VerITAS: Verifying Image Transformations at Scale

Trisha Datta, Binyi Chen, Dan Boneh
Stanford University

***Abstract*—Verifying image provenance has become an important topic, especially in the realm of news media. To address this issue, the Coalition for Content Provenance and Authenticity (C2PA) developed a standard to verify image provenance that relies on digital signatures produced by cameras. However, photos are usually edited before being published, and a signature on an original photo cannot be verified given only the published edited image. In this work, we describe VerITAS, a system that uses zero-knowledge proofs (zk-SNARKs) to prove that only certain edits have been applied to a signed photo. While past work has created image editing proofs for photos, VerITAS is the first to do so for realistically large images (30 megapixels). Our key innovation enabling this leap is the design of a new proof system that enables proving knowledge of a valid signature on a large amount of witness data. We run experiments on realistically large images that are more than an order of magnitude larger than those tested in prior work. In the case of a computationally weak signer, such as a camera, we are able to generate a proof of valid edits for a 90 MB image in just over thirteen minutes, costing about $0.54 on AWS per image. In the case of a more powerful signer, we are able to generate a proof of valid edits for a 90 MB image in just over three minutes, costing only $0.13 on AWS per image. Either way, proof verification time is less than a second. Our techniques apply broadly whenever there is a need to prove that an efficient transformation was applied correctly to a large amount of signed private data.**

1. Introduction

Verifying where and when a digital image was taken has become increasingly difficult. After Russia invaded Ukraine in February 2022, several photographs and videos [1], [2], [3] circulated online that falsely claimed to show the conflict. In one instance, a BBC program showed footage of what was supposedly the Russian invasion of Ukraine, but was actually footage of a Russian military parade rehearsal [4]. The fact that even reputable news organizations like the BBC can make these mistakes demonstrates that there is much room for improvement in current image provenance verification processes.

The Coalition for Content Provenance and Authenticity (C2PA) has developed a standard [5] to verify image provenance that relies on digital signatures. This standard proposes that cameras digitally sign every photo they take along with the photo's metadata (e.g., location, timestamp, focal length, exposure time, etc.). Leica, Sony, and Nikon have all developed cameras with such signing capabilities [6], [7]. Leica has even developed an on-camera trusted execution environment (TEE) to protect the signing key. More recently, AI companies, such as OpenAI, have also begun issuing C2PA attestations on images that they generate to ensure that they are not falsely blamed for content that they did not generate. In Section 3 we discuss the threat model that the C2PA is designed to address.

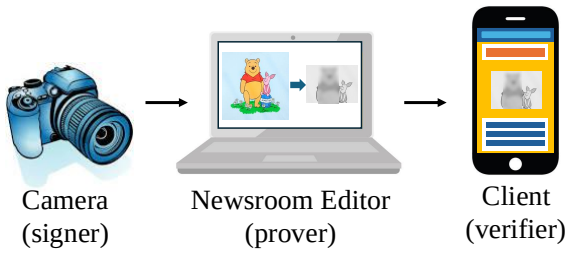

Figure 1: Image editing pipeline

Users could in theory verify the provenance of a photo in a news article by verifying the accompanying C2PA signature. However, photos are rarely published as is. Before being posted in a news story, they are often cropped, some faces and objects may be blurred to protect privacy, images are resized to save bandwidth, and in some cases they are converted to grayscale. The *Associated Press* published a list of acceptable edits [8] that do not fundamentally alter the content of the photo. See Figure 1 for an overview of the image editing pipeline.

Publishing edited photos presents a problem because the C2PA signature on the original image cannot be verified given only the edited image. To address this, the C2PA proposes that all edits be performed by a C2PA-enabled (and approved) editing application that maintains a secret signing key and signs the processed photo. These application-generated signatures will then be verified by the reader to validate the metadata of the photo. A major problem with this approach is that it changes the trust model and breaks end-to-end security. The end user must now trust the editing application and the security of its signing key. Moreover, it is not at all clear how open-source photo editing tools will be used in this context. These tools typically have no way to protect a signing key.

We therefore need a method for editing a signed photo such that a news consumer who only has the published edited photo can be assured that (i) the original unedited

photo was properly signed by a C2PA camera, (ii) only permissible edits, such as cropping, blurring, resizing, and grayscale, were made to the signed photo, and (iii) the metadata of the edited photo is equal to the metadata of the original photo. The scheme should preserve end-to-end security, from the camera to the user's screen, without requiring the user to trust some editing software, the article's publisher, or a third-party fact-checker. We call this property glass-to-glass security.

Our contributions. In this paper, we present VerITAS (Verifying Image Transformations At Scale), a system that uses succinct zero-knowledge arguments (zk-SNARKs) [9] to prove the provenance of edited photos. A zero-knowledge proof is a statement about a secret witness that can be verified by anyone without revealing anything about the witness other than the validity of the statement. These proofs are *complete*, meaning that verification will succeed for honestly generated proofs, and *knowledge sound*, meaning that verification will fail if the prover does not have a valid witness. These properties entail that the verifier need not trust the prover, which solves the trust problem posed by the C2PA protocol. Moreover, these proofs are *zero-knowledge*, meaning that the proof reveals nothing about elements that were removed from the original photo; this is vital when sensitive information is cropped or blurred. VerITAS uses succinct zero-knowledge arguments to enable the editor to make modifications to a captured C2PA image, and replace the signature with a zk-SNARK that the edited image was derived from a properly signed C2PA image via an authorized transformation. The resulting proofs are *succinct*, meaning that they are "short" and "fast" to verify. Proof verification can be done in the client application, which could automatically detect and verify these proofs in news articles.

In 2016, Naveh and Tromer [10] implemented PhotoProof, a system for producing zero-knowledge proofs for simple photo edits. While this work demonstrated the feasibility of creating zk-SNARKs for image edits, their proving time was too large to be practical. In concurrent work with ours, Kang et al. [11] developed a library that achieved a 100x speed-up over PhotoProof. However, the largest photo used in their experiments is 720p, or 900 kilopixels (KP). The pictures produced by the Sony and Leica cameras mentioned above are about 33 megapixels (MP), which is an order of magnitude larger than any photo used in previous work. VerITAS is the first system to produce ZK proofs of image edits for photos on the order of 30 MP or more. This paper is the full version of our work presented in [12] and [13].

We describe VerITAS as a protocol between a prover (a newsroom editor) and a verifier (a news consumer). Given a public edited image x (e.g., a photo in a news article) and a public editing function f, a prover convinces a verifier of the provenance of the published photo by proving that it knows a secret witness that comprises a photo w and a signature σ, such that (i) σ is a valid signature on w under a public verification key vk, and (ii) applying f to the witness photo w results in the public photo x. In other words, we need a zk-SNARK for the following instance-witness relation:

$$\mathcal{R} := \big\{ \big((\mathsf{vk}, f, x) \; ; \; (w, \sigma) \big) : \\ f(w) = x \quad \wedge \quad \text{SigVerify}(\mathsf{vk}, w, \sigma) = 1 \big\} \tag{1}$$

The witness (w, σ) provided as input to the zero-knowledge prover contains the original 30 megapixel (MP) image, which is about 90 MB, along with a signature on this image. Hence, in our settings, the prover must build a proof using an unusually large witness.

The main bottleneck in systems that use zk-SNARKs to prove simple photo edits is building a zk-SNARK for a circuit that verifies that the original image is properly signed. The difficult step is the first step of signature verification: proving that a 90 MB witness was hashed correctly with a collision resistant hash. Doing so using SHA256 is vastly inefficient because SHA256 employs many non-linear operations, which are expensive inside of a zk-SNARK circuit. Even proving knowledge of SNARK-friendly hashes like Poseidon, which is what Kang et al. [11] use, is too costly for hashing a 30 MP photo in a SNARK circuit, as discussed in Section 7.1.

Efficiently proving knowledge of a signature. VerITAS solves this problem by introducing two modes for proving knowledge of (w, σ) such that σ is a valid signature on w. One mode is designed for a computationally-limited signer (such as a camera); the other mode is designed for a more powerful signer (such an OpenAI). The former has a lightweight signing procedure but slower editing proof generation time. The latter has a more heavyweight signing procedure but a much faster editing proof generation time.

Mode 1. To accommodate a computationally limited signer (such as a camera), the VerITAS C2PA signer hashes the captured image using a lattice-based collision resistant hash to obtain a 1 KB digest. It then hashes that digest down to 32 bytes using Poseidon and signs the resulting hash value using its secret key. As observed in [14] and [15], the benefit of the lattice hash is that it uses only linear operations over a finite field. This makes the lattice hash far more amenable to being proved in a SNARK circuit than other collision resistant hash algorithms, especially for a large amount of data (such as a 90MB photo). We design a custom SNARK to show that a lattice hash has been computed correctly (see Section 5). We emphasize that no prior work has been able to create hashing proofs for 30 MP images, and has thus been limited to proving edits on photos that are an order of magnitude smaller than the size of photos captured by modern cameras.

To use our scheme, the C2PA camera would use our hash function to hash the captured image and then sign the computed hash using a standard signature scheme such as ECDSA. When editing the image, the newsroom would produce a proof that the original image is signed correctly by verifying the hash in the zk-SNARK circuit. This design may be of independent interest for anyone looking to create proofs for SNARK circuits that hash a large amount of data.

Mode 2. When the C2PA signer is computationally powerful (e.g., OpenAI), VerITAS uses a more heavyweight signing algorithm. Here the C2PA signer first computes a polynomial commitment to the captured photo and then signs the short polynomial commitment using, say, ECDSA. Computing a polynomial commitment takes more time and memory than computing a simple hash (we give detailed numbers in the evaluation section). The benefit is that now VerITAS can greatly reduce the time to generate a proof of valid edit. In particular, we modify how the zk-SNARK prover operates so that now the SNARK circuit only needs to verify the photo edits, *but does not need to verify the signature or the polynomial commitment*. Hence, by modifying the underlying proof system we obtain a massive savings for the newsroom editor when generating a proof of a valid edit. The details are provided in Section 4.1.

Implementation. We split our implementation of VerITAS into two parts: proving knowledge of a valid signature on the original photo and proving a valid edit of the original photo. We implement the former using the FRI [16] polynomial commitment scheme (PCS) implementation from Plonky2 [17], and we implement the latter by directly proving circuits with Plonky2. We report results for proof generation time, proof verification time, and peak memory usage. Our experiments show that proving knowledge of a signature using our Lattice-Poseidon hash for realistically-sized images is much faster compared to using a Poseidon hash alone. In fact, our machine could not prove knowledge of a Poseidon hash of even a one megapixel (1 MP) photo. If a photo is signed by a more powerful signer, as in mode 2, then an editor avoids proving knowledge of a signature altogether, and can just prove validity of the image edits, which takes less than ten minutes. However, for the signer, computing a polynomial commitment of a 30 MP photo requires more memory and computational power than a camera might have, which means that cameras will likely opt for the lattice hash method (mode 1).

To summarize, our contributions are threefold:

- VerITAS is the first system, to our knowledge, that can produce editing proofs for 30 MP signed images (all other work has been limited to proving edits on photos over an order of magnitude smaller);
- A custom proof system for computationally weak signers that can prove that the hash of a very large amount of witness data was computed correctly;
- A custom proof system for more powerful signers that enables editors to produce editing proofs without verifying a signature on the witness in the SNARK circuit. This greatly reduces the time to produce an editing proof.

Our techniques apply more broadly than images. They apply whenever there is a need to prove that an efficient transformation was applied correctly to a large amount of signed witness data. Some examples include signed financial or health records. However, our focus in this paper in on transformations applied to signed images.

Alternate designs. While VerITAS uses zk-SNARKs to support editing signed images, a very different approach is to use redactable signatures [18], [19], or more generally, homomorphic signatures [20], [21]. Homomorphic signatures enable anyone to transform a message-signature pair (m, σ) into another message-signature pair $\big(\big(f(m), f\big), \sigma'\big)$, where σ' is a valid signature on $\big(f(m), f\big)$. In other words, σ' is a signature on the transformed message m, and a description of the transformation function f. When f is a simple redaction operation, such as cropping, this can be implemented very efficiently using redactable signatures. However, more complicated transformations, such as blurring and resizing algorithms in image processing packages, cannot be reduced to redaction. For example, VerITAS provides a proof of correct resizing using the bilinear resizing algorithm [22], a standard resizing method in Adobe Photoshop, which uses linear transformations and cannot be reduced to redaction. Many other standard edits, such as brightness, contrast adjustments, tinting, dodging, and burning, can be proven in zero knowledge as in VerITAS, but cannot be done using redactable signatures. One could try to use homomorphic signatures [20], [21], but for these image transformations, the best homomorphic signatures that do not rely on SNARKs are impractical. We also note that for resizing, the camera does not know the resizing dimensions ahead of time and therefore cannot simply pre-sign a resized image.

2. Preliminaries

We use $[n]$ to denote the set $\{0, \ldots, n-1\}$ and use $[a, b]$ to denote the set $\{a, \ldots, b\}$. We use $\mathbb{F}_q$ to denote a finite field of size q. Let $r \leftarrow\!\!\$\ S$ denote drawing a random value from the finite set S. We let ω denote a primitive n^{th} root of unity in $\mathbb{F}$, so that the set $\Omega := \{1, \omega, ..., \omega^{n-1}\}$ has size n. We use $Z_\Omega \in \mathbb{F}[X]$ to denote the vanishing polynomial on Ω. This Z_Ω is the lowest-degree polynomial such that $Z_\Omega(x) = 0$ for all x in Ω. It has the form $Z_\Omega(X) = X^n - 1$, which can be evaluated using at most $2\log_2 n$ field multiplications. We use $\mathbb{F}^{<d}[X]$ to denote the set of all univariate polynomials of degree less than d over the field $\mathbb{F}$.

We use bold-faced lowercase letters for vectors. For a vector $\mathbf{v} \in \mathbb{F}^m$, we denote the elements of $\mathbf{v}$ as $(v_0, ..., v_{m-1})$. We write the concatenation of two vectors as $\mathbf{v}\|\mathbf{w}$. We denote matrices with bold-faced capital letters (e.g., $\mathbf{A} \in \mathbb{F}^{n \times m}$). We denote the columns of a matrix $\mathbf{A} \in \mathbb{F}^{n \times m}$ as $\mathbf{a}_0, ..., \mathbf{a}_{m-1} \in \mathbb{F}^n$. We denote element j of row i in matrix $\mathbf{A}$ as $A_{i,j}$ (e.g., the second element in the topmost row of $\mathbf{A}$ is $A_{0,1}$). We assume access to a hash function $\mathsf{H} : \mathbb{F}^* \to \mathbb{F}$ that can take as input any (finite) number of field elements as input.

2.1. Digital Signatures

A digital signature scheme $\mathcal{S}$ is a triple of efficient algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ such that:

- $\mathsf{KGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$, where sk is the secret signing key and vk is the public verification key.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$, where σ is a signature on message m.
- $\mathsf{Vf}(\mathsf{vk}, m, \sigma) \to 0/1$, where 0 implies rejection and 1 implies acceptance.

We say that a signature scheme is secure if it is existentially unforgeable under a chosen message attack [23] (see Appendix A.1 for the definition). Digital signatures in practice are implemented as a two step process: first hash the data using a collision-resistant hash and then sign the hash.

2.2. Commitment Schemes

A commitment scheme enables a party to commit to a value $x \in \mathcal{X}$ by producing a commitment string com. The commitment should be hiding and binding (see Appendix A.2 for definitions). More precisely, a commitment scheme $C = (\mathsf{setup}, \mathsf{commit})$ is a pair of PPT algorithms:

- $\mathsf{setup}(1^\lambda) \to \mathsf{pp}$, where pp are public parameters for the scheme
- $\mathsf{commit}(\mathsf{pp}, x, r) \to \mathsf{com}$, where com is a commitment to a message $x \in \mathcal{X}$ with randomness $r \in \mathcal{R}_\mathrm{C}$

To open the commitment com, the committer reveals x and r and the verifier accepts if $\mathsf{commit}(\mathsf{pp}, x, r) = \mathsf{com}$. In some cases the setup algorithm is trivial in which case we say that the commitment scheme is just the algorithm $\mathsf{commit}(x, r) \to \mathsf{com}$.

Polynomial commitments. A polynomial commitment scheme [24] lets a prover commit to a polynomial $f \in \mathbb{F}[X]$ of bounded degree d. Additionally, the committer can provide an evaluation proof for the committed polynomial at any point $x \in \mathbb{F}$. More precisely, a polynomial commitment scheme C is a tuple of four efficient algorithms $C = (\mathsf{setup}, \mathsf{commit}, \mathsf{open}, \mathsf{Vf})$ such that:

- $\mathsf{setup}(1^\lambda, d) \to \mathsf{pp}$, where pp are public parameters to commit to a polynomial of degree at most d.
- $\mathsf{commit}(\mathsf{pp}, f, r) \to \mathsf{com}$, where com is a commitment to a polynomial $f \in \mathbb{F}[X]$ of degree at most d using randomness $r \in \mathcal{R}_\mathrm{C}$.
- $\mathsf{open}(\mathsf{pp}, f, x, r) \to (\pi, y)$, where π is an opening proof that proves that $f(x) = y$.
- $\mathsf{Vf}(\mathsf{pp}, \mathsf{com}, x, y, \pi) \to 0/1$, where 0 implies rejection and 1 implies acceptance.

A polynomial commitment scheme must be correct, evalution binding, and optionally hiding. We defer these definitions to Appendix A.2. The polynomial commitment scheme used in our implementation is built from the Fast Reed Solomon IOP of Proximity (FRI IOPP) protocol [16] using a collision resistant hash function.

2.3. zk-SNARKs: Zero-Knowledge Succinct Arguments of Knowledge

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) are efficiently verifiable statements about a secret witness. A zk-SNARK Π for an instance-witness relation $\mathcal{R}$ is a tuple of PPT algorithms $\Pi = (\mathsf{setup}, \mathsf{prove}, \mathsf{Vf})$ such that for $(x, w) \in \mathcal{R}$:

- $\mathsf{setup}(1^\lambda) \to \mathsf{pp}$, where pp are public parameters.
- $\mathsf{prove}(\mathsf{pp}, x, w) \to \pi$, where, on input instance x and witness w, the proof π shows that $(x, w) \in \mathcal{R}$.
- $\mathsf{Vf}(\mathsf{pp}, x, \pi) \to 0/1$, where 0 implies rejection and 1 implies acceptance.

The zk-SNARK must be complete, knowledge sound, zero-knowledge, non-interactive, and succinct (see Appendix A.3 for definitions).

zk-SNARKs can be designed to prove a specific relation, or prove a general $\mathcal{NP}$ relation (e.g., Groth16 [25], PLONK [26]). We use both types in our system: we design a custom zk-SNARK for our lattice hash and use a general-purpose zk-SNARK for proving Poseidon hashes and photo transformations. Plonky2 [17] is a system that allows users to write constraints in the form of circuits that can then be proven and verified by the PLONK [26] polynomial interactive oracle proof (Poly-IOP) . We utilize PLONK to produce proofs for photo editing. For our scheme involving a computationally powerful signer, we make non-black-box use of PLONK.

2.3.1. Lookup Table Arguments. A lookup argument is a relation-specific zk-SNARK. Given a table $T \in \mathbb{F}^t$, a prover can use a lookup argument to show that all elements of some (committed) vector $\mathbf{v} \in \mathbb{F}^m$ are contained in T. Plookup [27], Baloo [28], cq [29], and Lasso [30] are state-of-the-art lookup arguments. Baloo and cq's prover complexity is sublinear in the table size t. However, compared to Plookup, their prover time grows faster in m (the dimension of $\mathbf{v}$). In our use of lookup tables we have $t \ll m$, and we therefore use a Plookup-based approach.

2.3.2. The Schwartz-Zippel Lemma. Let f be a non-zero ℓ-variate polynomial over a finite field $\mathbb{F}$, where the total degree of f is d. The Schwartz-Zippel Lemma [31], [32] says that for random elements $\alpha_1, \ldots \alpha_\ell \leftarrow\!\$\, \mathbb{F}$ we have

$$Pr\big[f(\alpha_1, \ldots, \alpha_\ell) = 0\big] \leq d/|\mathbb{F}|$$

We will use the this lemma to prove equality of polynomials.

2.3.3. Fiat-Shamir Transform. A public-coin interactive protocol can be made non-interactive using the Fiat-Shamir transform [33], which replaces verifier challenges with hashes of the transcript up until that point. For a protocol that has special soundness, applying the Fiat-Shamir transform retains its soundness properties [34].

2.3.4. PLONK. PLONK [26] is a zk-SNARK for proving correct evaluation of an arithmetic circuit. Figure 2 shows the (simplified) PLONK system design. For our photo editing proof that requires a computationally powerful signer, we modify the permutation argument in PLONK.

Let us briefly describe PLONK and its permutation argument. To prove correct evaluation of an arithmetic circuit we first build a table representing the computation trace

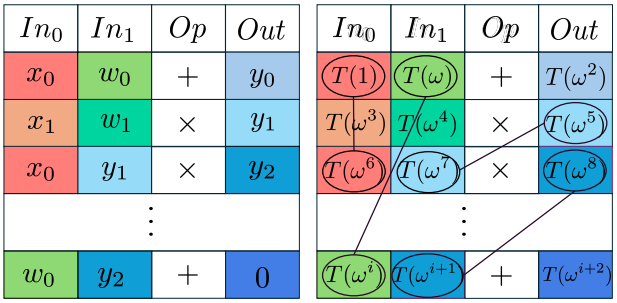

Figure 2: This figure illustrates how a circuit trace (left) is encoded as a polynomial $T(x)$ (right). The circled pairs on the right represent copy constraints (there are copy constraints between all cells of the same color); the PLONK prover must prove that the edge-connected cells have the same value and that the values of cells in each row satisfy the gate constraint. Recall that ω is a primitive n-th root of unity in $\mathbb{F}$, for a sufficiently large n.

(the left hand side of Figure 2). Every row of the trace corresponds to a single gate in the circuit. Each gate has two input wires $\mathsf{In}_0, \mathsf{In}_1$, one output wire Out, and an associated operation Op (here, either addition or multiplication). We require that $\mathsf{Op}(\mathsf{In}_0, \mathsf{In}_1) = \mathsf{Out}$ for all gates. This is called the *gate constraint*. Additionally, some input/output wires over different gates may be required to share the same values. This captures the wiring structure of the circuit and is called a *copy constraint*. For example, in the left hand side of Figure 2, the cells with same color must have identical values. Copy constraints are determined by the structure of the circuit and not by the wire values assigned by the prover. We categorize the set of wire values into three types: the public instance $\mathbf{x}$, the secret witness $\mathbf{w}$, and the internal wires $\mathbf{y}$. For example, in our application, $\mathbf{x}$ is the published edited photo, $\mathbf{w}$ is the original signed image (e.g. by the camera), and $\mathbf{y}$ is the intermediate values computed during the transformation from the original image $\mathbf{w}$ to the published image $\mathbf{x}$. In Figure 2, the public instance is $\mathbf{x} := (x_0, x_1)$, the secret witness is $\mathbf{w} := (w_0, w_1)$, and the internal wires are $\mathbf{y} := (y_0, y_1, y_2)$. Given a public instance $\mathbf{x}$, the prover needs to prove that it knows $\mathbf{w}$ and $\mathbf{y}$ that satisfy all the gate constraints and copy constraints.

PLONK uses a permutation argument to prove copy constraints. Let $T(x)$ be the polynomial that interpolates the trace values, i.e., for gate number i we have

$$T(\omega^{3i}) = \mathsf{In}_{i,0}, \quad T(\omega^{3i+1}) = \mathsf{In}_{i,1}, \quad T(\omega^{3i+2}) = \mathsf{Out}_i,$$

where $(\mathsf{In}_{i,0}, \mathsf{In}_{i,1}, \mathsf{Out}_i)$ are the wire values for gate i. Let $\tau : \Omega \to \Omega$ be a permutation such that for every copy constraint $T(s_1) = T(s_2) = \ldots = T(s_\ell)$ where $s_1, \ldots, s_\ell \in \Omega$, we have

$$\tau(s_1) = s_2, \quad \tau(s_2) = s_3, \quad \ldots, \quad \tau(s_\ell) = s_1.$$

We can represent τ as a polynomial of degree $n = |\Omega|$. It is clear that the copy constraints are satisfied if and only if $T(s) = T(\tau(s))$ for all $s \in \Omega$. The public statement in the PLONK permutation argument is a commitment to the polynomials T and τ. The argument proves that $T(s) = T(\tau(s))$ for all $s \in \Omega$. Moreover, the permutation argument supports proving permutation relations across multiple polynomials. That is, suppose we have polynomials $T_0, \ldots, T_{n-1}$ and a permutation $\tau : [n] \times \Omega \to [n] \times \Omega$. Then, we can use the permutation argument to prove that $T_i(s) = T_j(t)$ for all $s \in \Omega, i \in [n]$ where $\tau(i, s) = (j, t)$. This will be important in our photo editing proofs, as explained in Section 4.3. The details of the permutation argument can be found in [26], and several optimizations were proposed in [35], [36].

2.4. Short Integer Solution (SIS) and Lattice Hash

The Short Integer Solution (SIS) problem [37] is defined as follows. Fix some parameters $n, m, q, b \in \mathbb{N}$ where $n < m$ and q is a prime. An instance of the problem is specified by a random matrix $\mathbf{A} \leftarrow\$ \mathbb{F}_q^{n \times m}$. To solve the given SIS instance, the adversary must find a non-zero vector $\mathbf{v} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{v} = 0 \pmod q$ and $\|\mathbf{v}\|_\infty \leq b$ (i.e., $\mathbf{v}$ is short). For a sufficiently large $n \in \mathbb{N}$, solving SIS is conjectured to be hard for any choice of $m, q, b \in \mathbb{N}$ whenever $q > b \cdot \mathrm{poly}(n)$ and $m > n \log_2 q$.

We next describe a hash function whose collision resistance follows from the hardness of SIS [37], [38]. We represent the data to be hashed as a low-norm vector $\mathbf{v} \in \mathbb{Z}_q^m$. For a random matrix $\mathbf{A} \in \mathbb{F}_q^{n \times m}$, the hash function is defined as

$$H_{\mathbf{A}}(\mathbf{v}) := \mathbf{A}\mathbf{v} \pmod q \tag{2}$$

To see why this function is collision-resistant suppose towards a contradiction that there is an adversary $\mathcal{A}(\mathbf{A})$ that can find a collision for $H_{\mathbf{A}}$, when $\mathbf{A}$ is sampled as $\mathbf{A} \leftarrow\$ \mathbb{F}_q^{n \times m}$. Then $\mathcal{A}(\mathbf{A})$ will output low-norm distinct vectors $\mathbf{v}$ and $\mathbf{v}'$ in $\mathbb{Z}^m$, such that $\mathbf{A}\mathbf{v} = \mathbf{A}\mathbf{v}'$ in $\mathbb{F}_q$. But then $\mathbf{A}(\mathbf{v} - \mathbf{v}') = 0$, and since $\mathbf{v}$ and $\mathbf{v}'$ are low-norm, so is their difference. Hence, $\mathcal{A}$ can solve SIS. We stress that this shows that (2) is collision resistant only when $\mathbf{v}$ is low-norm.

3. Threat Model

In the C2PA setting, every camera is equipped with an embedded certified signing key for a secure signature scheme. The secret signing key is generated on camera and certified by a C2PA certificate authority at manufacturing time. Every time a camera takes a photo, it signs the raw RGB values of the photo's pixels and relevant metadata (e.g., location, timestamp, exposure time). We assume that the adversary has access to the camera and the camera's public key. However, the C2PA assumes that the attacker cannot extract the signing key from the camera, nor can the attacker cause the camera to sign an image that was not captured by the camera's optical hardware. In our settings, the only root of trust is the camera and its signing key. The editor of a photo is not trusted in any way. The verifier wants to ensure that the received edited photo is the result of applying an acceptable transformation [8] to a C2PA signed image.

Non threats. While C2PA is an important step towards image provenance, it is by no means a complete solution and must be combined with other defenses. Specific attacks on the C2PA are out of scope for this paper because our primary focus is on securing the image editing pipeline (Figure 1). Nevertheless, for completeness, we describe a few potential attacks on C2PA and how they might be addressed. These attacks are considered in the C2PA documentation.

First, the adversary might extract the C2PA signing key from some deployed camera. The Leica camera implements a hardware trusted execution environment (TEE) to protect the key and make extraction harder (but not impossible). Moreover, if a key is extracted, the standard includes a revocation mechanism that alerts all verifiers to revoke a compromised C2PA certificate.

Second, an attacker might try a picture-of-picture attack: it displays an AI-generated picture on a laptop screen and takes a picture of the screen using a C2PA camera. The result is a properly signed image of a fake event. This is a known challenge for the C2PA. One way to defend against this is to require the verifying client to run a picture-of-picture detector. For example, the focal length in the signed image metadata will be that of a camera taking a picture of a screen, and that is likely to be very different from the focal length needed to take a picture of the real-world portrayed event. Several other detection strategies have been suggested by C2PA, but this may turn into a cat-and-mouse game.

Third, the list of allowed transformations by the *Associated Press* may change the semantic meaning of the image. For instance, if presented with a photo of Alice and Bob, an adversary could crop out Alice and claim that Bob was alone when the photo was taken.

Fourth, C2PA may pose a privacy risk in that the signature on a photo can identify the camera that took it. This can be mitigated by having the camera sign photos using a group signature [39], [40]. We discuss this further in Section 9.

As explained above, these attacks are out of scope for this paper. Here we operate within the threat model that C2PA is designed to defend against — which excludes the attacks mentioned above — and focus on securing the editing pipeline.

4. The Design of VerITAS

We present VerITAS as an interaction between a news organization (prover) and a client reader (verifier). Every photo displayed in a news article should be accompanied by its metadata (location, timestamp, focal length, etc.), a description of the edits that were made to the original photo, and a succinct zero knowledge proof. The public statement consists of the published edited photo (x), the edits performed to the original photo (f), and the camera's public key (vk). The secret witness is the original photo (w) and the camera's signature (σ) on the original photo w. Recall that, abtractly, our goal is to design an efficient proof system for the instance-witness relation

$$\mathcal{R} := \big\{ \big((\mathsf{vk}, f, x) \; ; \; (w, \sigma) \big) \; : \\ f(w) = x \quad \wedge \quad \mathsf{Vf}(\mathsf{vk}, w, \sigma) = 1 \big\} \tag{3}$$

While (3) places vk in the public statement, we could protect the photographer's identity by moving vk and its certificate to the secret witness, and leaving only the CA public key in the public statement. To simplify the presentation we will use the relation (3) and discuss the more private variant in Section 9.

A client will only accept a photo's provenance if the photo x is accompanied by a triple (π, vk, f) where π is a valid ZK proof for $\mathcal{R}$, vk is a properly certified C2PA verification key, and the function f, which encodes the list of edits, is "acceptable," as defined by the Associated Press. We can thus think of VerITAS as enforcing a whitelist of allowable edits.

VerITAS relies on all the properties of a zk-SNARK. Completeness and knowledge-soundness of the zk-SNARK mean that the client does not need to trust the editor, preserving end-to-end security of the signature. Non-interactivity means that the news organization does not need to interact with any client and can instead publish a single proof along with the news article that any verifier can check. Zero-knowledge ensures that the proof does not reveal information that was cropped or blurred in the original photo. Succinctness ensures that the proof can be quickly verified by the client within a few seconds.

We next turn to designing a proof system for the relation $\mathcal{R}$ from (3). The proof has two parts:

- First, a proof that $f(w) = x$, for an image transformation function f. We come back to building such a proof in Section 6.2.
- Second, a proof that σ is a valid signature on w. This is more complicated, and we present our approach in Section 4.1.

We also need to ensure that the secret witness w used to generate both proofs are identical. We discuss how to achieve this in Section 6.1.2.

4.1. Proving Knowledge of a Valid Signature

When verifying a signature σ on some data w, the verifier: (i) computes $h \leftarrow \mathsf{H}(w)$, where H is a collision resistant hash function, and (ii) verifies that σ is a valid signature on h. When the data being verified is large, as in the case of a photo, most of the time is spent on computing the hash h in step (i). The same is true when proving knowledge of a valid signature. The challenge is to design an efficient SNARK circuit that can verify that the hash h of a large amount of data w is computed correctly. Once the verifier has a valid hash h, proving knowledge of an ECDSA signature on h can be done using existing circuits [41].

Ideally, we would like to use a standard hash function like SHA256. Unfortunately, proving that we have honestly applied SHA256 to a 30 megapixel (MP) witness is

practically infeasible. This is because SHA256 consists of mainly non-algebraic operations (e.g. logical operations), and proving non-algebraic constraints in a zk-SNARK is time-consuming. There are SNARK-friendly hash functions like Poseidon [42], but proving that we have honestly applied Poseidon to a 30 MP witness is also challenging in practice, as discussed in Section 7.1.

Our approach. We propose two solutions to this problem. Our first solution, presented in Section 4.2, is to design a collision resistant hash function H for which there is an especially efficient way to provide a SNARK proof for the instance-witness relation

$$\mathcal{R}_{\text{hash}} := \{ (h; w) \ : \ h = \mathsf{H}(w) \} \tag{4}$$

even when w is a large string. To do so we use a composition of a lattice-based hash function (see Section 2.4) and the Poseidon hash function. Using this hash function, in the context of a SNARK, is of independent interest.

Our second solution, presented in Section 4.3, uses a polynomial commitment scheme as the collision resistant hash H. Computing this hash function on the image w takes more computing resources than in our first solution. However, once the hash value is computed, incorporating it into a SNARK proof requires no additional work. Hence, this approach is suitable when the original photo signer has enough computing power to compute a polynomial commitment to the original photo. If so, then the editor's work to produce the proof-of-valid-edit completely eliminates the expensive step of producing a proof for the relation $\mathcal{R}_{\text{hash}}$.

4.2. Lattice + Poseidon Hash Function

The hash function used in our first solution is a sequential composition of the lattice hash from Section 2.4 and a Poseidon hash. We represent the photo w being hashed as three low-norm vectors, each containing either the R, G, or B (all 8 bits long) values for every pixel. This means that if a photo has m pixels, we transform it into three vectors $\mathbf{v}_r, \mathbf{v}_g, \mathbf{v}_b$ of length m whose values are all in $[0, 255]$. We hash these three vectors separately. We set q to be the prime field of the SNARK system used to prove knowledge of the signature (e.g., a 64 bit prime for a FRI-based SNARK). We then generate a random matrix $\mathbf{A} \in \mathbb{F}_q^{n \times m}$, where in our settings m is about thirty million and n is 128. Let $\mathsf{Poseidon}(x)$ be the function that applies the Poseidon hash function defined over $\mathbb{F}_q$ to the input x. We define our hash function H on input $\mathbf{v} \in \mathbb{F}_q^m$ as:

$$\mathsf{H}(\mathbf{v}) := \mathsf{Poseidon}(\mathbf{A} \cdot \mathbf{v} \bmod q) \tag{5}$$

as shown in Figure 3. We compute the Poseidon hash of the lattice hash because $\mathbf{Av}$ is still fairly large (1024 bytes) and the Poseidon hash is much smaller (32 bytes). The function in (5) is collision resistant because the composition of two collision resistant hash functions is also collision resistant.

The point is that computing $\mathsf{H}(\mathbf{v})$ requires mostly linear operations over $\mathbb{F}_q$. In particular, the large matrix-vector product is all linear, and by choosing q to be compatible

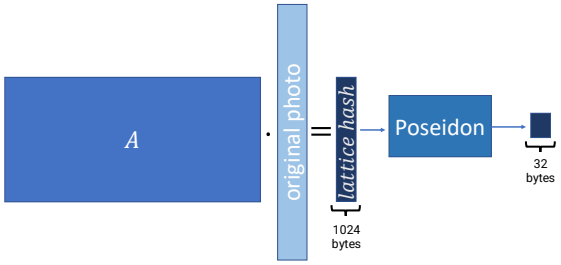

Figure 3: Our Lattice + Poseidon Hash Construction

with the SNARK field, we can prove these linear operations very efficiently in a zk-SNARK. The major challenge of this approach is that the lattice-based hash is only collision-resistant when the input is a low-norm vector, so the prover must additionally prove that $\mathbf{v}$ is low-norm. In other words, we need a proof system for the following relation:

$$\mathcal{R}_{\text{VH}} := \{ ((\mathbf{A}, \mathbf{h}, b) \ ; \ \mathbf{v}) \ : \ \mathbf{A} \in \mathbb{F}^{n \times m}, \ b \in \mathbb{N}, \ \mathbf{h} = \mathsf{Poseidon}(\mathbf{A} \cdot \mathbf{v}), \ \|\mathbf{v}\|_\infty < b \ \} \tag{6}$$

where b is a norm bound needed for collision resistance of the lattice hash (as in Section 2.4). This relation implies that (i) the prover has honestly calculated the product of the public matrix $\mathbf{A}$ and the secret vector $\mathbf{v}$, (ii) this vector $\mathbf{v}$ is low-norm, and (iii) the prover has honestly applied Poseidon to this product. To prove (iii), we can use an available Groth16 circuit for Poseidon [43]. Proving (i) and (ii) is the focus of Section 5.

4.3. A Polynomial Commitment Hash

We next describe our second approach for proving (3) that is designed for a signer that has more compute power (such as the owner of a generative AI model). Again, let us encode the original image w as three vectors $\mathbf{v}_r, \mathbf{v}_g, \mathbf{v}_b \in [0, 255]^m$. For simplicity, in this section we only consider one such vector, denoted by $\mathbf{v}$, which can be thought of as any of the three vectors. In our second approach (mode 2) the signer interpolates a univariate polynomial W, of degree at most $m-1$, such that $W(\omega^i) = v_i$ for $i = 1, \ldots, m$. We denote this polynomial by $\text{poly}(w)$, namely

$$W := \text{poly}(w).$$

Next, the signer computes a succinct polynomial commitment to $W(X)$ as

$$\mathsf{com} \leftarrow \mathsf{PCS.commit}(\mathsf{pp}, W, r)$$

where r is the commitment randomness chosen by the signer. The signer signs com to obtain a signature σ on com. It sends the image w along with $(\mathsf{vk}, \mathsf{com}, \sigma, r)$ to the news editor, where vk is the signer's certified public key.

The news editor will create an edited image $x := f(w)$. Because the polynomial commitment is binding, it suffices

for the editor to construct a zk-SNARK proof π for the following instance-witness relation

$$\mathcal{R}_{\text{simple}} := \big\{ \big((\mathsf{pp}, f, x, \mathsf{com}) \; ; \; (w, r) \big) \; : \; f(w) = x \; \wedge \; \mathsf{com} = \mathsf{PCS.commit}(\mathsf{pp}, \mathsf{poly}(w), r) \big\}$$

The editor then proves (3) by sending (vk, f, x) to the verifier along with the proof $\pi' := (\mathsf{com}, \sigma, \pi)$. The verifier checks that

- $\mathsf{Vf}(\mathsf{vk}, \mathsf{com}, \sigma)$ accepts (σ is a valid sig on com), and
- π is a valid proof that $(\mathsf{pp}, f, x, \mathsf{com})$ is a $\mathcal{R}_{\text{simple}}$ instance.

This proof system for (3) is knowledge sound and zero knowledge. Knowledge soundness follows because the proof system for $\mathcal{R}_{\text{simple}}$ is a zk-SNARK, which means we can extract a witness for $\mathcal{R}_{\text{simple}}$. This witness is by definition also a witness for (3). The proof system is zero-knowledge if we allow the simulator to take the public statement (vk, f, x) along with the signing key as input. The simulator then simulates the proof $\pi' := (\mathsf{com}, \sigma, \pi)$ using the simulator for the proof system for $\mathcal{R}_{\text{simple}}$ and the hiding propery of the polynomial commitment. Note that providing the simulator with the signing key does not leak information about the witness w because it is chosen before w. Alternatively, we can construct a simulator that only takes the public statement as input (without the signing key) by modifying the proof π' above to contain a zk-SNARK proof of a valid signature σ instead of the signature σ itself; see Section 9 for details.

The proof system. It remains to design a proof system for $\mathcal{R}_{\text{simple}}$. A naive way to do that is to have the SNARK circuit verify both that $f(w) = x$ and that com is a polynomial commitment to $\text{poly}(w)$. However, proving the latter would be far more expensive than proving that the editor has correctly hashed w, as we did in Section 4.2.

Instead, the key insight is that for the PLONK proof system, building a proof for $\mathcal{R}_{\text{simple}}$ is no more work for the editor than simply proving that $f(w) = x$. This means that the SNARK circuit never needs to hash w, which greatly reduces the work for the editor. To achieve this reduction in work, VerITAS modifies PLONK's permutation argument. Let us see how.

Let $\mathcal{C}(x, w)$ be a circuit that outputs 0 iff $f(w) = x$. Remarkably, if an editor wants to build a proof for $\mathcal{R}_{\text{simple}}$, it suffices to use PLONK to prove knowledge of a valid witness for $\mathcal{C}$; there is no need to expand $\mathcal{C}$ to also check that com is a polynomial commitment to $\text{poly}(w)$. Instead, we modify the PLONK prover to *indirectly* prove that com is a valid commitment to $\text{poly}(w)$. First, the editor builds the computation trace $T(X)$ for $\mathcal{C}(x, w)$ just as the standard PLONK prover would. The standard PLONK prover would then construct a proof π that $T(X)$ is a valid computation with respect to the gate constraints and copy constraints specified by $\mathcal{C}$. Our editor must additionally prove that com is a valid commitment to $\text{poly}(w)$. Recall that some entries in $T(\Omega)$ correspond to the witness w (see Figure 4 for an example). Thus, proving that com is a valid commitment to $\text{poly}(w) = W(X)$ is equivalent to proving that the witness elements represented by entries of $W(\Omega)$ are equal to the corresponding witness entries in $T(\Omega)$. The editor can prove this equality by extending the PLONK permutation argument.

Recall that the permutation argument proves that the vector $T(\Omega)$ is equal to the vector $T(\tau(\Omega))$, where τ is a polynomial that implements a permutation of Ω. The standard PLONK prover defines τ to capture copy constraints within the circuit $\mathcal{C}$ and then uses the permutation argument to prove that the computation trace respects the circuit wiring. In our scheme, the editor extends the PLONK permutation argument to prove that all the entries in $T(\Omega)$ that correspond to a witness element are equal to the corresponding witness element in $W(\Omega)$. More specifically, it extends τ to a new permutation τ' that captures additional copy constraints between $T(X)$ and $W(X)$. The right side of Figure 4 gives an example where the black edges represent the standard PLONK copy constraints, and the thick red edges represent the extended copy constraints between $T(\Omega)$ and $W(\Omega)$. In other words, we use the PLONK permutation argument to prove copy constraints across two different polynomials: $T(X)$ and $W(X)$. The permutation argument can be adapted to this task, as was already shown in the original PLONK paper [44, §5.1].

The news editor can use this to efficiently construct a proof that $\mathcal{C}(x, w) = 0$ and that com is a valid commitment to $\text{poly}(w)$. It uses both $T(X)$ and $W(X)$ to build a PLONK proof with respect to the permutation τ' which includes the new copy constraints shown in Figure 4. The result is a proof π that $(\mathsf{pp}, f, x, \mathsf{com})$ is a valid instance of $\mathcal{R}_{\text{simple}}$, as required.

Augmenting the permutation argument to operate over two polynomial T and W in this way does not change the proving time much over simply proving that $f(w) = x$ for a public value x and witness w. This is because the additional copy constraints are by far fewer than the copy constraints required for the circuit $\mathcal{C}$.

Summary. This method results in a massive reduction in work for the editor, because the zk-SNARK circuit is now much simpler than the circuit in Section 4.2. In particular, the circuit does not need to hash the large original image w. While this saves work for the editor, it creates more work for the signer because computing a polynomial commitment to w is more costly than computing a lattice hash of w. We quantify these tradeoffs in Section 7.

5. A Proof System for the Lattice Hash

In this section, we describe the custom proof system that VerITAS uses to prove that the hash function in (5) was computed correctly. Specifically, we construct a proof system for the following instance-witness relation

$$\mathcal{R}_{\text{LH}} := \big\{ \big((\mathbf{A}, \mathbf{h}, b) \; ; \; \mathbf{v} \big) \; : \; \mathbf{A} \in \mathbb{F}^{n \times m}, \; b \in \mathbb{N}, \; \mathbf{h} = \mathbf{A} \cdot \mathbf{v}, \; \|\mathbf{v}\|_\infty < b \big\} \tag{7}$$

That is, the prover shows that it knows a low-norm vector $\mathbf{v} \in \mathbb{F}^m$ such that $\mathbf{h} = \mathbf{A} \cdot \mathbf{v}$.

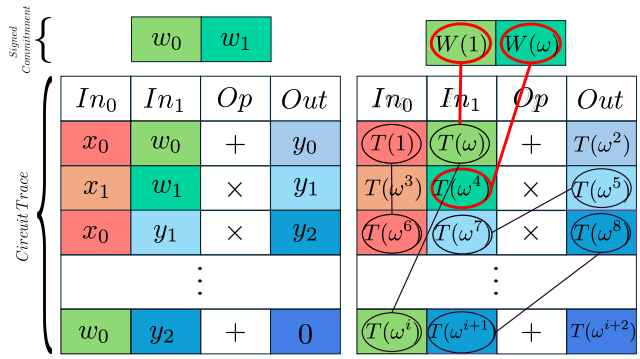

Figure 4: This figure illustrates the additional copy constraints (thick red lines) that the prover must prove in our polynomial commitment-based signing scheme. If $W(X)$ is the witness polynomial, then the prover must show that evaluations of W are the same as the corresponding witness cells in the circuit trace.

First, to prove that $\|\mathbf{v}\|_\infty < b$ it suffices to prove that all the elements of $\mathbf{v}$ are in the set* $[0, b-1]$. We can implement this range proof with a lookup argument using the table $T := [0, b-1]$. The lookup argument proves that every element of $\mathbf{v} \in \mathbb{F}^m$ is in T. In VerITAS, we set $b = 2^8$ and $m = 30,000,000$ (30 MP), so that $b \ll m$. In these settings a simplified Plookup-based lookup argument [45] minimizes prover work. Briefly, the argument works as follows: the prover commits to a vector $\mathbf{u} \in \mathbb{F}^b$, which is a list of the b values in $[0, b-1]$, and another vector $\mathbf{z} \in \mathbb{F}^{m+b}$, which is the sorted concatenation of $\mathbf{v}$ and $\mathbf{u}$. The prover then builds a zk-SNARK proof that (i) $\mathbf{z}$ is a permutation of $\mathbf{v}\|\mathbf{u}$, and (ii) that the difference between consecutive elements in $\mathbf{z}$ is either 0 or 1. Together, (i) and (ii) imply that all the elements of $\mathbf{v}$ lie in $[0, b-1]$, and the range proof is complete. We explain how to do this in Section 5.3.

Second, to prove $\mathbf{h} = \mathbf{A} \cdot \mathbf{v}$ we use the classic Freivalds' algorithm [46]. That is, we use the observation that to prove that $\mathbf{h} = \mathbf{A} \cdot \mathbf{v}$ it suffices to prove that $\mathbf{r}^\intercal\mathbf{h} = (\mathbf{r}^\intercal\mathbf{A})\mathbf{v}$ holds for a random vector $\mathbf{r} \leftarrow\$ \mathbb{F}^n$ chosen by the verifier. This collapses the matrix-vector product check to simply testing that the dot-product of $(\mathbf{r}^\intercal\mathbf{A})$ with $\mathbf{v}$ is equal to the public scalar $\mathbf{r}^\intercal\mathbf{h} \in \mathbb{F}$. This can be proved via the sum-check protocol, as observed by Thaler [47]. In our case we use a univariate sum-check proof introduced in the Aurora system [48]. We give the details in Section 5.4.

5.1. Polynomial Representation of Vectors

All of our subsequent proof systems prove statements about a witness vector $\mathbf{v} \in \mathbb{F}^m$. These protocols encode the vector $\mathbf{v}$ as a polynomial, and then prove the required statement about the polynomial. To do so, let us define the polynomial encoding for a vector $\mathbf{v}$ to be the unique polynomial $v(X) \in \mathbb{F}^{<m}[X]$ where $\forall i \in [m]$, $v(\omega^i) = v_i$. This means:

$$\mathbf{v} = (v_0, ..., v_{m-1}) = \big(v(1), ..., v(\omega^{m-1})\big) \in \mathbb{F}^m.$$

We let poly : $\mathbb{F}^m \to \mathbb{F}^{<m}[X]$ be the function that maps a vector $\mathbf{v}$ to a polynomial $v(X)$. This function can be implemented with any polynomial interpolation method.

*Technically, we need to prove that the elements are in $(-b, b)$, but since $\mathbf{v}$ only contains values in $[0, b)$, we can ignore the negative part.

5.2. Zero, Sum, and Permutation Check Proofs

In what follows we will be using proof systems for three well-known instance-witness relations: *ZeroCheck*, *SumCheck*, and *PermutationCheck*. As usual, let $\Omega_m := \{1, \omega, \omega^2, \ldots, \omega^{m-1}\} \subseteq \mathbb{F}$, and let $d > m$ be some degree bound.

- **The ZeroCheck relation:**

$$\mathcal{R}_{\text{ZC},m} := \big\{\big((\mathsf{pp}, \mathsf{com}_u)\ ;\ (u, r)\big)\ :\ u \in \mathbb{F}^{<d}[X],\ \forall \omega \in \Omega_m : u(\omega) = 0,\ \mathsf{commit}(\mathsf{pp}, u, r) = \mathsf{com}_u\ \big\}$$

- **The Univariate SumCheck relation:**

$$\mathcal{R}_{\text{SC},m} := \big\{\big((\mathsf{pp}, \mathsf{com}_u, s)\ ;\ (u, r)\big)\ :\ u \in \mathbb{F}^{<d}[X],\ \sum_{\omega \in \Omega_m} u(\omega) = s,\ \mathsf{commit}(\mathsf{pp}, u, r) = \mathsf{com}_u\ \big\}$$

- **The PermCheck relation:** Let $u \in \mathbb{F}^{<b}[X]$, $v \in \mathbb{F}^{<m}[X]$, and $z \in \mathbb{F}^{<b+m}[X]$ be three committed polynomials. A permutation check convinces the verifier that the vector $z(\Omega_{b+m})$ is a permutation of the vector $u(\Omega_b)\|v(\Omega_m)$. More precisely, it is a proof for the following relation

$$\begin{aligned}\mathcal{R}_{\text{PC}} :=\big\{\big((\mathsf{pp}, \mathsf{com}_u, \mathsf{com}_v, \mathsf{com}_z)\ ;\ (u, v, z, r_u, r_v, r_z)\big)\ :\ & u \in \mathbb{F}^{<b}[X],\quad v \in \mathbb{F}^{<m}[X],\quad z \in \mathbb{F}^{<b+m}[X],\\ & \prod_{\alpha \in \Omega_{b+m}} (X - z(\alpha)) = \prod_{\beta \in \Omega_b} (X - u(\beta)) \prod_{\gamma \in \Omega_m} (X - v(\gamma)),\\ & \mathsf{commit}(\mathsf{pp}, u, r_u) = \mathsf{com}_u,\\ & \mathsf{commit}(\mathsf{pp}, v, r_v) = \mathsf{com}_v,\\ & \mathsf{commit}(\mathsf{pp}, z, r_z) = \mathsf{com}_z\ \big\}\end{aligned}$$

  The equality on the third line is an equality of univariate polynomials in the indeterminate X. The equality holds if and only if z is a permutation of $u\|v$.

A zk-SNARK for the *ZeroCheck* relation works by (i) having the prover first commit to the quotient polynomial $q(X) := u(X)/Z_{\Omega_m}(X)$ and then (ii) proving that the polynomial equality $q(X) = u(X) \cdot Z_{\Omega_m}(X)$ holds, by proving that it holds at a random point in $\mathbb{F}$.

The Aurora proof system [48] gives a zk-SNARK for the *SumCheck* relation, and the Plonk system [26] gives zk-SNARK for the *PermCheck* relation.

We denote these three proof systems by $(\mathsf{P}_{\text{ZC}}, \mathsf{V}_{\text{ZC}})$, $(\mathsf{P}_{\text{SC}}, \mathsf{V}_{\text{SC}})$, and $(\mathsf{P}_{\text{PC}}, \mathsf{V}_{\text{PC}})$ respectively. All three proof systems produce proofs whose length is independent of the degree of the witness. Haböck [35] recently gave an

improved argument for *PermCheck*, by replacing the product by a sum of rational functions. *PermCheck* can also be proved efficiently using the GKR proof system [49].

5.3. The Range Proof

Next, we explain how to prove that all elements of a vector $\mathbf{v} \in \mathbb{F}^m$ are in a given set $T := [0, b-1]$. This range proof is inspired by the Plookup lookup table protocol [45]. Define the vector $\mathbf{u}_b := (0, 1, ..., b-1)$. A range proof amounts to proving that all elements of $\mathbf{v}$ are in $\mathbf{u}_b$.

We will work with the polynomial representation of these vectors, namely $v := \mathsf{poly}(\mathbf{v})$ and $u := \mathsf{poly}(\mathbf{u}_b)$. The verifier has the statement $(\mathsf{pp}, \mathsf{com}_v)$. The prover has the same statement along with a witness (v, r_v) such that $\mathsf{commit}(\mathsf{pp}, v, r_v) = \mathsf{com}_v$ and $v \in \mathbb{F}^{<m}[X]$. The range check proof system is described in Algorithms 1 and 2.

The public parameters pp for the proof system are generated using a one-time (possibly trusted) setup. These parameters are the public parameters for the zk-SNARKs $(\mathsf{P}_{\mathrm{ZC}}, \mathsf{V}_{\mathrm{ZC}})$ and $(\mathsf{P}_{\mathrm{PC}}, \mathsf{V}_{\mathrm{PC}})$, which include the public parameters for a polynomial commitment scheme (PCS). The degree bound d for the PCS is set to $m+b$, which is in turn determined by the size of the photos being processed.

Algorithm 1 $\mathsf{RangeCheckProver}(\mathsf{pp}, b, \mathsf{com}_v; v, r_v)$

$\mathbf{z} \leftarrow \mathsf{poly}(\mathsf{sort}(\mathbf{v} || \mathbf{u}_b))$
$\mathsf{com}_z \leftarrow \mathsf{commit}(\mathsf{pp}, \mathsf{poly}(\mathbf{z}), r_z)$
$\mathsf{com}_u \leftarrow \mathsf{commit}(\mathsf{pp}, \mathsf{poly}(\mathbf{u}_b), 0)$
// Do the permutation check on z and u, v.
$\pi_{\mathrm{PC}} \leftarrow \mathsf{P}_{\mathrm{PC}}(\mathsf{pp}, \mathsf{com}_u, \mathsf{com}_v, \mathsf{com}_z; u, v, z, 0, r_v, r_z)$
// Compute a polynomial f that is zero on Ω_{m+b} whenever
// the gap between consecutive elements of z is either 0 or 1.
$\lambda(X) \leftarrow \frac{\prod_{a \in \Omega_{m+b},\ a \neq 1}(X-a)}{\prod_{a \in \Omega_{m+b},\ a \neq 1}(1-a)}$ // Lagrange polynomial
$\mu(X) \leftarrow 1 - \lambda(X)$ // $\mu(\Omega_{m+b}) = 1$ except that $\mu(1) = 0$.
$f(X) \leftarrow \mu(X) \cdot \big(z(\omega X) - z(X)\big) \cdot \big(z(\omega X) - z(X) - 1\big)$
// The commitment com_z implies a commitment com_f to f meaning that:
// opening $f(X)$ at x can be done by opening $z(X)$ at x and ωx.
// Prove that f is zero on Ω_{m+b}.
$\pi_{\mathrm{ZC}} \leftarrow \mathsf{P}_{\mathrm{ZC,m+b}}(\mathsf{pp}, \mathsf{com}_f; f, r_f)$
Output $\pi \leftarrow (\mathsf{com}_z, \pi_{\mathrm{PC}}, \pi_{\mathrm{ZC}})$

Algorithm 2 $\mathsf{RangeCheckVerifier}(\mathsf{pp}, b, \mathsf{com}_v; \pi)$

parse $(\mathsf{com}_z, \pi_{\mathrm{PC}}, \pi_{\mathrm{ZC}}) \leftarrow \pi$
// The commitment com_z implies a commitment com_f to f as in Alg. 1.
$\mathsf{com}_u \leftarrow \mathsf{commit}(\mathsf{pp}, \mathsf{poly}(\mathbf{u}_b), 0)$
accept if $\mathsf{V}_{\mathrm{PC}}(\mathsf{pp}, \mathsf{com}_u, \mathsf{com}_v, \mathsf{com}_z; \pi_{\mathrm{PC}})$
and $\mathsf{V}_{\mathrm{ZC,m+b}}(\mathsf{pp}, \mathsf{com}_f; \pi_{\mathrm{ZC}})$ both accept

The following theorem states the security property of this proof system.

Theorem 5.1. Suppose that $(\mathsf{P}_{\mathrm{ZC}}, \mathsf{V}_{\mathrm{ZC}})$ and $(\mathsf{P}_{\mathrm{PC}}, \mathsf{V}_{\mathrm{PC}})$ are zk-SNARKs for $\mathcal{R}_{\mathrm{ZC}}$ and $\mathcal{R}_{\mathrm{PC}}$ respectively. Further, suppose that the polynomial commitment scheme used is secure and unconditionally hiding. Then the proof system in Algorithms 1 and 2 is a zk-SNARK for the relation

$$\mathcal{R}_{\mathrm{RP}} := \Big\{ \big((\mathsf{pp}, b, \mathsf{com}_v)\ ;\ (v, r_v)\big)\ :\ v \in \mathbb{F}^{<m}[X],\quad b \in [d-m],\quad \forall \omega \in \Omega_m : v(\omega) \in [0, b-1],\quad \mathsf{com}_v = \mathsf{commit}(\mathsf{pp}, v, r_v) \Big\}$$

where d is the degree bound used when generating pp.

Completeness is immediate. We briefly outline the argument for why the proof system is zero-knowledge and knowledge soundness.

Zero-Knowledge: We construct a PPT simulator $\mathsf{sim}_{\mathrm{RC}}$ that simulates an accepting transcript for all x in the language specified by $\mathcal{R}_{\mathrm{RP}}$. Let $\mathsf{sim}_{\mathrm{ZC}}$ and $\mathsf{sim}_{\mathrm{PC}}$ be the simulators for $(\mathsf{P}_{\mathrm{ZC}}, \mathsf{V}_{\mathrm{ZC}})$ and $(\mathsf{P}_{\mathrm{PC}}, \mathsf{V}_{\mathrm{PC}})$ respectively.

$\mathsf{sim}_{\mathrm{RC}}$ takes as input pp and $x = (b, \mathsf{com}_v)$. It computes $\mathsf{com}_u \leftarrow \mathsf{commit}(\mathsf{pp}, \mathsf{poly}(\mathbf{u}_b), 0)$ and sets com_z to be a commitment to the zero polynomial using some randomness r. By the unconditional hiding property of the PCS, com_z is sampled from a statistically close distribution to the real commitment to z. Next $\mathsf{sim}_{\mathrm{RC}}$ runs $\mathsf{sim}_{\mathrm{PC}}(\mathsf{pp}, (\mathsf{com}_u, \mathsf{com}_v, \mathsf{com}_z))$ to get π'_{PC}. By the zero-knowledge property of $(\mathsf{P}_{\mathrm{PC}}, \mathsf{V}_{\mathrm{PC}})$, π'_{PC} will be indistinguishable from a proof produced by an honest prover. Similarly, the simulator runs $\mathsf{sim}_{\mathrm{ZC}}(\mathsf{pp}, \mathsf{com}_f)$ and obtains π_{ZC}. By the zero-knowledge property of $(\mathsf{P}_{\mathrm{ZC}}, \mathsf{V}_{\mathrm{ZC}})$, the proof π_{ZC} is indistinguishable from a proof produced by an honest prover. Putting it all together, the proof $(\mathsf{com}_z, \pi_{\mathrm{PC}}, \pi_{\mathrm{ZC}})$ is indistinguishable from a proof produced by an honest prover.

Knowledge Soundness: Let $\mathcal{A}$ be a prover that outputs a valid proof $\pi = (\mathsf{com}_z, \pi_{\mathrm{PC}}, \pi_{\mathrm{ZC}})$ given pp and the statement $x = (b, \mathsf{com}_v)$ as input. We construct an extractor $\mathcal{E}_{\mathrm{RP}}$ that outputs a valid $\mathcal{R}_{\mathrm{RP}}$ witness (v, r_v) for x. Our $\mathcal{E}_{\mathrm{RP}}$ first computes $\mathsf{com}_u \leftarrow \mathsf{commit}(\mathsf{pp}, \mathsf{poly}(\mathbf{u}_b), 0)$. It then runs the extractor $\mathcal{E}^{\mathcal{A}}_{\mathrm{PC}}(\mathsf{pp}, (\mathsf{com}_u, \mathsf{com}_v, \mathsf{com}_z))$ to extract $(u', v', z', 0, r_{v'}, r_{z'})$. By the definition of $\mathcal{R}_{\mathrm{PC}}$, com_z is a commitment to z'. As noted in Algorithm 1, this means that $\mathcal{E}_{\mathrm{RP}}$ can derive a commitment com_f to $f(X) = \mu(X) \cdot \big(z'(\omega X) - z'(X)\big) \cdot \big(z'(\omega X) - z'(X) - 1\big)$ such that $\mathsf{V}_{\mathrm{ZC,m+b}}(\mathsf{pp}, \mathsf{com}_f; \pi_{\mathrm{ZC}})$ accepts. $\mathcal{E}_{\mathrm{RP}}$ can therefore run $\mathcal{E}^{\mathcal{A}}_{\mathrm{ZC}}(\mathsf{pp}, (\mathsf{com}_f))$ to extract $(f', r_{f'})$. By the definition of $\mathcal{R}_{\mathrm{ZC}}$, com_f is a commitment to f', so $f' = f$, or we have broken the binding property of the commitment scheme.

We claim that $(x, (v', r_{v'})) \in \mathcal{R}_{\mathrm{RP}}$. By the definition of $\mathcal{R}_{\mathrm{PC}}$, $\mathsf{com}_v = \mathsf{commit}(\mathsf{pp}, v', r_{v'})$. We must additionally show that $\forall \omega \in \Omega_m, v'(\omega) \in [0, b-1]$. We explained in the introduction to this section that this is equivalent to showing that (i) the vector $z'(\Omega_{b+m})$ is a permutation of the vector $u'(\Omega_b) \| v'(\Omega_m)$ and that (ii) the difference between consecutive elements of $z'(\Omega_{b+m})$ is either 0 or 1. (i) is true from the definition of $\mathcal{R}_{\mathrm{PC}}$. We now show that (ii) also holds. By the definition of $\mathcal{R}_{\mathrm{ZC}}$, $f'(\omega) = f(\omega) = 0$ for all $\omega \in \Omega_{b+m}$. Examining the definition of f, we see that this directly implies that all $z'(\omega x) - z'(x) \in \{0, 1\}$ for all $x \in \Omega_{b+m} \setminus \{1\}$, which means that the difference between consecutive elements of $z'(\Omega_{b+m})$ is either 0 or 1.

5.4. The Lattice Hash Proof

Finally, we show a proof system that lets the prover show that, given a lattice hash $\mathbf{h} \in \mathbb{F}^n$, it knows a low-norm preimage $\mathbf{v} \in \mathbb{F}^m$. That is, we provide a proof system for the relation $\mathcal{R}_{\text{LH}}$ from (7), as required.

First, we augment the relation $\mathcal{R}_{\text{LH}}$ as follows

$$\begin{aligned}\mathcal{R}'_{\text{LH}} := \big\{ \big((\mathbf{A},&\mathbf{h}, b, \mathsf{pp}, \mathsf{com}_v)\ ;\ (\mathbf{v}, r_v)\big)\ :\\ &\mathbf{A} \in \mathbb{F}^{n\times m}, \quad \mathbf{v} \in \mathbb{F}^m, \quad b \in [d-m],\\ &\mathbf{h} = \mathbf{A}\cdot\mathbf{v}, \quad \|\mathbf{v}\|_\infty < b,\\ &\mathsf{com}_v = \mathsf{commit}(\mathsf{pp}, \mathrm{poly}(\mathbf{v}), r_v) \quad \big\}\end{aligned} \tag{8}$$

This relation is the same as $\mathcal{R}_{\text{LH}}$ except that we force the prover to send to the verifier a commitment com_v to $\mathbf{v}$. Observe that the only difference between this relation and the relation $\mathcal{R}_{\text{RP}}$ from Theorem 5.1 is the additional constraint that $\mathbf{h} = \mathbf{A}\cdot\mathbf{v}$. We explained at the beginning of Section 5 that this constraint can be reduced to checking a single dot-product by taking a random linear combination of the rows of $\mathbf{A}$. The random linear combination is provided by the public coin of the verifier, and the protocol can be made non-interactive using the Fiat-Shamir transform. Finally, this single dot-product is exactly a univariate SumCheck relation, and can be verified by a single univariate SumCheck proof. Hence, a proof system for $\mathcal{R}'_{\text{LH}}$ uses the proof system from Section 5.3 along with a univariate SumCheck proof.

The lattice hash proof system is described in Algorithms 3 and 4. We preset the non-interactive versions of the protocols by using a random oracle H in place of the verifier's public coin.

Algorithm 3 $\mathsf{LatticeHashProver}(\mathbf{A}, \mathbf{h}, b, \mathsf{pp}, \mathsf{com}_v; \mathbf{v}, r_v)$

// Do the range proof on $\mathbf{v}$ using Section 5.3.
$\pi_{\text{RP}} \leftarrow \mathsf{P}_{\text{RP}}(\mathsf{pp}, b, \mathsf{com}_v; \mathrm{poly}(\mathbf{z}), r_v)$
// Compute the challenge using a Fiat-Shamir random oracle H
$\mathbf{r} \leftarrow \mathsf{H}(\mathbf{A}, \mathbf{h}, \mathsf{pp}, \mathsf{com}_v, \pi_{\text{RP}}) \in \mathbb{F}^n$
// Prove $\mathbf{r}^\intercal\mathbf{h} = (\mathbf{r}^\intercal\mathbf{A})\mathbf{v}$ holds for the challenge vector $\mathbf{r}$
$\mathbf{v}_{A,r} \leftarrow \big((\mathbf{r}^\intercal\mathbf{a}_0)v_0, \ldots, (\mathbf{r}^\intercal\mathbf{a}_{m-1})v_{m-1}\big) \in \mathbb{F}^m$
$v_{A,r}(X) \leftarrow \mathrm{poly}(\mathbf{v}_{A,r})$
$h \leftarrow \mathbf{r}^\intercal\mathbf{h} \in \mathbb{F}$
// $\mathbf{A}$, $\mathbf{r}$, and com_v imply a commitment $\mathsf{com}_{v_{A,r}}$ to $v_{A,r}(X)$
// Compute a SumCheck proof on $v_{A,r}(X)$
$\pi_{\text{SC}} \leftarrow \mathsf{P}_{\text{SC,m}}(\mathsf{pp}, \mathsf{com}_{v_{A,r}}, h)$
Output $\pi \leftarrow (\pi_{\text{RP}}, \pi_{\text{SC}})$

Algorithm 4 $\mathsf{LatticeHashVerifier}(\mathbf{A}, \mathbf{h}, b, \mathsf{pp}, \mathsf{com}_v; \pi)$

parse $(\pi_{\text{RP}}, \pi_{\text{SC}}) \leftarrow \pi$
$\mathbf{r} \leftarrow \mathsf{H}(\mathbf{A}, \mathbf{h}, \mathsf{pp}, \mathsf{com}_v, \pi_{\text{RP}}) \in \mathbb{F}^n$
$h \leftarrow \mathbf{r}^\intercal\mathbf{h} \in \mathbb{F}$
// $\mathbf{A}$, $\mathbf{r}$, and com_v imply a commitment $\mathsf{com}_{v_{A,r}}$ to $v_{A,r}(X)$
accept if $\mathsf{V}_{\text{RP}}(\mathsf{pp}, b, \mathsf{com}_v; \pi_{\text{RP}})$
and $\mathsf{V}_{\text{SC,m}}(\mathsf{pp}, \mathsf{com}_{v_{A,r}}, h; \pi_{\text{SC}})$ both accept

Theorem 5.2. Suppose that $(\mathsf{P}_{\text{RP}}, \mathsf{V}_{\text{RP}})$ and $(\mathsf{P}_{\text{SC}}, \mathsf{V}_{\text{SC}})$ are zk-SNARKs for $\mathcal{R}_{\text{RP}}$ and $\mathcal{R}_{\text{SC}}$ respectively. Then the proof system in Algorithms 3 and 4 is a zk-SNARK for the relation $\mathcal{R}'_{\text{LH}}$.

Completeness is immediate. Knowledge soundness follows from the knowledge soundness of $(\mathsf{P}_{\text{RP}}, \mathsf{V}_{\text{RP}})$, $(\mathsf{P}_{\text{SC}}, \mathsf{V}_{\text{SC}})$, and Freivalds' algorithm. Zero-knowledge follows from the zero-knowledge of $(\mathsf{P}_{\text{RP}}, \mathsf{V}_{\text{RP}})$ and $(\mathsf{P}_{\text{SC}}, \mathsf{V}_{\text{SC}})$ and from the fact that we can program the random oracle H.

6. VerITAS Implementation Details

We implement the two components of VerITAS separately: we use the FRI-PCS from the Plonky2 [17] library to generate proofs of correct hashing (for the relation $\mathcal{R}_{\text{VH}}$ from (6)), and we use Plonky2 to generate the photo editing proofs.[†] In addition, the editor would generate a proof of knowledge of a valid ECDSA signature on the Lattice+Poseidon hash using an existing signature checking circuit [41] on top of our proof system, which only adds 45 seconds to proof generation time.

6.1. Implementing a Proof System for $\mathcal{R}_{\text{VH}}$

To implement our proof system for the relation $\mathcal{R}_{\text{VH}}$ from (6), we use the FRI-based polynomial commitment scheme [16] implementation in the Plonky2 library [17]. This implementation allows us to batch commit to polynomials and to batch open these commitments at multiple points.

To prevent the prover and verifier from having to store all the elements in the (large) hashing matrix $\mathbf{A}$, we generate the entries of $\mathbf{A}$ using the upper 32 bits of a linear congruential generator [50] with a 64-bit modulus q'. For our SIS parameters, we set $n = 128$, and $b = 256$. As discussed in Section 4.2, the choice of q depends on the polynomial commitment scheme used. FRI uses a 64 bit prime. If q is 64 bits, the SIS lattice estimator calculator for these parameters gives 192 bits of security [51]. The prover generates the random Fiat-Shamir challenge for the permutation argument by taking the hash of the transcript thus far. In addition to proving knowledge of a lattice hash $\mathbf{Av}$ as described in Section 5, the VerITAS prover must also prove that applying a Poseidon hash to this lattice hash results in the final public hash $\mathbf{h}$. This can be done using another Plonky2 circuit.

6.1.1. Optimized $(\mathbf{r}^\intercal\mathbf{A})$ Derivation. Recall that in Section 5, we use the Freivalds' algorithm to reduce the checking of the matrix vector product $\mathbf{h} = \mathbf{A}\cdot\mathbf{v}$ to the dot-product of $(\mathbf{r}^\intercal\mathbf{A})$ with $\mathbf{v}$, where $\mathbf{r}$ is a random vector. The most time-intensive part for the verifier is to rederive $\mathbf{r}^\intercal\mathbf{A}$—the random linear combination of $\mathbf{A}$'s rows. To reduce verifying time, we implement opt-VeriTAS where we assume the existence of public trusted commitments to the rows of $\mathbf{A}$. These commitments can be generated in a preprocessing phase and used for every proof thereafter. Given the trusted (polynomial) commitments, the prover provides an opening

[†]Our code available at https://github.com/zk-VerITAS/VerITAS

proof for each row polynomial at a random point α. The verifier can then get the evaluation of poly($\mathbf{r}^\intercal \mathbf{A}$) at α and verify the proof. The cost is a roughly a factor of two increase in prover time, and a factor of six increase in proof size, for a roughly 20 times reduction in the verifier's time.

6.1.2. Consistency with Photo Editing Proofs. Both the SNARK circuits for the hash proof and the photo editing proof take the original photo w as part of the secret witness. However, a malicious prover might assign different values for the original photo in these two circuits. To prevent this, we leverage the fact that both proofs use polynomial commitments. The idea is to require the prover to provide a polynomial commitment com to the original photo. Then by the technique described in Section 4.3, we can ensure that the vector committed in com is consistent with the partial witness used in both the hash and the photo editing circuits.

6.2. Photo Editing Proof Implementation

We generate the photo editing proofs using Plonky2 [17], a Rust-based general-purpose zk-SNARK system. Plonky2 lets developers specify a circuit, and use PLONK to prove that, given some public instance and private witness as input, the circuit output equals a certain value. For every edit we want to prove, we construct a circuit that applies the edit on the private witness (the original photo) and outputs the result. The verifier can then check that this output is the same as the public instance (the published edited photo).

Our cropping circuit computes the cropped photo by outputting the RGB values of the original photo in the cropped range.

Our grayscale circuit applies the standard grayscale formula used by Adobe Photoshop [52] to the RGB values of the original photo. This formula obtains the gray value $gray$ for a pixel in the edited image by taking a weighted linear combination of the RGB values in the original image: $gray = round(0.30R + 0.59G + 0.11B)$. We perform this transformation using fixed-point arithmetic. This means that we scale the RGB values by a factor of 100, calculate the weighted linear combination from above, and then round to the nearest multiple of 100 to get the value for $gray$. We represent this transformation with the following equation: $100 \cdot gray = 30R + 59G + 11B + rem$ where rem is the remainder from rounding, which is required to be in the range of $[-49, 50]$. Thus, the prover must prove knowledge of some $rem \in [-49, 50]$ such that the second equation holds for the original RGB values and the $gray$ value in the edited photo. Because the purpose of grayscale conversion is not to conceal some part of the photo, the remainders do not leak any additional information, so our prover includes the remainders associated with the rounding calculations as part of the statement. The verifier can check them manually.

Our resizing circuit implements bilinear resizing, which is one of the standard resizing options offered in Adobe Photoshop [22]. Bilinear resizing calculates the RGB values for every pixel in the resized image by taking a weighted linear combination of the RGB values of four pixels in the original image. Just as with the grayscale circuit, we accommodate floating point arithmetic by passing remainders to the verifier. Just as with grayscale conversion, the prover uses fixed-point arithmetic and includes the remainders involved in rounding calculations as part of the statement.

Our blur circuit implements a box blur, which is one of the standard blur options offered in Adobe Photoshop [53]. A box blur calculates the RGB values for a pixel at position (i, j) by averaging the RGB values of the pixels in the 3x3 "box" in the original image where pixel (i, j) is at the center. Just as in the grayscale and resizing circuits, the box blur calculation involves fixed-point arithmetic. However, because the purpose of blurring is to obscure information, including remainders for the blurred region in the statement may potentially leak sensitive information to the verifier. Instead, we check within the photo editing circuit that the remainders are in the range $[0, 8]$. These range proofs cause the longer proving times for blurring compared to the proving times for other edits.

7. Experimental Results

We report proof generation time, verification time, and proof size for the hash relation $\mathcal{R}_{\text{VH}}$ from (6) and for the image editing relations. We report generation times for both non-ZK and ZK proof generation. We ran our timing experiments on randomly-generated RGB channels on a virtual machine with 131 GB of RAM and 12 CPU cores. When considering what is a reasonable amount of time to generate and verify proofs, it is important to remember that proof generation only needs to happen once per photo, while proof verification needs to be performed by every client that accesses the article. This means that while proof generation needs to be fast, proof verification needs to be very fast. Given peak memory usage and running time, we estimate that generating a non-ZK proof for the hash relation for a single RGB channel would cost about \$0.41 on AWS per image for VerITAS and \$1.46 for opt-VerITAS, and generating a ZK proof for the hash relation would cost about \$0.82 on AWS per image for VerITAS and \$1.80 for opt-VerITAS. Non-ZK proofs for the editing relations would add a maximum of \$0.13 per edit for to the cost, while ZK proofs for the editing relations would add a maximum of \$0.18 per edit to the cost. Recall that the polynomial commitment hash method described in Section 4.3 only needs a proof for the editing relation; there is no need to prove the hash relation.

7.1. $\mathcal{R}_{\text{VH}}$ Proof Generation Results

Figure 5 compares the proving times for proving knowledge of a Poseidon hash using the arkworks [54] Rust library and proving knowledge of a lattice hash using our FRI-PCS implementations of VerITAS and opt-VerITAS. The FRI-PCS has both blinding and non-blinding implementations; we report timing results using both modes. Because Plonky2 has not been optimized for zero-knowledge, the blinding mode numbers are pessimistic. We assume that the hashes for the RGB vectors $\mathbf{v}_r$, $\mathbf{v}_g$, and $\mathbf{v}_b$ are generated in

parallel. For a 30 MP photo, the time to generate a proof of knowledge for our Lattice + Poseidon hash construction in both VerITAS (10.25 min with non-blinding FRI-PCS and 20.29 min with blinding FRI-PCS) and opt-VerITAS (37.67 min with non-blinding FRI-PCS and 48.29 min with blinding FRI-PCS) is less than the time to generate a proof of knowledge for a Poseidon hash. In fact, when we tried to generate a proof of knowledge of a Poseidon hash for a picture of >1 MP, our machine ran out of memory and aborted the process (the 10 MP point at 34 min and 30 MP point at 103 min shown for Poseidon are projected points). With parallelism, the opt-VerITAS proving time could be cut down to 15 minutes with non-blinding FRI-PCS and to about 31 minutes with blinding FRI-PCS (which, again, would be a once-per-image cost). It takes less than a second to generate the Plonky2 proof that proves that applying Poseidon to the lattice hash results in the final public hash.

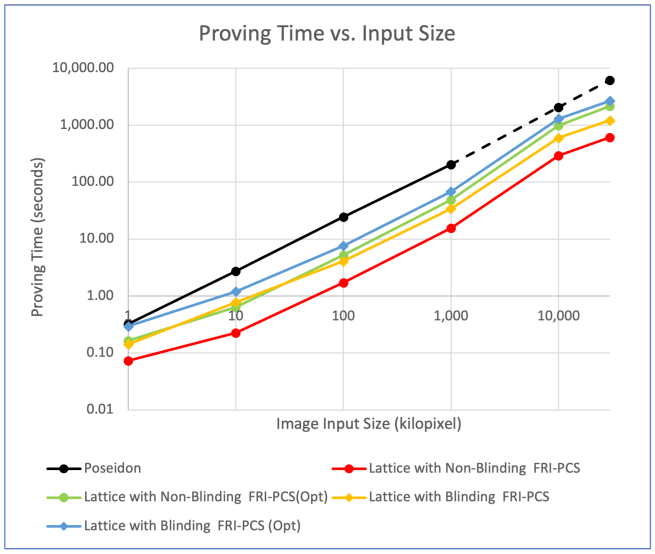

Figure 5: Graph showing proof generation time for generating a proof of a Poseidon hash and generating a proof of a Lattice hash (our construction) in both VerITAS and opt-VerITAS with both the non-blinding and blinding FRI-PCS. The dashed part of the Poseidon line refers to extrapolated values for sizes that exceeded the prover's capacity.

We report peak memory usage and verification time for our lattice hash proof generation using the non-blinding FRI-PCS in Table 1, and we report the same metrics using the blinding FRI-PCS in Table 2. For 30 MP images, verification time is about 15 seconds for VerITAS and about 0.8 seconds for opt-VerITAS. Proof size is about 530 KB for VerITAS and 2 MB for opt-VerITAS for a 30 MP image. Since these proofs are sent along with a 90 MB image, these sizes are reasonable.

7.2. Photo Edit Proof Generation Results

To demonstrate the practicality of our Plonky2 implementations, we report setup and proof generation timing

Image Size (KiloPixel)	Peak Memory (GB)	Verify Time (sec)	Opt Peak Memory (GB)	Opt Verify Time (sec)
1	3.33	0.004	3.39	0.013
10	3.40	0.009	3.35	0.017
100	3.55	0.058	3.55	0.022
1,000	5.08	0.502	6.12	0.045
10,000	37.99	6.70	62.79	0.435
30,000	75.60	15.32	119.52	0.783

TABLE 1: Prover memory needs and Verifier time for Lattice hash generation for VerITAS and opt-VerITAS with non-blinding FRI-PCS.

Image Size (KiloPixel)	Peak Memory (GB)	Verify Time (sec)	Opt Peak Memory (GB)	Opt Verify Time (sec)
1	3.33	0.005	3.39	0.013
10	3.40	0.011	3.40	0.019
100	3.55	0.054	3.55	0.022
1,000	5.08	0.49	6.12	0.047
10,000	45.04	5.87	65.33	0.47
30,000	87.82	16.38	127.14	0.763

TABLE 2: Prover memory needs and Verifier time for Lattice+Poseidon hash generation for VerITAS and opt-VerITAS with blinding FRI-PCS.

results for "realistic" image sizes. The signature-producing Sony camera mentioned earlier is a 33 MP camera. The edited photo size depends on clients. E.g., photos on *The New York Times* are resized to 2048 x 1365 pixels. Thus, in our experiments, for the editing operations that involve changing image dimensions (resizing and cropping), we report the times associated with resizing a 33 MP photo to the standard *New York Times* size. For operations that do not involve changing dimensions (grayscale conversion, blurring), we report the times associated with editing a photo of the standard *New York Times* size. For blurring, we report results for blurring 10% of the pixels. For operations where the RGB values in the new photo are calculated independently (cropping, resizing, and blurring), we report the time to generate a proof for a single RGB channel. Because the values for each channel are independent, proofs for these edits can be generated in parallel.

Table 3a shows the timing results for cropping proof generation for a single color channel. Table 3b shows the timing results for resizing proof generation for a single color channel. Table 3c shows the timing results for grayscale proof generation. Table 3d shows the timing results for blur proof generation for a single color channel. We include timing results for generating both zero-knowledge proofs and non-zero-knowledge proofs for each edit. Generating non-zero-knowledge proofs is faster than generating zero-knowledge proofs, so for edits whose purpose is not to hide sensitive information (e.g., grayscale conversion and resizing), editors may choose to generate non-zero-knowledge proofs to reduce proof generation time. For edits whose purpose is to hide sensitive information (e.g., cropping and

(a) Timing Results for Cropping

Original Size (pixels)	Reduced Size (pixels)	Setup Time (min)	Proof Gen Time (min)	ZK Setup Time (min)	ZK Proof Gen Time (min)
6632 x 4976	2048 x 1365	1.17	0.64	1.15	0.93

(b) Timing Results for Resizing

Original Size (pixels)	Reduced Size (pixels)	Setup Time (min)	Proof Gen Time (min)	ZK Setup Time (min)	ZK Proof Gen Time (min)
6632 x 4976	2048 x 1365	5.94	3.15	6.06	4.41

(c) Timing Results for Grayscale Conversion

Photo Size (pixels)	Setup Time (min)	Proof Gen Time (min)	ZK Setup Time (min)	ZK Proof Gen Time (min)
2048 x 1365	1.99	1.38	5.04	3.67

(d) Timing Results for Blurring

Original Size (pixels)	Blur Region Size (pixels)	Setup Time (minutes)	Proof Gen Time (minutes)	ZK Setup Time (min)	ZK Proof Gen Time (min)
2048 x 1365	529 x 529	1.74	1.36	4.47	4.20

TABLE 3: The time to generate a photo edit proof

blurring), editors can choose to generate zero-knowledge proofs.

Overall, setup and proof generation take just a few minutes. Because proofs only need to be generated once by the news organization, these times are suitable for practical implementation. Verification time less than a second. Plonky2 proofs are about 100-200 KB, which is reasonable compared to edited photos on the order of 8 MB. Moreover, Plonky2 proofs can be further compressed via a constant-sized zkSNARK (e.g., Groth16 or PLONK) that proves the correctness of Plonky2 proof verification.

7.3. Comparing the Two Signing Schemes

Table 4 compares how long it takes to calculate a SHA256 hash, a lattice hash, and a FRI polynomial commitment (as in Plonky2), of a 30 MP picture. We assume the photo is read in as a stream. In practice, we expect C2PA to use an FRI-based proof system, so we report lattice hash timings with a 64 bit prime q. As in our other experiments, we performed these experiments on a virtual machine with 131 GB of RAM and 12 CPU cores.

We first discuss the feasibility of computing a lattice hash on a camera. The VM on which we ran these experiments is much more powerful than a CPU-constrained camera, so instead of focusing on absolute time, we highlight the relative difference in time between calculating a SHA256 hash and lattice hash. Calculating a lattice hash is about twice as slow as calculating a SHA256 hash. Because several commercial cameras implementing photo signing using SHA256 hashing, increasing hash calculation time twofold seems reasonable. Hashing can also be made faster than SHA256 using a hardware hashing engine and parallelization. Table 4 shows that it takes much longer to compute a polynomial commitment of an image than a simple hash (using either SHA256 or lattice hash), making it much more feasible for a computationally-limited signer like a camera to sign a lattice hash rather than a polynomial commitment. Furthermore, FRI-PCS requires access to a large amount of memory to efficiently perform FFTs. Our experiments, for instance, required about 20 GB of RAM, which is far above than what a camera would have. Thus, in practice, computationally-limited signers, such as cameras, would most likely use our lattice-based signature scheme.

Hashing Scheme	Time (s)	Memory (GB)
SHA256	1.71	0.003
Lattice (64 bit)	4.24	0.003
FRI-PCS	19.84	18.90

TABLE 4: Timing comparison for different hashing schemes of a 30 MP image. Hashing using FRI-PCS takes much longer and requires more resources than the first two hash functions.

8. Related Work

One cryptographic tool proposed for image authentication is perceptual hashing [55]. The goal of perceptual hashing is to design a hash function that is resilient to content-preserving manipulations but can detect malicious manipulations. While this attempts to guarantee semantic meaning in a photo, our solution aims to guarantee something more rigorous, namely to certify that only certain edits have been made to a photo.

Another potential cryptographic approach to image authentication is homomorphic signatures [20], [21], as discussed in the introduction. Homomorphic signatures that are not built from a zk-SNARK can be used for cropping (or other forms of redaction), but are too inefficient to handle more complex image edits, such as blurring and resizing.

The most closely related work to ours is that of Naveh and Tromer [10] and Kang et al. [11] mentioned in the introduction. Those works applied to images that are more than an order of magnitude smaller than image sizes from modern cameras. More recently, Della Monica et al. [56] proposed dividing a photo into N non-overlapping "tiles" and producing a proof for each tile that attests to the hash and a certain transformation on that image. Because these tiles are smaller than the entire image, the memory and time required to generate the N proofs is smaller than the memory and time required to generate one large proof. Just like Kang et al., Della Monica et al. only consider photos ≤ 900 KP, which is an order of magnitude smaller than the

photos we consider. Moreover, verification time for these tiled proofs is about 3 minutes, which could be problematic for someone reading a newspaper in a browser. Another issue with this approach is that image transformations must be applied per tile rather than on the image as whole, which is the standard practice in photo editing software like Adobe Photoshop. For instance, to resize or blur a photo, tiles must be individually resized or blurred and then collated together. Consequently, these methods are unable to support standard Photoshop algorithms. Very recently Dziembowski et al. [57] experimented with folding schemes for proving image edits.

We also note that previous work by Ben-Sasson et al. [14] and Kosba et al. [15] has, like VerITAS, used lattice-based hashes as SNARK-friendly hashes.

9. Extensions and Conclusion

In this paper, we have discussed how to use zk-SNARKs to enable practical provenance verification for realistically large edited images in online news articles. Our system uses signing keys embedded in cameras as the origin of trust, but rather than trusting a third-party application to digitally sign edited images, we propose to use zero-knowledge proofs to prove to a news reader that an edited published photo was taken when and where the article claims it was taken. We create proofs for 30 MP images, which is the size of images produced by actual cameras equipped with embedded signing keys. The bottleneck in image editing proof systems is proving knowledge of a valid signature on the unedited photo. Our key innovations are two-fold: first, we introduce a new SNARK-friendly hashing method that reduces the hash proof generation time. We believe this SNARK-friendly method, which is a sequential composition of lattice hash with a Poseidon hash, may be of independent interest to those looking to create SNARKs that prove hashes of large amounts of data. Additionally, we introduce a polynomial commitment hash that completely eliminates the need for proving knowledge of a valid signature in the SNARK circuit. However, signing the unedited image using a polynomial commitment hash is more expensive than the lattice hash scheme.

We note that the description of VerITAS given here does not protect the identity of the signer (the photographer). Indeed, vk (and in σ in mode 2) are sent along with the ZK proof to the verifier. If the editor wants to hide the identity of the signer, then the editor could replace vk (and σ) by a public commitment com to those values, and move vk and σ to the zk-SNARK secret witness. The zk-SNARK circuit would then verify that com is a valid commitment to (vk, σ), instead of directly using those values as public inputs. This fully hides vk and σ from the verifier and protects the identity of the signer.

Finally, this paper has only discussed how to prove edits for photos, but videos are also a major source of misinformation. The main challenge with videos is that once they are edited, they are stored in a lossy compressed format. Directly applying the techniques discussed here to videos would thus require us to prove statements about video compression in a SNARK, which is challenging due to the size of a video file. Another avenue for future research is exploring different kinds of range proofs. In our work, we used a Plookup-based range proof. More recent methods, such as Lasso [30], may lead to time and memory savings for the editor.

Acknowledgments. This work was funded by NSF, DARPA, the Simons Foundation, UBRI, and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[1] A. Coleman and S. Sardarizadeh, "Ukraine conflict: Many misleading images have been shared online," *BBC News*, 2022, link.

[2] V. Pavilonis, "Fact check: Images show mosul in 2017, kyiv one day after russian invasion began," *USA Today*, 2022, link.

[3] A. Coleman, "Ukraine conflict: Further false images shared online," *BBC News*, 2022, link.

[4] "BBC breakfast uses old footage of russian parade rehearsal to show invasion of ukraine," *Full Fact*, 2022, link.

[5] "C2PA technical specification," link.

[6] "Partnership for greater trust in digital photography: Leica and content authenticity initiative," *Leica*, 2022, link.

[7] "Sony unlocks in-camera forgery-proof technology," *Sony*, 2022, link.

[8] "Visuals," *Associated Press*, 2022, link.

[9] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *ITCS 2012*, S. Goldwasser, Ed. ACM, Jan. 2012, pp. 326–349.

[10] A. Naveh and E. Tromer, "Photoproof: Cryptographic image authentication for any set of permissible transformations," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 255–271.

[11] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, "ZK-IMG: Attested images via zero-knowledge proofs to fight disinformation," arXiv 2211.04775, 2022.

[12] T. Datta and D. Boneh, "Using ZK proofs to fight disinformation," Medium, 2022, link.

[13] ——, "Using ZK proofs to fight disinformation," Real World Crypto, 2023, link.

[14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 276–294.

[15] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 839–858.

[16] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast reed-solomon interactive oracle proofs of proximity," in *ICALP 2018*, ser. LIPIcs, I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds., vol. 107. Schloss Dagstuhl, Jul. 2018, pp. 14:1–14:17.

[17] "plonky2," https://github.com/0xPolygonZero/plonky2.

[18] R. Johnson, D. Molnar, D. Song, and D. Wagner, "Homomorphic signature schemes," in *Topics in Cryptology — CT-RSA 2002*, B. Preneel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 244–262.

[19] D. Derler, H. C. Pöhls, K. Samelin, and D. Slamanig, "A general framework for redactable signatures and new constructions," in *Information Security and Cryptology - ICISC 2015*, S. Kwon and A. Yun, Eds. Cham: Springer International Publishing, 2016, pp. 3–19.

[20] D. Boneh and D. M. Freeman, "Homomorphic signatures for polynomial functions," in *EUROCRYPT 2011*, ser. LNCS, K. G. Paterson, Ed., vol. 6632. Springer, Heidelberg, May 2011, pp. 149–168.

[21] S. Gorbunov, V. Vaikuntanathan, and D. Wichs, "Leveled fully homomorphic signatures from standard lattices," in *47th ACM STOC*, R. A. Servedio and R. Rubinfeld, Eds. ACM Press, Jun. 2015, pp. 469–477.

[22] "Image size and resolution," Adobe, 2024, link.

[23] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*, 2023, https://toc.cryptobook.us/book.pdf.

[24] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 2010, pp. 177–194.

[25] J. Groth, "On the size of pairing-based non-interactive arguments," Cryptology ePrint Archive, Paper 2016/260, 2016, https://eprint.iacr.org/2016/260. [Online]. Available: https://eprint.iacr.org/2016/260

[26] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Paper 2019/953, 2019, https://eprint.iacr.org/2019/953. [Online]. Available: https://eprint.iacr.org/2019/953

[27] A. Gabizon and Z. J. Williamson, "Plookup: A simplified polynomial protocol for lookup tables," Cryptology ePrint Archive, Paper 2020/315, 2020, https://eprint.iacr.org/2020/315. [Online]. Available: https://eprint.iacr.org/2020/315

[28] A. Zapico, A. Gabizon, D. Khovratovich, M. Maller, and C. Ràfols, "Baloo: Nearly optimal lookup arguments," Cryptology ePrint Archive, Paper 2022/1565, 2022, https://eprint.iacr.org/2022/1565. [Online]. Available: https://eprint.iacr.org/2022/1565

[29] L. Eagen, D. Fiore, and A. Gabizon, "CQ: Cached quotients for fast lookups," Cryptology ePrint Archive, Paper 2022/1763, 2022, https://eprint.iacr.org/2022/1763. [Online]. Available: https://eprint.iacr.org/2022/1763

[30] S. Setty, J. Thaler, and R. Wahby, "Unlocking the lookup singularity with lasso," Cryptology ePrint Archive, Paper 2023/1216, 2023, link.

[31] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *J. ACM*, vol. 27, no. 4, p. 701–717, oct 1980. [Online]. Available: https://doi.org/10.1145/322217.322225

[32] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, ser. EUROSAM '79. Berlin, Heidelberg: Springer-Verlag, 1979, p. 216–226.

[33] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.

[34] T. Attema, S. Fehr, and M. Klooß, "Fiat-shamir transformation of multi-round interactive proofs (extended version)," *Journal of Cryptology*, vol. 36, no. 4, p. 36, Oct. 2023.

[35] U. Haböck, "Multivariate lookups based on logarithmic derivatives," Cryptology ePrint Archive, Report 2022/1530, 2022, https://eprint.iacr.org/2022/1530.

[36] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, "HyperPlonk: Plonk with linear-time prover and high-degree custom gates," in *EUROCRYPT 2023, Part II*, ser. LNCS, C. Hazay and M. Stam, Eds., vol. 14005. Springer, Heidelberg, Apr. 2023, pp. 499–530.

[37] M. Ajtai, "Generating hard instances of lattice problems," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 99–108.

[38] O. Goldreich, S. Goldwasser, and S. Halevi, "Collision-free hashing from lattice problems," ser. Lecture Notes in Computer Science. Springer, 2011, vol. 6650, pp. 30–39.

[39] M. Bellare, D. Micciancio, and B. Warinschi, "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions," in *EUROCRYPT 2003*, ser. LNCS, E. Biham, Ed., vol. 2656. Springer, Heidelberg, May 2003, pp. 614–629.

[40] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *CRYPTO 2004*, ser. LNCS, M. Franklin, Ed., vol. 3152. Springer, Heidelberg, Aug. 2004, pp. 41–55.

[41] "circom-ecdsa circuit," link.

[42] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems," Cryptology ePrint Archive, Paper 2019/458, 2019, https://eprint.iacr.org/2019/458. [Online]. Available: https://eprint.iacr.org/2019/458

[43] "Poseidon circuit," link.

[44] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," Cryptology ePrint Archive, Report 2019/953, 2019, https://eprint.iacr.org/2019/953.

[45] A. Gabizon and Z. J. Williamson, "plookup: A simplified polynomial protocol for lookup tables," Cryptology ePrint Archive, Report 2020/315, 2020, https://eprint.iacr.org/2020/315.

[46] R. Freivalds, "Probabilistic machines can use less running time," p. 839–842, 1977.

[47] J. Thaler, "The unreasonable power of the sum-check protocol," The Art of Zero Knowledge, 2020, link.

[48] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent succinct arguments for R1CS," in *EUROCRYPT 2019, Part I*, ser. LNCS, Y. Ishai and V. Rijmen, Eds., vol. 11476. Springer, Heidelberg, May 2019, pp. 103–128.

[49] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *40th ACM STOC*, R. E. Ladner and C. Dwork, Eds. ACM Press, May 2008, pp. 113–122.

[50] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 2007.

[51] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," Cryptology ePrint Archive, Paper 2015/046, 2015, https://eprint.iacr.org/2015/046. [Online]. Available: https://eprint.iacr.org/2015/046

[52] S. Valentine, "How photoshop translates rgb color to gray," *insider*, 2018, https://insider.kelbyone.com/how-photoshop-translates-rgb-color-to-gray-by-scott-valentine/.

[53] "Blur and sharpen effects," Adobe, 2024, link.

[54] "arkworks," https://github.com/arkworks-rs/.

[55] F. Ahmed, M. Y. Siyal, and V. Uddin Abbas, "A secure and robust hash-based scheme for image authentication," *Signal Process.*, vol. 90, no. 5, p. 1456–1470, may 2010, link.

[56] P. D. Monica, I. Visconti, A. Vitaletti, and M. Zecchini, "Do not trust anybody: Zk proofs for image transformations tile by tile on your laptop," Real World Crypto, 2024.

[57] S. Dziembowski, S. Ebrahimi, and P. Hassanizadeh, "VIMz: Verifiable image manipulation using folding-based zkSNARKs," Cryptology ePrint Archive, Paper 2024/1063, 2024, https://eprint.iacr.org/2024/1063. [Online]. Available: https://eprint.iacr.org/2024/1063

Appendix A. Preliminaries

This appendix contains the security definitions for the primitives defined in Section 2.

A.1. Digital Signatures

For a signature scheme SIG to be existentially unforgeable under a chosen message attack, every efficient adversary $\mathcal{A}$ with access to the verification key vk, and a signing oracle for messages of its choice, should not be able to produce a signature on a new message m^* for which $\mathcal{A}$ has not queried the signing oracle. The advantage of the adversary $\mathcal{A}$ in the corresponding security game with security parameter λ is $\mathsf{Adv}^{\mathrm{sig}}_{\mathcal{A},\mathsf{SIG}}(\lambda)$.

A.2. Commitment Schemes

We define the hiding and binding security property for commitment schemes below:

- **Hiding:** for every PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$:

$$\Pr\left[b' \neq b : \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda) \\ (x_0, x_1) \leftarrow\!\$\ \mathcal{A}(\mathsf{pp}) \\ b \leftarrow\!\$\ \{0,1\} \\ r \leftarrow\!\$\ \mathcal{R}_{\mathrm{C}} \\ c \leftarrow \mathsf{commit}(\mathsf{pp}, m_b, r) \\ b' \leftarrow \mathcal{A}(c) \end{array}\right] \leq \frac{1}{2} + \nu(\lambda)$$

- **Binding:** for every PPT adversary $\mathcal{A}$ the following probability is negligible

$$\mathsf{Adv}^{\mathrm{bind}}_{\mathcal{A},C}(\lambda) := \Pr\big[\mathsf{commit}(\mathsf{pp}, x, r) = \mathsf{commit}(\mathsf{pp}, x', r')\big]$$

where $\mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda)$ and $(x, x', r, r') \leftarrow\!\$\ \mathcal{A}(\mathsf{pp})$.

We define correctness, evaluation binding, and hiding for a polynomial commitment scheme below.

- **Correctness**: for all $\lambda, d \in \mathbb{N}$, all $x \in \mathbb{F}$, and all $f \in \mathbb{F}[X]$ of degree a most d, the following probability is 1:

$$\Pr\left[\begin{array}{l} \mathsf{Vf}(pp, \mathsf{com}, \\ x, y, \pi) = 1 \end{array} : \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda, d) \\ r \leftarrow\!\$\ \mathcal{R}_{\mathrm{C}} \\ \mathsf{com} \leftarrow \mathsf{commit}(\mathsf{pp}, f, r) \\ (\pi, y) \leftarrow \mathsf{open}(\mathsf{pp}, f, x, r) \end{array}\right]$$

- **Evaluation Binding**: it is not possible to open a committed polynomial to two different values at one point. That is, for every PPT adversary $\mathcal{A}$ and for all $\lambda, d \in \mathbb{N}$, the following function is negligible

$$\Pr\left[\begin{array}{c} \mathsf{Vf}(\mathsf{pp}, \mathsf{com}, x, y, \pi) = 1 \\ \wedge \\ \mathsf{Vf}(\mathsf{pp}, \mathsf{com}, x, y', \pi') = 1 \\ \wedge y \neq y' \end{array} : \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda, d) \\ (\mathsf{com}, x, y, \pi, \\ y', \pi') \leftarrow\!\$\ \mathcal{A}(\mathsf{pp}, d) \end{array}\right]$$

Optionally, a polynomial commitment scheme can also satisfy the following property:

- **Hiding**: for every PPT adversary $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that:

$$\Pr\left[b' \neq b : \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda) \\ (f_0, f_1) \leftarrow\!\$\ \mathcal{A}(\mathsf{pp}) \\ b \leftarrow\!\$\ \{0,1\} \\ r \leftarrow\!\$\ \mathcal{R}_{\mathrm{C}} \\ c \leftarrow \mathsf{commit}(\mathsf{pp}, f_b, r) \\ b' \leftarrow \mathcal{A}(c) \end{array}\right] \leq \frac{1}{2} + \nu(\lambda)$$

A.3. zk-SNARKs

We define completeness, knowledge soundness, zero-knowledge, non-interactivity, and succinctness for a zk-SNARK below.

- **Completeness**: if $(x, w) \in \mathcal{R}$, then verification should pass. That is, for all $\lambda \in \mathbb{N}$ and all $(x, w) \in \mathcal{R}$:

$$\Pr\left[\ \mathsf{Vf}(\mathsf{pp}, x, \pi) = 1 \quad : \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda) \\ \pi \leftarrow \mathsf{prove}(\mathsf{pp}, x, w) \end{array}\right] = 1$$

- **Knowledge Soundness**: if an adversary can produce a valid proof for some x, then there should be a polytime extractor that can compute a witness w such that $(x, w) \in \mathcal{R}$. That is, Π has knowledge error $\epsilon \in [0, 1]$ if for every PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ there exists a PPT extractor $\mathcal{E}$ such that:

$$\Pr\left[\ (x, w) \in \mathcal{R} \quad : \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda) \\ (x, \mathsf{state}) \leftarrow\!\$\ \mathcal{A}_0(\mathsf{pp}) \\ w \leftarrow\!\$\ \mathcal{E}^{\mathcal{A}_1(\mathsf{state})}(\mathsf{pp}, x) \end{array}\right] \geq$$
$$\Pr\left[\ \mathsf{Vf}(\mathsf{pp}, x, \pi) = 1 \quad : \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda) \\ (x, \mathsf{state}) \leftarrow\!\$\ \mathcal{A}_0(\mathsf{pp}) \\ \pi \leftarrow\!\$\ \mathcal{A}_1(\mathsf{state}) \end{array}\right] - \epsilon$$

- **Zero-Knowledge**: We state the definition in the random oracle model where all the algorithms are oracle machine that can query an oracle $H : \mathcal{X} \to \mathcal{Y}$ for some finite sets $\mathcal{X}$ and $\mathcal{Y}$. The zk-SNARK is zero knowledge if there is a PPT simulator $\Pi.\mathsf{sim}$ such that for all $(x, w) \in \mathcal{R}$ and all PPT adversaries $\mathcal{A}$, the following function is negligible

$$\mathsf{Adv}^{\mathrm{zk}}_{\mathcal{A},\Pi}(\lambda) := \left|\begin{array}{r} \Pr\big[\mathcal{A}^H\big(\mathsf{pp}, x, \mathsf{prove}^H(\mathsf{pp}, x, w)\big) = 1\big] - \\ \Pr\big[\mathcal{A}^{H[h]}\big(\mathsf{pp}, x, \pi\big)\big) = 1\big] \end{array}\right|$$

where $\mathsf{pp} \leftarrow\!\$\ \mathsf{setup}(1^\lambda)$ and $(\pi, h) \leftarrow\!\$\ \Pi.\mathsf{sim}(\mathsf{pp}, x)$. Here h is a partial function $h : \mathcal{X} \to \mathcal{Y}$ output by $\Pi.\mathsf{sim}$, and $H[h]$ refers to the oracle $H : \mathcal{X} \to \mathcal{Y}$ modified by entries in h. That is, we allow $\Pi.\mathsf{sim}$ to program the oracle H.
- **Non-interactive**: the proof is non-interactive, and a proof created by the prover can be checked by any verifier.
- **Succinct**: the proof size and verifier runtime are $o(|w|)$. The verifier can run in linear time in $|x|$.