returns an edited image $I_{out} = t(I_{in}, \gamma)$ with a proof for its authenticity π_{out}.

The verifier $V_\mathsf{IA}(vk_\mathsf{IA}, I, \pi) \to 0/1$, receives a verification key vk_IA, an N-image I and a proof π and returns a decision whether I is authentic.

The signature scheme's signing algorithm S_s is intended to be executed by a trusted party (e.g., a secure camera). For simplicity, we consider a scenario where every device is embedded with the same secret key, generated by the generator. This is easily generalized to a system with certificate chains and possibility for revocation, as discussed in Section V-G.

The following properties must hold for $G_\mathsf{IA}, P_\mathsf{IA}, V_\mathsf{IA}$.[9]

Completeness. An IA scheme $\mathbf{IA}$ is *complete* if it is always possible to successfully prove the authenticity of an authentic image. That is, for every N and for every N-image I with a permissible provenance $e = (O, (u_1, ..., u_n), \Gamma)$:

$$\Pr\left[V_\mathsf{IA}(vk_\mathsf{IA}, I, \pi) = 1 \;\middle|\; \begin{array}{r}(pk_\mathsf{IA}, vk_\mathsf{IA}, sk_\mathsf{IA}) \leftarrow G_\mathsf{IA}\left(1^N, 1^\lambda\right) \\ \sigma \leftarrow S_\mathsf{s}(sk_\mathsf{IA}, O) \\ (I, \pi) \leftarrow \texttt{prove}(e, \sigma)\end{array}\right] = 1$$

where we define $\texttt{prove}(e, \sigma)$ to yield (I_{n+1}, π_{n+1}), computed using the recurrence relation:

$$I_1 \leftarrow O,\ \pi_1 \leftarrow \sigma, \text{ and } (I_{i+1}, \pi_{i+1}) \leftarrow P_\mathsf{IA}(pk_\mathsf{IA}, I_i, \pi_i, u_i, \gamma_i).$$

Unforgeability. The unforgeability of the signature scheme $\mathcal{S}$ still holds, even when considering adversaries that receive pk_IA and vk_IA as an auxiliary input.

Proof-of-knowledge. If the verifier accepts a proof π for an image I generated by some adversary, then the image is authentic, and moreover, the adversary "knows" a permissible provenance of I and a signature according to which I is authentic. Formally, an IA scheme $\mathbf{IA}$ *has the proof-of-knowledge property* if, for every polynomial-time stateful adversary A, there is a polynomial-time knowledge extractor E that can output a provenance and a signature, such that for every N and a large enough λ, and every polynomial-length series of N-images $(I_1, .., I_r)$ for some $r \in \text{poly}(\lambda)$, it holds

[6] This is easily generalized to image *relations* that check conditions between input and output images (e.g., each pixel value changes by at most 1).

[7] The word provenance is originally used for referring for a work of art's chronology of custody.

[8] A more general definition would be to use some *originality decider* that decides whether a given image is original, according to some witness. This way, the construction does not necessarily depend on a digital signature scheme. We chose to stick with the more specific version, as it is more intuitive and practical.

[9] These are adapted from the general definitions of PCD in Appendix D of [7]. See also Section III.

that:

$$\Pr\left[\begin{array}{r|r} V_{\mathsf{IA}}(vk_{\mathsf{IA}},I,\pi)=1 & (pk_{\mathsf{IA}},vk_{\mathsf{IA}},sk_{\mathsf{IA}})\leftarrow G_{\mathsf{IA}}\left(1^N,1^\lambda\right) \\ D_{\mathcal{T}}(vk_{\mathsf{IA}},I,e,\sigma)=0 & \sigma_i\leftarrow S_{\mathsf{S}}(sk_{\mathsf{IA}},I_i),1\leq i\leq r \\ & (I,\pi)\leftarrow A\left(pk_{\mathsf{IA}},vk_{\mathsf{IA}},(I_i,\sigma_i)_i\right) \\ & (e,\sigma)\leftarrow E\left(pk_{\mathsf{IA}},vk_{\mathsf{IA}},(I_i,\sigma_i)_i\right) \end{array}\right] \leq \mathsf{negl}\left(\lambda\right) \tag{1}$$

Above, $D_{\mathcal{T}}\left(vk_{\mathsf{IA}},I,e,\sigma\right)$ is the procedure that returns 1 iff I is authentic with respect to $\mathcal{T},p_{\mathsf{S}}$ according to e and σ, where p_{S} is the public key of the signature scheme, which is contained in vk_{IA}.

Succinctness and efficiency. IA proofs are short and easy to verify, i.e., $V_{\mathsf{IA}}\left(vk_{\mathsf{IA}},I,\pi\right)$ runs in time $O_\lambda\left(|I|\right)$ and an honestly generated proof is of length $O_\lambda\left(1\right)$ (where O_λ hides a fixed polynomial in λ). The prover and generator run in polynomial time in their inputs and in the (worst-case) running times of the transformations in $\mathcal{T}$ on images of size at most N.[10]

Statistical zero-knowledge. The IA proofs reveal nothing beyond the authenticity of the image. Formally, proofs are *statistical zero-knowledge:* there exists a polynomial-time stateful simulator S such that for every stateful distinguisher D that creates and observes proofs of legitimate images of its choice, the distinguisher D cannot tell whether it is interacting with a real prover or with the simulator S which does not even know the original image. Formally, the following two probabilities are negligibly close (in λ):

$$\Pr\left[\begin{array}{r|r} t\in\mathcal{T} & 1^N\leftarrow D\left(1^\lambda\right) \\ V_{\mathsf{IA}}(vk_{\mathsf{IA}},I_{in},\pi_{in})=1 & (pk_{\mathsf{IA}},vk_{\mathsf{IA}},sk_{\mathsf{IA}})\leftarrow G_{\mathsf{IA}}\left(1^N,1^\lambda\right) \\ D(\pi)=1 & (I_{in},\pi_{in},t,\gamma)\leftarrow D(pk_{\mathsf{IA}},vk_{\mathsf{IA}},sk_{\mathsf{IA}}) \\ & (I,\pi)\leftarrow P_{\mathsf{IA}}(pk_{\mathsf{IA}},I_{in},\pi_{in},t,\gamma) \end{array}\right] \tag{2}$$

$$\Pr\left[\begin{array}{r|r} & 1^N\leftarrow D\left(1^\lambda\right) \\ t\in\mathcal{T} & (pk_{\mathsf{IA}},vk_{\mathsf{IA}},sk_{\mathsf{IA}})\leftarrow S\left(1^N,1^\lambda\right) \\ V_{\mathsf{IA}}(vk_{\mathsf{IA}},I_{in},\pi_{in})=1 & (I_{in},\pi_{in},t,\gamma)\leftarrow D(pk_{\mathsf{IA}},vk_{\mathsf{IA}},sk_{\mathsf{IA}}) \\ D(\pi)=1 & I\leftarrow t(I_{in},\gamma) \\ & \pi\leftarrow S(I) \end{array}\right] \tag{3}$$

Remark 2. As mentioned, the signing algorithm should be executed by a trusted party, e.g., a secure camera (see Section A). sk_{IA} should be available only to the generator algorithm and the signing algorithm. In practice, this also means that the key should be kept in a protected storage, thus unavailable to outside untrusted users.

Remark 3. Zero-knoweldge (ZK) is a very useful property for an IA scheme to have (e.g., to ensure that private or embarrassing details that were cropped out of an image remain secret), but one may ask what is the cost of making it a requirement. In our particular case, our construction yields zero-knowledge "for free", since the succinctness of the proofs relies on SNARK constructions, and today's most efficient SNARK constructions [43], [22], [4] provide ZK with negligible overhead over their non-ZK variant..

Remark 4. Our definition of proof-of-knowledge is non-standard. We discuss this in Section IV-C.

Remark 5. The prover algorithm is comprised of two parts: (i) creating the output image by applying the input transformation on the input image with the input parameters, and (ii) generating a new proof for the new output image. It is possible to separate these two steps, by defining an editor algorithm and a prover algorithm, e.g., to enable a user to edit an image with one utility and then generate a proof with another. While this may be desirable for some applications, it is easy to see that the two methods are equivalent from a cryptographic point of view. We chose to stick with the first one for two reasons. First, the interfaces are simpler and more elegant. Second, the translation of parameters from the "image processing" form to the "arithmetic circuit input" form is handled internally without requiring the user's involvement.

Remark 6. `PhotoProof` leaves the specification of permissible transformations at the hands of the system administrator. It is up to the system administrator to make sure that the transformations chosen cannot be abused, for example, by being repeatedly applied to accumulate to overall undesirable changes. Note that in this case, `PhotoProof` can also restrict the number of transformations that can be performed to prevent such an abuse, see Section V-G.

III. Proof-Carrying Data

Proof-Carrying Data [11] is a cryptographic primitive for enforcing properties of distributed computation executed by untrusted parties, using proofs attached to every message. As our construction of the IA scheme is based on PCD, we provide a brief and informal description of the concept and terminology. (See [12] for a detailed overview and [9], [7] for the latest definitions. Here we follow the definitions in Appendix D of [7]).

Consider the scenario of a distributed computation between multiple untrusting parties, where every party can receive inputs from previous parties, add its own local data, perform a computation and output its result to the following parties (see Figure 1). We can think of this computation as a directed acyclic graph, where nodes represent computation and edges represent messages between parties. Edges are labeled with the content of the message, and vertices are labeled with the content of their node's additional local data (if any). This graph is called the *transcript* of the computation.

The property to be enforced is encoded as a *compliance predicate*, denoted Π, which inspects a single node (i.e., its incoming input edges, outgoing output edges, and its local data) and accepts or rejects them. A transcript is said to be Π-*compliant* if Π accepts all nodes in the transcript. A message value m is said to be Π-compliant if it is the final edge in some Π-compliant transcript.

Given a compliance predicate Π, PCD transforms the original computation into an augmented computation that enforces

[10] In our construction, the dependence is *quasilinear* in the size of the arithmetic circuit that computes the compliance predicate, which contains the sub-circuits performing the transformations in $\mathcal{T}$.

compliance, in the following sense. Each party attaches, to each of its output messages, a proof to the claim that the message is Π-compliant. Thus, in particular, the final output of the computation can be verified to result from a Π-compliant computation by merely inspecting the final proof associated with that output. The running time of the prover and verifier, and the size of the proofs, are all essentially independent of the transcript's size.

More concretely, a *(preprocessing) PCD system* is a triplet of algorithm $(G_{\mathsf{PCD}}, P_{\mathsf{PCD}}, V_{\mathsf{PCD}})$.[11] The *generator* algorithm $G_{\mathsf{PCD}}\left(\Pi, 1^{\lambda}\right)$, given a compliance predicate Π and a security parameter λ, probabilistically outputs a pair of public keys: *proving key* pk_{PCD} and *verification key* vk_{PCD}. The generated keys can be used by all parties to prove and verify messages for an unlimited number of times. For input messages $\vec{z}_{in}$ with matching proofs $\vec{\pi}_{in}$, local data l and an output message z_{out}, the *prover* algorithm $P_{\mathsf{PCD}}\left(pk_{\mathsf{PCD}}, \vec{z}_{in}, \vec{\pi}_{in}, l, z_{out}\right)$ outputs a new proof π_{out} attesting that z_{out} is Π-compliant (if this is indeed the case). Given a message z and its proof π, the *verifier* algorithm $V_{\mathsf{PCD}}\left(vk_{\mathsf{PCD}}, z, \pi\right)$ decides whether the proof is valid. See Appendix D of [7] for the full definitions.

A PCD system has the following properties. *Completeness*: for any result of a Π-compliant transcript, the prover can generate a message that convinces the verifier. *Succinctness:* proof size is $O_{\lambda}\left(1\right)$ and V_{PCD} runs in $O_{\lambda}\left(|z|\right)$ (and, in particular, is independent of the size of Π or the transcript). *Proof-of-knowledge*, which is a strengthening of *soundness*: an adversary that successfully convinces the verifier of the claim that some message z is Π-compliant "knows" of a Π-compliant transcript which outputs z. The adversary can get additional polynomial-length auxiliary input, which is chosen before the PCD keys are generated. *(Statistical) zero-knowledge*: a proof for a message does not contain information about the transcript which produced it.

On the theoretical side, PCD was shown to be constructable by recursive composition of SNARK proofs [9] (which are, essentially the single-message case of PCD). SNARK constructions are known, and some are implemented, on the basis of knowledge-of-exponent assumptions [25], [35], [10], [43], [6], [4], [22] and extractable collision resistant hashes [8], [39], [10]. Recently, Ben-Sasson et al. [7] presented the first implementation of preprocessing PCD, which was also made public via the `libsnark` library [47]; see below for further details.

Expressing compliance predicates. As discussed, PCD proves "compliance" as expressed by a predicate Π that applies to every node of the transcript. We did not specify, however, what language is used to express Π.

One possibility is to represent it as a computer program using some programming language. There are implementations of SNARKs (the 1-hop private case of PCD) for correct execution of C programs [43], [7], [56], [4], [6]. However, compiling C programs into an underlying "native" level of SNARKs, such as *Quadratic Arithmetic Programs (QAP)* [22]) comes at a large overhead.

For this reason we chose to encode our compliance predicate using a low-level language called the Rank 1 Constraint System (R1CS), an NP-complete language similar to arithmetic circuit satisfiability but allowing general bilinear gates at unit cost (see Appendix E.1 in [5] for more details). This allows a tight reduction to QAP, and preserves its expressive power.

Definition 7. A *Rank 1 Constraint System* $\mathcal{S}$ *of size* $n \in \mathbb{N}$ *over finite prime order field* $\mathbb{F}_p$ ($\mathbb{F}_p$-R1CS) is defined by vectors $\vec{a_i}, \vec{b_i}, \vec{c_i} \in \mathbb{F}_p^{m+1}$, $i \in \{1, \ldots n\}$. For a vector $\vec{\alpha} \in \mathbb{F}_p^m$, we say that $\vec{\alpha}$ *satisfies* $\mathcal{S}$ if $\langle \vec{a}_i, (1, \vec{\alpha}) \rangle \left\langle \vec{b}_i, (1, \vec{\alpha}) \right\rangle = \langle \vec{c}_i, (1, \vec{\alpha}) \rangle$ for all $i \in \{1, \ldots n\}$, where $\langle \cdot, \cdot \rangle$ denotes the inner product of vectors.

The language of $\mathcal{S}$ is the set of all vectors $\vec{x} \in \mathbb{F}_p^k$, $k \leq m$. such that $\vec{x}$ can be extended to a vector that satisfies $\mathcal{S}$, i.e., there is a $\vec{w} \in \mathbb{F}_p^{m-k}$ s.t. $(\vec{x}, \vec{w})$ satisfies $\mathcal{S}$.

As discussed in Section V, our `PhotoProof` implementation use `libsnark` [47], an open-source C++ library which implements the preprocessing SNARK scheme of [22] and PCD of Ben Sasson et al. [7]. This PCD implementation receives its compliance predicates as an $\mathbb{F}_p$-R1CS, and all of its messages, local data and proofs are vectors over $\mathbb{F}_p$. The `libsnark` library also includes implementation of *gadget* classes, used to construct instances of R1CS recursively. A gadget has two main functionalities: generate a constraint system and generate an assignment. For example, a gadget for a relation $R\left(x, y\right)$ can (i) generate a constraint system (that includes x and y as variables) which can be satisfied iff $R\left(x, y\right) = 1$ and (ii) given assignments to x, y compute a witness assignment that satisfies this constraint system.

Remark 8. It is usually more intuitive to think of Π as an arithmetic circuit, in which all the wires carry values from $\mathbb{F}_p$ and gates are field operations. There is a linear-time reduction from any arithmetic circuit to a R1CS. Briefly, the reduction is performed by adding a variable for each wire of the circuit, and adding a rank-1 constraint on those variables for each gate. The resulting system size is linear in the number of gates of the original circuit. The full details can be found in Appendix E.1 of [5].

IV. Image Authentication using PCD

A. Construction

For any set of image transformations $\mathcal{T}$, we construct an IA for $\mathcal{T}$, from a preprocessing PCD system and an existentially unforgeable digital signature scheme [24]. We call this construction `PhotoProof` (`PP`).

The layout of the construction is as follows. The `PhotoProof` generator, given a maximal image size N, first translates the set of permissible transformations into a compliance predicate Π, which, given two N-images (and some

[11] We follow the definition of [7], which is a somewhat stricter version than the one in [11], in the sense that we assume that for any compliance predicate Π, the generator G_{PCD} can generate the appropriate keys efficiently (the original definition only requires that a G^{Π}_{PCD} exists, i.e., G_{PCD} may be non-uniform). In a non-preprocessing PCD system, there are no keys or G.

auxiliary input), checks whether the images represent a permissible transformation's input-output pair (for some parameters). The generator also creates a digital signature key pair and calls the PCD generator on the compliance predicate to create the proof system; images are signed inside the camera, using the secret signing key, and the prover and verifier apply the PCD prover and verifier to generate/check proofs.

Formally, let $(G_{\mathsf{PCD}}, P_{\mathsf{PCD}}, V_{\mathsf{PCD}})$ be a PCD system, and let $\mathcal{S} = (G_{\mathsf{S}}, S_{\mathsf{S}}, V_{\mathsf{S}})$ be an existentially unforgeable digital signature scheme's generation, signing and verification algorithms.

Our PCD system will be defined over messages of image and public-key pairs, i.e., $z = (I, p_{\mathsf{S}})$.

We first define the following compliance predicate:

Algorithm 1 compliance predicate $\Pi^{\mathcal{T}}(z_{in}, z_{out}, t, \gamma)$

Input: incoming and outgoing messages $z_{in} = (I_{in}, p_{in})$, $z_{out} = (I_{out}, p_{out})$, an image transformation t and a parameter string γ.
Output: 0/1.

if $z_{in} = \perp, t = \perp$ **and** γ is a digital signature **then**
  return $V_{\mathsf{S}}(p_{out}, I_{out}, \gamma)$
if $t \in \mathcal{T}$ **and** $t(I_{in}, \gamma) = I_{out}$ **and** $p_{in} = p_{out}$ **then**
  return 1
return 0

The compliance predicate deals with two situations. For the base case, where there is no input image but only output image, it verifies the image's signature using the given public key.[12] For any other case, it checks whether the transformation between the input and the output image is indeed permissible *and* also checks that the given public key is not changed. The goal of including the public key in the message is for allowing the final verifier, which knows the public key that appears in the system's verifying key, to be convinced that the same public key was used for the signature verification of the original image. Another way of achieving this could have been encoding the signature verification key inside the compliance predicate (to yield $\Pi^{\mathcal{T}}_{p_{\mathsf{S}}}$). The main drawback in doing so is that the PCD keys become dependent in the signature keys, which complicates the construction and its security proof.

We continue to define the main `PhotoProof` algorithms in Algorithms 2–4.

[12]We implemented a slightly modified version due to efficiency considerations, see Section V-F.

Algorithm 2 `PhotoProof` generator $G_{\mathsf{PP}}(1^N, 1^\lambda)$

Input: a maximal image size N and a security parameter λ.
Output: a proving key pk_{PP}, a verification key vk_{PP} and a signing key sk_{PP}.

$(s_{\mathsf{S}}, p_{\mathsf{S}}) \leftarrow G_{\mathsf{S}}(1^\lambda)$
{generate a secret key and a public key of the signature scheme}
generate an $\mathbb{F}_p$-R1CS instance C_N which computes $\Pi^{\mathcal{T}}$ when applied on N-images.
$(pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}) \leftarrow G_{\mathsf{PCD}}(C_N, 1^\lambda)$
{generate PCD keys}
return $(pk_{\mathsf{PCD}} || p_{\mathsf{S}}, vk_{\mathsf{PCD}} || p_{\mathsf{S}}, s_{\mathsf{S}})$

Algorithm 3 `PhotoProof` prover $P_{\mathsf{PP}}(pk_{\mathsf{PP}}, I_{in}, \pi_{in}, t, \gamma)$

Input: a proving key pk_{PP}, an N-image I_{in}, a proof π_{in}, an image transformation t and a parameter string γ.
Output: an edited image I_{out} and a proof π_{out}.

parse pk_{PP} as $pk_{\mathsf{PCD}} || p_{\mathsf{S}}$
if π_{in} is a digital signature string **then**
  $\pi'_{in} \leftarrow P_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, \perp, \perp, \pi_{in}, (I_{in}, p_{\mathsf{S}}))$
  {"convert" the signature to PCD proof by calling the PCD prover}
else
  $\pi'_{in} \leftarrow \pi_{in}$
$I_{out} \leftarrow t(I_{in}, \gamma)$
$l \leftarrow (t, \gamma)$
$z_{in} \leftarrow (I_{in}, p_{\mathsf{S}})$
$z_{out} \leftarrow (I_{out}, p_{\mathsf{S}})$
$\pi_{out} \leftarrow P_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, z_{in}, \pi'_{in}, l, z_{out})$
return π_{out}

Algorithm 4 `PhotoProof` verifier $V_{\mathsf{PP}}(vk_{\mathsf{PP}}, I, \pi)$

Input: a verification key vk_{PP}, an N-image I and a proof π.
Output: 0/1.

parse vk_{PP} as $vk_{\mathsf{PCD}} || p_{\mathsf{S}}$
if π is a digital signature **then**
  return $V_{\mathsf{S}}(p_{\mathsf{S}}, I, \pi)$
if π is a PCD proof **then**
  return $V_{\mathsf{PCD}}(vk_{\mathsf{PCD}}, (I, p_{\mathsf{S}}), \pi)$
return 0

B. Proof of security

We now sketch the proof that `PhotoProof` fulfills the requisite properties.

Theorem 9. *For any set of polynomial-time image transformations $\mathcal{T}$, and given PCD and an existentially unforgeable digital signature scheme, the corresponding* $\mathtt{PP} = (\mathcal{S}, G_{\mathsf{PP}}, P_{\mathsf{PP}}, V_{\mathsf{PP}})$ *is an IA scheme for $\mathcal{T}$.*

Proof sketch. In the following, recall that a `PhotoProof` proof π may be of one of two types: a digital signature or a PCD proof. We now prove the different properties.

Succinctness. This follows from the efficiency of digital signatures and the succinctness property of PCD. The running time of V_{PP} is essentially the sum of V_{PCD}'s and V_{S}'s running time, both of which are $O_\lambda(|I|)$ regardless of the predicate $\Pi^{\mathcal{T}}$. G_{PP} consists of a call to G_{S} (which is polynomial in the security parameter λ), a generation of C_N that computes $\Pi^{\mathcal{T}}$ for N-images, which is polynomial in its worst-case running time, and a call to G_{PCD} on the generated R1CS instance, which from the PCD properties is quasilinear in the size of C_N. P_{PP} essentially performs an efficient transformation $t \in \mathcal{T}$ and runs P_{PCD} at most twice, and thus also fulfills the requirement.

Completeness. Let e be a permissible provenance $e = (O, (u_1, ..., u_n), \Gamma)$ and a signature $\sigma = S_{\mathsf{S}}(sk_{\mathsf{PP}}, O)$. First note that the correctness of $\mathcal{S}$ guarantees that $V_{\mathsf{S}}(p_{\mathsf{S}}, O, \sigma) = 1$. Hence, for every step of `prove`, the compliance predicate $\Pi^{\mathcal{T}}$ is satisfied. PCD completeness then yields that every proof generated inside `prove` will convince the PCD verifier with probability 1. Therefore the final proof will also convince it with probability 1.

Unforgeability. This property trivially holds for our construction. We need to show that knowing $pk_{\mathsf{PP}}, vk_{\mathsf{PP}}$ does not help an adversary attacking the signature scheme, except with a negligible probability. Having $pk_{\mathsf{PP}}, vk_{\mathsf{PP}}$ is the same as having $pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}, p_{\mathsf{S}}$. The PCD keys are (randomly) generated independently from the signature keys, and the signature scheme is secure against adversaries with access to the public key and some (key-independent) auxiliary input.

Proof-of-knowledge. Using the terminology of [7], we can look at a provenance of an image as a *distributed computation transcript* T, where transformations and parameter strings are the nodes' local data, and images are the messages on edges. The IA proof-of-knowledge then follows from the PCD proof-of-knowledge (PCD-PoK). Indeed, let A be a polynomial-time adversary attacking the `PhotoProof` scheme. We need to show a polynomial-time extractor E such that, whenever A convinces V_{PP} that some image I is authentic using a proof π, E produces the evidence (provenance e and signature σ) of authenticity, i.e., Eq. 1 holds.

Using A, we construct A_{PCD}, an adversary attacking the PCD scheme. Recall that PCD-PoK allows for the adversary and the extractor to be given an additional auxiliary-input string (chosen prior to key generation). Our A_{PCD} will interpret its auxiliary input as a series of images with matching signatures $(I_i, \sigma_i)_i$, along with a matching public key $p_{\mathcal{S}}$, will run $A(pk_{\mathsf{PCD}} || p_{\mathsf{S}}, vk_{\mathsf{PCD}} || p_{\mathsf{S}}, (I_i, \sigma_i))$, and will then output the PCD message and a proof corresponding to A's output. PCD-PoK then guarantees that there is an extractor E_{PCD} such that (for every N and large enough λ and every (polynomial-length) auxiliary input a, the following holds:

$$\Pr\left[\begin{array}{r|l} V_{\mathsf{PCD}}(vk_{\mathsf{PCD}}, z, \pi) = 1 & (pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}) \leftarrow G_{\mathsf{PCD}}\left(C_N, 1^\lambda\right) \\ \mathsf{out}(\mathsf{T}) \neq z \text{ or } C_N(\mathsf{T}) = 0 & (z, \pi) \leftarrow A_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}, a) \\ & \mathsf{T} \leftarrow E_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}, a) \end{array}\right] \leq \mathsf{negl}(\lambda) \tag{4}$$

where $out(\mathsf{T})$ denotes the last message of the transcript T, and $C_N(\mathsf{T})$ returns 1 iff T is C_N-compliant.

Note that the probability in Eq. 1 is over the generation of $(pk_{\mathsf{PP}}, vk_{\mathsf{PP}}, sk_{\mathsf{PP}})$, while in Eq. 4 it is only on the generation of the PCD keys. However, PCD-PoK holds for every a, and in particular such a that is generated by first generating $(p_{\mathsf{S}}, sk_{\mathsf{PP}}) \leftarrow G_{\mathsf{S}}\left(1^\lambda\right)$ and choosing any r images $I_1, ..., I_r$, and then taking $a = (p_{\mathsf{S}}, (I_1, S_{\mathsf{S}}(sk_{\mathsf{PP}}, I_1)), ..., (I_r, S_{\mathsf{S}}(sk_{\mathsf{PP}}, I_1)))$. So the probability remains negligible even when adding the generation of $(p_{\mathsf{S}}, sk_{\mathsf{PP}})$ to the probability space, i.e., for every $(I_1, .., I_r)$,

$$\Pr\left[\begin{array}{r|l} V_{\mathsf{PCD}}(vk_{\mathsf{PCD}}, z, \pi) = 1 & (p_{\mathsf{S}}, sk_{\mathsf{PP}}) \leftarrow G_{\mathsf{S}}\left(1^\lambda\right) \\ \mathsf{out}(\mathsf{T}) \neq z \text{ or } C_N(\mathsf{T}) = 0 & \sigma_i \leftarrow S_{\mathsf{S}}(sk_{\mathsf{PP}}, I_i), 1 \leq i \leq r \\ & (pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}) \leftarrow G_{\mathsf{PCD}}\left(C_N, 1^\lambda\right) \\ & (z, \pi) \leftarrow A_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}, a) \\ & \mathsf{T} \leftarrow E_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}, a) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Now, we define the extractor E as the algorithm that (a) when A outputs an image and a digital signature, outputs the same image and signature and (b) when A outputs an image and a proof, invokes E_{PCD} and reads off the permissible provenance and signature from the output transcript's graph labels. So the above probability implies:

$$\Pr\left[\begin{array}{r|l} V_{\mathsf{PCD}}(vk_{\mathsf{PCD}}, z, \pi) = 1 & (pk_{\mathsf{PP}}, vk_{\mathsf{PP}}, sk_{\mathsf{PP}}) \leftarrow G_{\mathsf{PP}}\left(1^N, 1^\lambda\right) \\ D_{\mathcal{T}}(vk_{\mathsf{PP}}, z, e, \sigma) = 0 & \sigma_i \leftarrow S_{\mathsf{S}}(sk_{\mathsf{PP}}, I_i), 1 \leq i \leq r \\ & (I, \pi) \leftarrow A\left(pk_{\mathsf{PP}}, vk_{\mathsf{PP}}, (I_i, \sigma_i)_i\right) \\ & (e, \sigma) \leftarrow E\left(pk_{\mathsf{PP}}, vk_{\mathsf{PP}}, (I_i, \sigma_i)_i\right) \end{array}\right] \leq \mathsf{negl}(\lambda) \tag{5}$$

Now, Eq. 1 follows by splitting it into two cases, according to the proof that A outputs. The case of a signature trivially holds. For the case of a PCD proof, it follows from $V_{\mathsf{PP}}(vk_{\mathsf{PP}}, I, \pi) = V_{\mathsf{PCD}}(vk_{\mathsf{PCD}}, z, \pi)$ and Eq. 5.

Statistical zero-knowledge. We need to show a polynomial-time stateful simulator S_{PP} such that for every stateful distinguisher D_{PP} the probabilities in Eq. 2 and 3 are negligibly close. By the statistical zero-knowledge of the underlying PCD, there exists a simulator S_{PCD} such that for every distinguisher D_{PCD} that (given 1^λ) outputs some compliance predicate Π', and given proving and verification keys outputs some $(\vec{z}_{in}, \vec{\pi}_{in}, l, z_{out})$, D_{PCD} cannot distinguish between a PCD-generated or a S_{PCD}-generated proof for z_{out} with more than negligible probability. That is, the following two probabilities are negligibly close (in λ):

$$\Pr\left[\begin{array}{r|l} \Pi'(\vec{z}_{in}, l, \pi) = 1 & \Pi' \leftarrow D_{\mathsf{PCD}}\left(1^\lambda\right) \\ V_{\mathsf{PCD}}(vk_{\mathsf{PCD}}, \vec{z}_{in}, \vec{\pi}_{in}) = 1 & (pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}) \leftarrow G_{\mathsf{PCD}}\left(\Pi', 1^\lambda\right) \\ D_{\mathsf{PCD}}(\pi) = 1 & (\vec{z}_{in}, \vec{\pi}_{in}, l, z_{out}) \leftarrow D_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}) \\ & \pi \leftarrow P_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, \vec{z}_{in}, \vec{\pi}_{in}, l, z_{out}) \end{array}\right]$$

$$\Pr\left[\begin{array}{r|l} \Pi'(\vec{z}_{in}, l, \pi) = 1 & \Pi' \leftarrow D_{\mathsf{PCD}}\left(1^\lambda\right) \\ V_{\mathsf{PCD}}(vk_{\mathsf{PCD}}, \vec{z}_{in}, \vec{\pi}_{in}) = 1 & (pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}) \leftarrow S_{\mathsf{PCD}}\left(\Pi', 1^\lambda\right) \\ D_{\mathsf{PCD}}(\pi) = 1 & (\vec{z}_{in}, \vec{\pi}_{in}, l, z_{out}) \leftarrow D_{\mathsf{PCD}}(pk_{\mathsf{PCD}}, vk_{\mathsf{PCD}}) \\ & \pi \leftarrow S_{\mathsf{PCD}}(z_{out}) \end{array}\right]$$

This PCD simulator S_{PCD} is then used to construct the IA simulator S_{PP} needed to show the IA's statistical zero-knowledge. Let S_{PP} be the following simulator: when invoked

as $S_{\mathsf{PP}}\left(1^N,1^\lambda\right)$ it runs a modified version of G_{PP} (Algorithm 2) where instead of calling G_{PCD} it calls $S_{\mathsf{PCD}}(C_N,1^\lambda)$; when later invoked as $S_{\mathsf{PP}}(I)$ it simply runs and outputs $S_{\mathsf{PCD}}(I)$.

To see that this simulator succeeds, first note that the distinguisher in the IA zero-knowledge definition is weaker than that of the PCD zero-knowledge definition, since the former is limited to choosing only N to determine the compliance predicate, and can control only I_{in},π_{in},t and γ (and not I_{out}). Thus, from D_{PP} one can easily construct a distinguisher D_{PCD} for the PCD zero-knowledge, by tacking on to D_{PCD} the requisite fragments of the `PhotoProof` algorithms so that it presents a PCD interface instead of the more limited IA interface. Formally, when we expand G_{PP} and P_{PP} Eq. 2 becomes:

$$\Pr\left[\begin{array}{r|l} & 1^N\leftarrow D_{\mathsf{PP}}\left(1^\lambda\right) \\ & (p_{\mathsf{S}},s_{\mathsf{S}})\leftarrow G_{\mathsf{S}}\left(1^\lambda\right) \\ & C_N=\mathcal{C}\left(\Pi^{\mathcal{T}},1^N\right) \\ t\in\mathcal{T} & (pk_{\mathsf{PCD}},vk_{\mathsf{PCD}})\leftarrow G_{\mathsf{PCD}}\left(C_N,1^\lambda\right) \\ V_{\mathsf{PP}}(vk_{\mathsf{PP}},I_{in},\pi_{in})=1 & (pk_{\mathsf{PP}},vk_{\mathsf{PP}},sk_{\mathsf{PP}})\leftarrow(pk_{\mathsf{PCD}}||p_{\mathsf{S}},vk_{\mathsf{PCD}}||p_{\mathsf{S}},s_{\mathsf{S}}) \\ D_{\mathsf{PP}}(\pi)=1 & (I_{in},\pi_{in},t,\gamma)\leftarrow D_{\mathsf{PP}}(pk_{\mathsf{PP}},vk_{\mathsf{PP}},sk_{\mathsf{PP}}) \\ & \left(z_{in},\pi'_{in},l,z\right)\leftarrow F_D(I_{in},\pi_{in},t,\gamma) \\ & (\pi)\leftarrow P_{\mathsf{PCD}}\left(pk_{\mathsf{PCD}},z_{in},\pi'_{in},l,z\right) \end{array}\right]$$

where $F_D\left(I_{in},\pi_{in},t,\gamma\right)$ contains the steps of P_{PP} which convert a signature to a PCD proof when necessary and output the data in its PCD form (i.e., as messages and local data). Now we define D_{PCD} by repacking the extra steps before/after D_{PP}'s execution, to be left only with the PCD interface, and obtaining:

$$\Pr\left[\begin{array}{r|l} C_N(z_{in},l,z)=1 & C_N\leftarrow D_{\mathsf{PCD}}\left(1^\lambda\right) \\ V_{\mathsf{PP}}(vk_{\mathsf{PP}},I_{in},\pi_{in})=1 & (pk_{\mathsf{PCD}},vk_{\mathsf{PCD}})\leftarrow G_{\mathsf{PCD}}\left(C_N,1^\lambda\right) \\ D_{\mathsf{PCD}}(\pi)=1 & \left(z_{in},\pi'_{in},l,z\right)\leftarrow D_{\mathsf{PCD}}(pk_{\mathsf{PCD}},vk_{\mathsf{PCD}}) \\ & \pi\leftarrow P_{\mathsf{PCD}}\left(pk_{\mathsf{PCD}},z_{in},\pi'_{in},l,z\right) \end{array}\right]$$

Using the same reasoning, Eq. 3 becomes:

$$\Pr\left[\begin{array}{r|l} C_N(z_{in},l,z)=1 & C_N\leftarrow D_{\mathsf{PCD}}\left(1^\lambda\right) \\ V_{\mathsf{PP}}(vk_{\mathsf{PP}},I_{in},\pi_{in})=1 & (pk_{\mathsf{PCD}},vk_{\mathsf{PCD}})\leftarrow S_{\mathsf{PCD}}\left(C_N,1^\lambda\right) \\ D_{\mathsf{PCD}}(\pi)=1 & \left(z_{in},\pi'_{in},l,z\right)\leftarrow D_{\mathsf{PCD}}(pk_{\mathsf{PCD}},vk_{\mathsf{PCD}}) \\ & \pi\leftarrow S_{\mathsf{PCD}}\left(pk_{\mathsf{PCD}},z_{in},\pi'_{in},l,z\right) \end{array}\right]$$

We now split to cases according to π_{in}. When π_{in} is a PCD proof, V_{PP} is the same as V_{PCD}, and so we are left with the exact probabilities from the PCD definition. When π_{in} is a digital signature, then $V_{\mathsf{PP}}\left(vk_{\mathsf{PP}},I_{in},\pi_{in}\right)=1$ iff $V_{\mathsf{S}}\left(p_{\mathsf{S}},I_{in},\pi_{in}\right)=1$ and whenever this holds, then the "conversion" to PCD succeeds, i.e., $V_{\mathsf{PCD}}\left(vk_{\mathsf{PCD}},z_{in},\pi'_{in}\right)=1$. From PCD soundness, the opposite holds too except for negligible probability.

Thus, the distinguishing advantage in IA (difference between Eq. 2 and 3) for S_{PP} and D_{PP} is the same as the distinguishing advantage in the PCD zero-knowledge definition for S_{PCD} and D_{PCD} up to a negligible difference and the latter advantage is negligible, by definition of S_{PCD}. □

C. Strengthening proof-of-knowledge

Proof-of-knowledge of IA schemes lets adversaries see signatures under $\mathcal{S}$ for any series of images (of polynomial size). Note, though, that these images are chosen before pk_{IA},vk_{IA} and sk_{IA} are generated. A stronger notion of proof-of-knowledge would be to consider adversaries with a signing oracle, so they can query it for images that are adaptively chosen after the system keys are generated.

To see why this definition is stronger than the one we used, consider an artificially weakened version of our `PhotoProof` construction, where the verifier algorithm is added a "trapdoor" rule, to accept any image that is given along with a proof π which contains a valid signature on the proving key (i.e.,$V_{\mathsf{S}}\left(p_{\mathsf{S}},pk_{\mathsf{IA}},\pi\right)=1$). This is of course undesirable, and indeed this artificial construction does not fulfill the stronger notion of proof-of-knowledge. However, it does fulfill the original IA definition, since the adversary there cannot ask for signatures after seeing pk_{IA}, and thus cannot produce an image with the "trapdoor" signature.

The problem with the stronger definition is that the PCD proof-of-knowledge does not guarantee anything against adversaries with oracle access. In a recent work, Fiore and Nitulescu [18] study the security of SNARK proof-of-knowledge in scenarios where adversaries are given access to oracles. They too suggest a non-addaptive notion of proof-of-knowledge very similar to our IA proof-of-knowledge.

V. Implementation

A. System description

We implemented a `PhotoProof` prototype for a set of permissible transformations including identity, (arbitrary) rectangular crop, horizontal and vertical flip, transpose, general brightness/contrast adjustments and rotation. We also demonstrated the protection of metadata fields by adding protected timestamp data to our images. Although fully operational, our implementation is still more of a proof-of-concept than a real-world-ready system, due to relatively long proving times and large proving keys for small images. We review our prototype's performance in Section V-B, and list some ideas and directions for making the system usable in practice in Sections V-G and VI.

Our implementation makes use of `libsnark`[47], a C++ library implementing SNARKs and PCD. It contains the preprocessing PCD for $\mathbb{F}_p$-R1CS of Ben-Sasson et al. [7], using the SNARK scheme of Gennaro et al. [22]). The `libsnark` library creates proofs for computation (expressed $\mathbb{F}_p$-R1CS) over a large prime field $\mathbb{F}_p$ (chosen from a set offered by `libsnark`, corresponding to its supported elliptic curves), and offers a gadget library for expressing such computation (see Section III for more details).

At the heart of our implementation lies a collection of gadgets which form an image processing tool-box expressed in R1CS. Using them, we implement a gadget for each permissible transformation we support. All these gadgets are then combined to create the compliance predicate gadget,

which, given a maximal image size N, generates a R1CS which computes the compliance predicate on images of size up to N (and fixed size local data).

Our code has 4 levels (see Figure 3). At the higher level is Python code that handles image processing and user interface. It makes use of Pillow, the Python Imagining Library fork [38]. It also handles serializing images and parameters before sending to the next level. The second level supplies a C++ wrapper for `libsnark` functionality. Level 3 contains the PCD implementation of [7]. Level 4 is the `PhotoProof` gadget library. For example, to edit an image and generate a new proof, the user uses the Python prover function (level 1). This function applies the transformation on the image, and then serializes all input: hashes, images with metadata, input proof, transformation, parameters and other data. It then invokes the C++ executable containing the prover wrapper (level 2). This code takes all the input and prepares an assignment for the input variables to the compliance predicate. It then calls level 4 to generate the witness to the compliance predicate constraint system. Finally, it invokes `libsnark`'s PCD prover (level 3) on the witness and the input proof to generate the new proof, which it outputs back to the Python code, where it is output to the user.

Our implementation does not include a secure camera. To simulate original camera images, we use the generated secret signing key to sign our "original" images. The secret key is, of course, not used in any other part of the system. For digital signatures we use an ECDSA library [58] with a NIST192p curve (about 80 bits of security).

B. Performance

We ran our prototype on images of $N \times N$ pixels for various N values and measured the average running time of the generator, prover and verifier, as well as the sizes of the keys and the proofs. Our benchmark machine was a desktop with a 4-core 3.4GHz Intel i7-4770 processor and 32GB of RAM. Our results are summarized in Table II. The security of our scheme is guaranteed by PCD security, and indeed, for all the images and transformations we ran our prototype on (hundreds of executions), no completeness or soundness errors were recorded. Every illegal change of an image, even of a single bit, is detected by the scheme.

As expected, generation and proving times are much slower than verification times. One thing to keep in mind when assessing overall performance is that generation takes place only once in the lifetime of the system, by some trusted, preferably high-performance server; proving is done only once per edit, and can be delegated to (untrusted) cloud servers. Verification, to be done by all viewers of the image, is fast.

The reported proving times in Table II assume that the proving key pk_{PP} is preloaded into RAM. This holds, e.g., for scenarios where proving is done by a dedicated server or by a software plugin that loads the key into memory upon startup.

The measured proving times are per-edit, and independent of which transformation is applied on the image (as every proving step proves compliance with a predicate that "considers" all permissible transformations). When performing multiple edit steps (e.g., crop and then rotate) an additional proving step is needed for each transformation applied. In Section V-G we discuss using PCD with multiple compliance predicates, thus shortening the proving time per edit step by considering only the applied transformation.

Although 128×128 images are too small for practical use, our prototype is the first working proof-of-concept of an IA scheme (as defined in Section II-B). Proving time and key size grow quasi-quadratically in N, as expected from the algorithm's complexity and confirmed by Table II. Thus, for example, handling 1024×1024 images will increase proving time by a factor of less than 100. Since the cryptographic algorithms are highly amenable to parallelism via GPGPU, FPGA, and ASIC, we expect that such implementations will greatly improve performance and allow handling of larger images. Key size improvements are also possible, e.g., by using PCD with multiple compliance predicates (see Section V-G).

C. Instantiation challenges and solutions

There are many interesting challenges when instantiating `PhotoProof`. Many of them arise from the need to tailor advanced functionality to arithmetic circuits.[13]

In our implementation, compliance predicate size is a significant bottleneck, which affects both running time and (proving) key size. Current SNARK technology is on the borderline of feasibility. The SNARK implementation we use, which is the fastest currently available, can prove satisfaction of arithmetic circuits in approximately 0.1 milliseconds per gate [6]. As the size of a circuit that performs transformations on images depends on the maximal image size it can receive as an input, and real-world images are typically large (hundreds of kilobytes to few megabytes), the circuit should be carefully designed.

Efficiently implementing an image-processing operation such as an arithmetic circuit is an interesting problem. For our prototype we had to design circuits that perform rectangular crop, horizontal and vertical flip, transpose, general brightness/contrast adjustments and rotation. We next describe some of the challenges in doing this, using the example (one of many) of the image rotation transformation.

D. Case study: implementing image rotation

The rotation transformation, $\text{rotate}\,(I, \alpha) \rightarrow J$, rotates an image I by an angle α to create an image J (for this discussion we assume $0 \leq \alpha \leq \frac{\pi}{4}$). In its basic form, without interpolating pixels and anti-aliasing techniques, the algorithm sends to the pixel in position (x, y), the pixel in position $(x\cos\alpha - y\sin\alpha, x\cos\alpha + y\sin\alpha)$ (assuming rotation around $(0, 0)$).

Our goal is to design an arithmetic circuit that, given an $N \times N$ input image and output image, and optionally additional parameters, checks whether the output is a rotation of the

[13] Our compliance predicate is implemented as an R1CS, but it is more intuitive to think of it as an arithmetic circuit. See Remark 8.

Table II
PhotoProof PROTOTYPE RUNNING TIMES AND KEY SIZES FOR $N \times N$ IMAGES

N	#CP	G_{PP}		P_{PP}		V_{PP}		pk_{PP}(MB)	vk_{PP}(MB)	π(KB)
		avg. (s)	σ (%)	avg. (s)	σ (%)	avg. (s)	σ (%)			
16	171,815	16.92	1.7	13.97	0.1	0.09	2.1	144.4	2.7	2.67
32	706,959	32.9	0.1	25.2	0.3	0.1	2.2	255.5		
64	2,966,167	83.2	0.1	69.8	0.4	0.14	1.6	635.1		
128	12,531,999	367	0.5	306	0.7	0.5	9	2601.4		

Average and (normalized) standard deviation (σ) are over 10 iterations each. Reported proving times assume pk_{PP} is preloaded into RAM. #CP is the size of the generated compliance predicate.

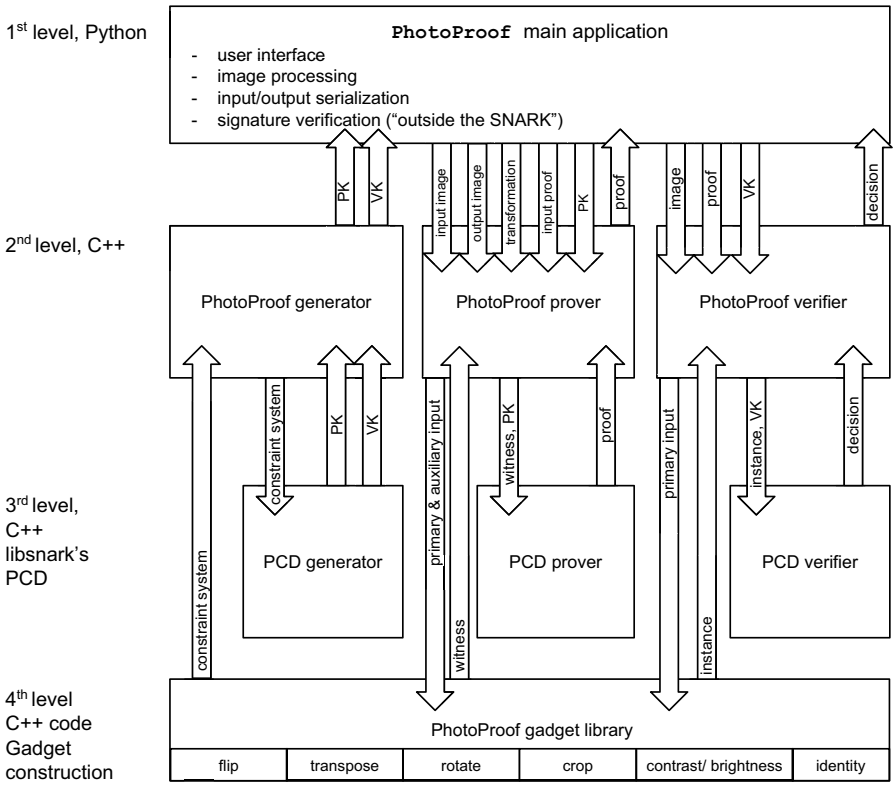

Figure 3. High-level design of the PhotoProof application.

input. Our circuit will rotate the input image and compare the result to the output image.

In the following, we list some general considerations, and exemplify each using the image rotation case.

Circuit design. *There are many differences between writing a computer program and writing an arithmetic circuit. For example, a program can receive input of arbitrary length and have a maximum running time of its worst code flow, while a circuit has a fixed size input and its size is the sum of all its sub-circuits. Another difference is that in circuits, as a rule of thumb, the more input gates affect each output gate, the larger the circuit has to be, at least when implemented naively.*

In image rotation, every output pixel depends on multiple input pixels, as there is an "arch" of pixels that can be sent to a given output pixel (depending on the given angle). A program that computes rotate will only require $O\left(N^2\right)$ operations. But a circuit has to wire many input pixels to each output pixel (as many as $O\left(N\right)$), depending on the specific algorithm. Naively, that will require $O\left(N^2\right)$ MUX gadgets (i.e., sub-circuits), a total of $O\left(N^3\right)$ gates. A better solution is to use the *rotation through shearing* of Paeth [42], which we outline here. An *x-shear* is a matrix in the form $\left(\begin{smallmatrix} 1 & a \\ 0 & 1 \end{smallmatrix}\right)$. Similarly, a *$y$-shear* is a matrix $\left(\begin{smallmatrix} 1 & 0 \\ b & 1 \end{smallmatrix}\right)$. As the name implies, x-shears work only in the x direction, that is, a point (x, y) is translated under an x-shear to $(x + ay, y)$. In the context of images, that means that x-shears work on rows while y-shears work on columns. Note this identity: $\left(\begin{smallmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{smallmatrix}\right) = \left(\begin{smallmatrix} 1 & -\tan\left(\frac{\alpha}{2}\right) \\ 0 & 1 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} 1 & 0 \\ \sin\alpha & 1 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} 1 & -\tan\left(\frac{\alpha}{2}\right) \\ 0 & 1 \end{smallmatrix}\right)$. Hence, it is possible to rotate an image using only shears: first, shear all rows with $a = -\tan\left(\alpha/2\right)$; then all columns with $b = \sin\alpha$; then, again all rows. Shears can be implemented efficiently in circuits using a barrel shifter, which

shifts a row of size k in $O(k \log k)$ gates. Shifting N rows, then N columns, and again N rows, sums up to $O\left(N^2 \log N\right)$ gates.

Nondeterministic programming. *Recall that in each computation step, PCD proves compliance of the local computation. The proof is attached to the output message, and ascertains, in particular, the existence of inputs and local data that (together with the output) satisfy the compliance predicate. The local data may provide arbitrary hints ("nondeterministic advice") that, while not trusted to be correct, may help check the relation between input and output messages more efficiently.*

In our rotation example, there is the issue of computing the trigonometric functions, given an angle α. This is hard to do over $\mathbb{F}_p$. Instead, it is possible to directly supply the circuit with the values of $a = \tan\left(\frac{\alpha}{2}\right)$, $b = \sin(\alpha)$, which are all that are needed for the shearing. This, however, requires adding a check that the given a and b are indeed derived from the same angle (knowing the value of this angle is not necessary). To do so, another nondeterminism trick can be used. The circuit will also be given $c = \sin\left(\frac{\alpha}{2}\right)$ and $d = \cos\left(\frac{\alpha}{2}\right)$, so that it only needs to check 3 arithmetic conditions: $c^2 + d^2 = 1$, $da = c$ and $2cd = b$.

Arithmetic over $\mathbb{F}_\mathbf{p}$. *All functionality that is incorporated into our compliance predicate needs to be implemented as an R1CS using the basic operations of $\mathbb{F}_p$. Functions that are "unnatural" in $\mathbb{F}_p$ can be hard to implement and end up requiring many field operations; these include integer comparison, fraction arithmetic and trigonometric functions.*

In image rotation (and everywhere else in our compliance predicate) we had to compute arithmetic of real values. We solve this by using a fixed-point representation for the numbers and storing them inside field elements as integers. This sometimes results in some small calculation error. For this reason, we allow for a small and configurable deviation from the aforementioned 3 conditions (less than 2^{-14} in our implementation). By ensuring that the rotation transformation applied on the input image (before the call to the PCD prover) is computed in the exact same way, we guarantee the same rounding errors will occur inside and outside the circuit, hence preserving completeness.

E. Implementing other transformations

Following is a complete list of the transformations we have implemented. For all our transformations (which are implemented as arithmetic circuits, as described above), images are represented as an $N \times N$ matrix, where the upper-left corner of the image is always at index $(0, 0)$, the image's real width and height are specified in the metadata (in width and height fields), and the pixels outside of the image borders are zeros. In all transformations, the input image is first transformed, then checked to see whether it is identical to the output image.

Identity. Checks whether the input and output image are identical. Identical images have the same pixel data as well as the same metadata.

Crop. The input image is first cropped, i.e., every pixel is either left unchanged or zeroed using MUX gates, where the chooser value is computed according to the pixel location with respect to the crop borders. Then, the image is translated (using barrel-shifters, as described for rotation), so its (new) upper-left pixel is moved to location $(0, 0)$ of the pixel matrix. Metadata is changed accordingly (e.g., width is set to the new width).

Transpose. Input image is transposed. The transpose operation requires only "rewiring" input pixels to corresponding output pixels. Transpose keeps the upper-left corner in place so no translation is needed.

Flip. Input image is flipped (similarly to transpose), then moved to the upper-left corner using barrel shifters (since flipping may move padding to location $(0, 0)$).

Image rotation. As described in Section V-D.

Contrast/brightness. Given α, β, each pixel of the input image undergoes the transformation $\lfloor \alpha x + \beta \rfloor$, where negative values are kept 0 and overflows are set to 255. The following conditions are checked: $0 \leq \alpha \leq 255$, $|\beta| \leq 255$. The real-value arithmetic is done using fixed-point representation, in the same way as described for rotation.

F. Signature verification

Another challenge is in verifying the original image's digital signature inside the compliance predicate.[14] From the circuit size perspective, this can be expensive, especially when the signature scheme is of some algebraic nature unrelated to the PCD field $\mathbb{F}_p$. We explored the following options:

$\mathbb{F}_p$-friendly signatures. One possibility to make this more efficient would be to use a signature scheme whose verification can be compactly expressed over $\mathbb{F}_p$. For example, one can use an RSA-based digital signature scheme with a small public-key exponent, $e = 3$, requiring just 3 modular multiplications to verify, and these can be efficiently implemented using radix $\lfloor \sqrt{p} \rfloor$ arithmetic. Leveraging nondeterminism, as discussed above, can further reduce the size of the circuit: for example, modular reduction can be implemented by big-integer multiplication (guessing the quotient and remainder) instead of division. Another option would be to use an elliptic-curve signature scheme in a curve over the field $\mathbb{F}_p$.

Signatures outside Π. We chose to implement a different approach: moving the signature verification out of the compliance predicate, and letting the `PhotoProof` verifier check the signature outside the PCD verifier.

For each original image, the camera computes an *original hash*, a collision-resistant hash digest of the original image, and signs it. This original hash and its signature are then passed on in every edit step. The compliance predicate is modified to check that the original hash either matches the image it received as the local input (this happens for the case of

[14] A recent work by Backes et al. [2] suggested `ADSNARK`, a proof system on authenticated data. Their work does not discuss recursive composition of proofs, nor the IA-specific considerations we described in this section.

the original image), or is passed from input to output without modification (thus forcing the original hash to stay the same for every image in the edit chain). Given an image, a proof, an original hash and a signature, the `PhotoProof` verifier checks that (a) the PCD proof is valid for the image with its attached original hash and that (b) the signature of the original hash is valid under the signature scheme's public key.

In other words, we move the signature verification from the first proving step to the last verification step, by using PCD to make sure that the original hash digest does not change along the way.

A collision resistant hash in an arithmetic circuit can be computed cheaply using the subset-sum CRH function over $\mathbb{F}_p$ [1], [23], as suggested in [7]. This results in a smaller compliance predicate than the one obtained by checking signatures in Π, but yields slightly larger proofs.[15]

Note that by attaching an original hash to each image, the zero-knowledge property of the proof system as defined above no longer holds (e.g., given two authentic images, it is possible to compare their associated original hashes and thereby deduce whether they originated in the same original image). However, we can make sure the original hash itself does not reveal any information about the original image, by making it a statistically-hiding commitment (e.g., a hash of the original image concatenated with sufficiently-long random padding). Thus, a slightly weaker zero-knowledge property still holds: the IA proof does not reveal information about the image other than the original hash and its signature.[16]

PCD-based signatures. Signature verification can also be removed from the PCD by assuming that the secure camera can run a PCD prover. In this case, the camera can output the original image along with a hash of its secret key, a certificate for this hash, and a PCD proof for the claim "the image was authorized by a camera which had access to a secret key with this specific hash digest" (e.g., by supplying the prover with a hash digest of the key and a hash on the image together with the same key). The key remains secret due to the zero-knowledge of the proof. This alternative offers the best of both worlds: full zero-knowledge and a compliance predicate of size similar to "signatures outside Π". However, it requires heavy PCD computation to be run on the camera's secure processor.

G. Additional features and extensions

Many additional currently unimplemented features can be incorporated into `PhotoProof`.

Certificates. It is imprudent to use a single secret key for all cameras. A large scale system with multiple devices should use revocable certificates (e.g., X.509 [51]). This can be done as follows: every manufactured camera is assigned a unique public-private pair of signing keys, and a certificate for the public key, chained up to a root certificate. The set of authorized root public keys should be specified in the IA proving key. The compliance predicate is then modified to include a check that the certificate chain, as well as the image's signature, are valid. Note that the identity of the camera that took the original image remains secret, thanks to zero-knowledge.

Revocation. A revocation mechanism may be useful in case some device's secret key is compromised. Continuing the above description of certificates, one way of doing so is adding a hash digest of the camera's public key to the image, and letting the compliance predicate (a) check it for the base case of the original image and (b) check that it remains unchanged after every transformation. Thus, keys can be revoked a posteriori, by a conventional key revocation mechanism (such as CRL or OCSP). Of course, images originating in the same camera can then be identified.

Multiple compliance predicates. A recent work [13] extends PCD to multiple compliance predicates, allowing us to use a separate compliance predicate for each permissible transformation, thereby making the proving costs dependent on the (constraint system) size of the transformation actually employed, rather than the sum of the (constraint system) sizes across all permissible transformations. This also allows for multiple smaller proving keys (instead of a single large key), thus making it more feasible for an entire key to fit in a proving machine's RAM during a proving operation.

Proof channel. Proofs can be attached to image files in an associated but separate "sidecar" file. Alternatively, they can be incorporated into the image file, within a metadata extension header (e.g., EXIF tags in JPEG and TIFF files). For seamless integration, the proof could even be embedded in the image pixels, using a lossless embedding technique (e.g. [28], [40]).

`PhotoProof` **plugin.** A plugin for image-editing software (e.g., the GNU Image Manipulation Program) will allow users to conveniently edit images and generate proofs. The plugin can handle `PhotoProof` algorithms and keys transparently. The user only needs to edit the image, as done in any image editing program, while the plugin keeps track of the applied (permissible) transformations. Only when the editing is complete does the plugin call the prover for the required number of times with the correct parameters, and output the proof.

Copyright message and metadata protection. Metadata information is sometimes as important as the image content, but can easily be edited or forged. In our prototype, we demonstrated protection of a metadata field by including a protected timestamp (see Section V-A). In the same way, it is possible to protect GPS location tags, name of camera owner, or any other type of information that can be added automatically by the camera.

We can also protect fields that are added "manually" by the user after the image was signed, e.g., caption, copyrights message, face tags, etc. This is made possible by allowing certain fields to be edited with access to the original image

[15] In our implementation we used ECDSA signatures of 384-bit length, which fit in 2 field elements. The original hash is an additional single field element.

[16] This new definition is the same as the zero-knowledge definition in Section II-B except the simulator is given the original hash and its signature in addition to the image.

only (this can be checked by the compliance predicate). The original can then be destroyed, ensuring that no one will be able to edit these fields without invalidating the proof.

Image provenance tracking. For some applications it may be desirable to keep track of (and perhaps limit) the list of transformations that the image went through (and their order). This is possible using a provenance metadata field. The original (signed) image will be generated with an empty list. The set of permissible transformations will include only transformations that append themselves to the provenance field. Note that the length of this field is bounded due to the limit on the overall image size. Alternatively, it is possible to keep track only of the length of the provenance. This, in particular, would mitigate the risk of numerous small permissible changes accumulating into an overall change that is considered impermissible.

VI. Conclusions and Future Directions

We presented IA schemes, a cryptographic primitive for image authentication, and constructed `PhotoProof`, an IA scheme based on Proof-Carrying Data and digital signatures. We also implemented a working prototype together with a collection of supported permissible transformations. Our implementation is the first proof-of-concept of an IA scheme.

Further improvements are required to make the technology usable for real world applications. This includes lowering generation and proving times, extending the set of supported transformations and raising the limit on image size. This may be achieved by faster SNARK technology that will be available in the future to prove larger predicates in less time, better circuit designs for image transformations that will lower the required constraint-per-pixel ratio for the compliance predicate, and accelerated implementations using GPGPU, FPGA or ASIC.

One could also implement a variant of PhotoProof, including its zero-knowledge guarantees, using a PCD-like mechanism based on trusted hardware with attestation capabilities such as TPM (also used by CertiPics; see Section I-B) or Intel's SGX. In this alternative implementation, every editing step attests for the correct execution of the computation that verified the previous step's attestation and did a permissible transformation. This would yield succinctness and zero-knowledge comparable to `PhotoProof`, with much higher performance, but based on trusted hardware and careful platform configuration, instead of cryptographic proofs.

Increased image size and decreased proof size will enable practical use of methods to embed the proof inside the image in an invisible way.

`PhotoProof` demonstrates the power of PCD in tracking and enforcing authenticity and provenance for digital images, while still offering the editing flexibility required by applications. Analogous needs for authenticity and provenance arise also for other document types, such as text (e.g., tracking citations), audio (e.g., proving authenticity of a recording), databases (e.g., tracking use of sensitive or unreliable information), and other structured data. We pose the challenge of identifying, and implementing, specific applications in these domains.

Acknowledgments

This work was supported by the Broadcom Foundation and Tel Aviv University Authentication Initiative; by the Check Point Institute for Information Security; by the Israeli Ministry of Science and Technology; by the Israeli Centers of Research Excellence I-CORE program (center 4/11); and by the Leona M. & Harry B. Helmsley Charitable Trust.

References

[1] M. Ajtai, "Generating hard instances of lattice problems," in *ACM Symposium on the Theory of Computing (STOC) 1996*, 1996, pp. 99–108.

[2] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk, "ADSNARK: Nearly practical and privacy-preserving proofs on authenticated data," in *IEEE Symposium on Security and Privacy (SP) 2015*, 2015, pp. 271–286.

[3] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin," in *IEEE Symposium on Security and Privacy 2014*, 2014, pp. 459–474.

[4] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *CRYPTO 2013*, 2013, pp. 90–108.

[5] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," Cryptology ePrint Archive, Report 2013/507, 2013, http://eprint.iacr.org/.

[6] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive arguments for a von Neumann architecture," Cryptology ePrint Archive, Report 2013/879, 2013.

[7] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," in *Advances in Cryptology–CRYPTO 2014*. Springer, 2014, pp. 276–294.

[8] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Innovations in Theoretical Computer Science (ITCS) 2012*, 2012, pp. 326–349.

[9] ——, "Recursive composition and bootstrapping for SNARKs and proof-carrying data," in *ACM Symposium on the Theory of Computing (STOC) 2013*, 2013, pp. 111–120.

[10] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, "Succinct non-interactive arguments via linear interactive proofs," in *Theory of Cryptography*. Springer, 2013, pp. 315–333.

[11] A. Chiesa and E. Tromer, "Proof-carrying data and hearsay arguments from signature cards." in *ICS*, vol. 10, 2010, pp. 310–331.

[12] ——, "Proof-carrying data: Secure computation on untrusted platforms," *The Next Wave: The National Security Agency's review of emerging technologies*, vol. 19, no. 2, pp. 40–46, 2012.

[13] A. Chiesa, E. Tromer, and M. Virza, "Cluster computing in zero knowledge," ser. EUROCRYPT '15, 2015, pp. 371–403.

[14] DARPA, "DARPA Broad Agency Announcement 15-58: Media Forensics (MediFor)," 2015, https://www.fbo.gov/index?s=opportunity&mode=form&id=bfa29e5f04566fbb961cd773a8a8649f&tab=core&_cview=1.

[15] Elcomsoft Co. Ltd., "Canon original data security system compromised: ElcomSoft discovers vulnerability," 2010. [Online]. Available: https://www.elcomsoft.com/PR/canon_101130_en.pdf

[16] H. Farid, "Image forgery detection," *Signal Processing Magazine, IEEE*, vol. 26, no. 2, pp. 16–25, 2009.

[17] W. Feng and Z.-Q. Liu, "Region-level image authentication using bayesian structural content abstraction," *IEEE Transactions on Image Processing*, vol. 17, no. 12, pp. 2413–2424, 2008.

[18] D. Fiore and A. Nitulescu, "On the (in)security of SNARKs in the presence of oracles," Cryptology ePrint Archive, Report 2016/112, 2016, http://eprint.iacr.org/.

[19] R. Frank, *Newslore: Contemporary folklore on the Internet*. Univ. Press of Mississippi, 2011, pp. 59–62.

[20] J. Fridrich and M. Goljan, "Robust hash functions for digital watermarking," in *International Conference on Information Technology: Coding and Computing (ITCC) 2000*. IEEE, 2000, pp. 178–183.
[21] G. L. Friedman, "The trustworthy digital camera: Restoring credibility to the photographic image," *IEEE Transactions on Consumer Electronics*, vol. 39, no. 4, pp. 905–910, 1993.
[22] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *EUROCRYPT 2013*, 2013, pp. 626–645.
[23] O. Goldreich, S. Goldwasser, and S. Halevi, "Collision-free hashing from lattice problems," Tech. Rep., 1996, eCCC TR95-042.
[24] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988.
[25] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT) 2010*, 2010, pp. 321–340.
[26] M. M. Hancock, "Ethics in the age of digital manipulation," 2009, http://globaljournalist.jour.missouri.edu/stories/2009/07/01/ethics-in-the-age-of-digital-manipulation.
[27] C. K. Ho and C.-T. Li, "Semi-fragile watermarking scheme for authentication of jpeg images," in *International Conference on Information Technology: Coding and Computing (ITCC)*, vol. 1. IEEE, 2004, pp. 7–11.
[28] C. W. Honsinger, P. W. Jones, M. Rabbani, and J. C. Stoffel, "Lossless recovery of an original image containing embedded data," Aug. 21 2001, US Patent 6,278,791.
[29] M. K. Johnson and H. Farid, "Exposing digital forgeries in complex lighting environments," *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 3, pp. 450–461, 2007.
[30] E. Kee and H. Farid, "Digital image authentication from thumbnails," in *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2010, pp. 75 410E–75 410E.
[31] J. Kelsey, B. Schneier, and C. Hall, "An authenticated camera," in *Computer Security Applications Conference*. IEEE Computer Society, 1996, pp. 24–24.
[32] C.-Y. Lin and S.-F. Chang, "Semifragile watermarking for authenticating JPEG visual content," in *Electronic Imaging*. International Society for Optics and Photonics, 2000, pp. 140–151.
[33] ——, "A robust image authentication method distinguishing jpeg compression from malicious manipulation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 2, pp. 153–168, 2001.
[34] Y.-C. Lin, D. Varodayan, and B. Girod, "Image authentication using distributed source coding," *IEEE Transactions on Image Processing*, vol. 21, no. 1, pp. 273–283, 2012.
[35] H. Lipmaa, "Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments," in *Theory of Cryptography*. Springer, 2012, pp. 169–189.
[36] E. C. Ltd., "Elcomsoft discovers vulnerability in Nikon's image authentication system," 2011. [Online]. Available: https://www.elcomsoft.com/PR/nikon_110428_en.pdf
[37] J. Lukas, J. Fridrich, and M. Goljan, "Digital camera identification from sensor pattern noise," *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 2, pp. 205–214, 2006.
[38] F. Lundh, A. Clark, and Secret Labs AB, "Pillow (PIL fork)," http://pillow.readthedocs.org.
[39] S. Micali, "Computationally sound proofs," *SIAM Journal on Computing*, vol. 30, no. 4, pp. 1253–1298, 2000, preliminary version appeared in FOCS 1994.
[40] Z. Ni, Y. Q. Shi, N. Ansari, W. Su, Q. Sun, and X. Lin, "Robust lossless image data hiding designed for semi-fragile image authentication," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 4, pp. 497–509, 2008.
[41] M. Nizza and P. J. Lyons, "In an iranian image, a missile too many," The Lede — The New York Times News Blog, 2008, http://thelede.blogs.nytimes.com/2008/07/10/in-an-iranian-image-a-missile-too-many.
[42] A. W. Paeth, "A fast algorithm for general raster rotation," in *Graphics Interface*, vol. 86, 1986, pp. 77–81.
[43] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *IEEE Symposium on Security and Privacy (Oakland) 2013*, 2013, pp. 238–252.
[44] V. Schetinger, M. M. Oliveira, R. da Silva, and T. J. Carvalho, "Humans are easily fooled by digital images," *arXiv preprint arXiv:1509.05301*, 2015.
[45] F. B. Schneider, K. Walsh, and E. G. Sirer, "Nexus authorization logic (NAL): Design rationale and applications," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 8, 2011.
[46] M. Schneider and S.-F. Chang, "A robust content based digital signature for image authentication," in *International Conference on Image Processing (ICIP) 1996*, vol. 3. IEEE, 1996, pp. 227–230.
[47] SCIPR Lab, "libsnark: a C++ library for zkSNARK proofs," https://github.com/scipr-lab/libsnark.
[48] J. S. Seo, J. Haitsma, T. Kalker, and C. D. Yoo, "A robust image fingerprinting system using the radon transform," *Signal Processing: Image Communication*, vol. 19, no. 4, pp. 325–339, 2004.
[49] E. G. Sirer, W. de Bruijn, P. Reynolds, A. Shieh, K. Walsh, D. Williams, and F. B. Schneider, "Logical attestation: an authorization architecture for trustworthy computing," in *ACM Symposium on Operating Systems Principles (SOSP) 2011*. ACM, 2011, pp. 249–264.
[50] D. Sklyarov, "Forging canon original decision data," Presented at CONFidence 2.0, 2010. [Online]. Available: https://www.elcomsoft.com/presentations/Forging_Canon_Original_Decision_Data.pdf
[51] D. Solo, R. Housley, and W. Ford, "Internet x. 509 public key infrastructure certificate and crl profile," 1999.
[52] R. Sun, H. Sun, and T. Yao, "A SVD-and quantization based semi-fragile watermarking technique for image authentication," in *International Conference on Signal Processing*, vol. 2. IEEE, 2002, pp. 1592–1595.
[53] N. Y. Times, "Guidelines On Integrity," New York Times, 2008, http://www.nytco.com/pdf/guidelines-on-integrity-2/.
[54] F. Van Riper, "Manipulating truth, losing credibility," The Washington Post, 2003, http://www.washingtonpost.com/wp-srv/photo/essays/vanRiper/030409.htm.
[55] R. Venkatesan, S.-M. Koon, M. H. Jakubowski, and P. Moulin, "Robust image hashing," in *International Conference on Image Processing (ICIP)*, vol. 3. IEEE, 2000, pp. 664–666.
[56] R. S. Wahby, S. Setty, Z. Ren, A. J. Blumberg, and M. Walfish, "Efficient RAM and control flow in verifiable outsourced computation," Cryptology ePrint 2014/674, Tech. Rep., 2015.
[57] K. A. Walsh, "Authorization and trust in software systems," Ph.D. dissertation, Cornell University, Ithaca, NY, USA, 2012.
[58] B. Warner, "Pure-Python ECDSA," https://pypi.python.org/pypi/ecdsa.
[59] H.-l. Zhang, C.-q. Xiong, and G.-z. Geng, "Content based image hashing robust to geometric transformations," in *International Symposium on Electronic Commerce and Security*, vol. 2. IEEE, 2009, pp. 105–108.
[60] Y. Zhao, S. Wang, X. Zhang, and H. Yao, "Robust hashing for image authentication using zernike moments and local features," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 55–63, 2013.

APPENDIX

Secure camera: caveats

Most existing image authentication solutions rely on a secure camera as a root of trust. Like any other secure device, the camera may be prone to attacks resulting from software and hardware vulnerabilities, side channel and fault injection attacks, and reverse engineering.

One example of this is Canon's *Original Decision Data (ODD)*, a feature in some of their high-end cameras which proves authenticity of images by digitally signing them inside the camera. Unfortunately, their implementation was insecure [50], [15]. The same holds for Nikon's analogous Image Authentication system [36].

Even when ignoring issues like implementation bugs and hardware flaws, there are several attack vectors at the camera level. One possible attack is *image injection*. An attacker can exploit the insecure link between a camera's sensor and its Image Signal Processor (ISP), by connecting physically to the ISP, transmitting raw data of an arbitrary image and retrieving

a digital signature for it. One way to prevent this is to encrypt the sensor-to-ISP channel. Another way is to program the ISP to sign only signals that bear a specific analog fingerprint unique to its matching sensor (e.g., its PRNU [37]). A third way is to use accelerometers to check whether the video feed just before taking the picture shows movements that match their reading, and only then sign.

Another attack is *2D scene staging*, discussed also in [21], [31]. An attacker can fabricate an arbitrary image, print or project it in high quality, and photograph it with a secure camera. In this case the output image will be signed and considered genuine, although it is merely a picture of a picture. One partial solution is to include some additional data in the image that will help determine whether the visual content of the image corresponds to the physical surroundings of the camera at the time of its capture. Examples of such data include the focus distance of the lens [21], or the range from target [31] (measurable with a laser beam). One more possibility is to check the timestamp information (already supported by our prototype) against external information about event timing. Another idea is for the camera to take a 3D picture (using 2 lenses and sensors) and determine whether the captured picture is of a 2D or 3D object, using (nontrivial) image processing algorithms. Finally, we note that a sufficiently dedicated attacker might precisely fabricate a 3D scene and photograph it in a studio — in which case it will be deemed authentic no matter what.