VIMz: Private Proofs of Image Manipulation using Folding-based zkSNARKs*

Stefan Dziembowski
University of Warsaw / IDEAS NCBR
stefan.dziembowski@crypto.edu.pl

Shahriar Ebrahimi
IDEAS NCBR
shahriar.ebrahimi@ideas-ncbr.pl

Parisa Hassanizadeh
IPPT PAN / IDEAS NCBR
parisa.hassanizadeh@ideas-ncbr.pl

Abstract

Ensuring the authenticity and credibility of daily media on internet is an ongoing problem. Meanwhile, genuinely captured images often require refinements before publication. Zero-knowledge proofs (ZKPs) offer a solution by verifying edited image without disclosing the original source. However, ZKPs typically come with high costs, particularly in terms of prover complexity and proof size. This paper presents VIMz, a framework for efficiently proving the authenticity of high-resolution images using folding-based zkSNARKs; a type of proving system that minimizes computational overhead by recursively folding multiple evaluations of the same constraints into a compact proof. As a complete proof system, VIMz proves the integrity of both the original and edited images, as well as the correctness of the transformation without revealing intermediate images within a chain of edits—only the final result is disclosed. Moreover, VIMz maintains the anonymity of the original signer and all subsequent editors while proving the authenticity of the final image. We also compare VIMz with the system model in Coalition for Content Provenance and Authenticity (C2PA) from different perspectives and show that VIMz offers higher level of security guarantee by eliminating the need to trust the editing environment. Experimental results show that VIMz performs efficiently in both prover and verifier sides. It can prove the transformations on 8K (33MP,i.e., 100MB) images with up to 13%∼25% faster than the competition, while reaching to a peak memory of only 10 GB. Moreover, VIMz has a verification time of under 1 second and achieves succinct proofs of less than 11 KB for all resolutions, which is more than 90% improvement compared to the competition. VIMz's low memory complexity allows for proving multiple transformations in parallel to achieve a 3.5× additional speedup on average.

Keywords

zkSNARKs, Proof of Provenance, C2PA, Folding Schemes

1 Introduction

With the growing accessibility of generative AI, ensuring media originality and authenticity is becoming challenging. A notable example occurred in 2023 when the recipient of the Sony World Photography Award declined the honor to reveal that the image was generated by AI [25]. Currently, it is relatively easy to create realistic face-swapped videos and images using consumer-level GPUs and libraries like DeepFaceLab [3]. Furthermore, popular text-to-image models such as Midjourney [5], DALL•E [2], and Stability AI [15] are accessible through affordable monthly subscriptions. Therefore, it is necessary to assess the sources of information an distinguish between original and synthetically produced or altered media.

To this end, leading corporations are actively engaged in research on detecting deepfakes [34, 36, 47]. Several studies aim to train models for detecting artifacts or forensic noises in AI-generated media [28, 52, 53], and some others explore utilization of advance watermarks [45]. Ideally, a perfect model would distinguish between authentic and artificially generated images without relying on any trust assumptions. Nevertheless, recent reviews [36] emphasize a significant accuracy gap between generative AI and deepfake detection models still. This suggests that even on limited datasets, such techniques fall short of being reliable for real-world applications.

Another solution involves embedding signed image metadata (e.g., location, photographer, date, and time) within the image data and recording signed edits applied to the original image. A notable initiative of this approach is the Coalition for Content Provenance and Authenticity (C2PA) [9] project. From a cryptographic standpoint, approaches like the C2PA solution rely on digital signatures. When a user captures a photo, the camera or a specific software application (e.g., Truepic [16] on mobile phones) adds metadata to the photo and signs it. The photo is then refined by some trusted editing software (e.g., Adobe Photoshop). As edits are applied to the image, newly signed metadata is appended to the existing records. In an ideal world, trusting the software to handle the signatures and their corresponding private keys would make the proposed C2PA protocol complete. However, in practice, such reliance on software has been shown to introduce direct and serious attack vectors [35, 37, 39, 40, 46]. To this end, as suggested in the C2PA trust model [8], the software should be run within trusted execution environment (TEE) e.g., Intel SGX [21], which, in turn, introduces a new layer of trust assumption and opens new attack vectors [19, 29, 41].

Another drawback of approaches like C2PA is that the privacy of the media provider, e.g., photographer, is not guaranteed by the protocol trustlessly. At some point, the media provider is required to disclose confidential data to at least one trusted party.

A more ideal and secure approach would involve the ability to prove the authenticity of the refined image solely with a valid signature of the original image, without relying on additional trust assumptions in third parties or specific software. This can be achieved by leveraging techniques like SNARKs[1] to generate a succinct proof for the edited image that is publicly verifiable by everyone [30, 42, 49]. Unlike the approach of C2PA, the SNARKs proving systems do not require any kind of trust during proof generation.

*Author names are ordered alphabetically.

Proceedings on Privacy Enhancing Technologies 2025(2), 344–362

https://doi.org/10.56553/popets-2025-0065

[1] Succinct Non-Interactive Arguments of Knowledge.

Consequently, anyone can publicly verify whether the refined image aligns with the provided proof or not.

However, the main drawback of this approach is the computational complexity of the prover, particularly with the increasing resolutions of images to 4K and 8K. For example, in related work [30], complete[2] proof generation (proof of valid transformations along with proof of integrity) for one convolution-based transformation on an HD image demands up to 309 GB of RAM and takes over 21 minutes of computation on a 64-core AWS server [30].

To tackle the challenge of high memory requirements, we propose breaking down each transformation into a series of recursive steps. Then, we prove the correctness of each step within an incrementally verifiable computation (IVC) framework. Finally, we "*fold*" these steps together using Nova's folding scheme [33], resulting in a compressed SNARKs that proves the entire transformation process. This method limits peak memory usage to only the maximum resources needed for computations within a single step, and therefore, significantly reduces overall memory requirements. The outcome is VIMz[3] (Verifiable Image Manipulation using folding-based zkSNARKs), which, is the first system of its kind to leverage the higher efficiency of folding-based zkSNARKs. Implementation of VIMz is entirely open-source[4] and comprises various programming stacks, including Python, Circom and Rust.

VIMz has low memory requirement for proving image transformations of high resolutions. Our experiments demonstrate that it is feasible to prove transformations on images ranging from HD to 8K resolutions using only a midrange laptop with an Intel Core i5 CPU. Additionally, we show that even on such a hardware, multiple VIMz instances can run in parallel to prove a sequence of edits, achieving 3.5× additional speedup in proving time.

VIMz offers several advantages over the state of the art. Firstly, VIMz generates succinct proofs,with proof sizes for 8K (33 MP) images being less than 11 KB—up to 90% smaller than competing solutions [22, 38]. Additionally, VIMz achieves 20% faster proving time compared to these alternatives. Moreover, unlike the competition [22, 38], VIMz provides *"complete proofs"* that verify not only the correctness of transformations but also the hashes of both the original and transformed images. This ensures that each proof serves as a binding commitment to the resulting transformed image. Therefore, we can securely prove the authenticity of a sequence of transformations while keeping the intermediate images confidential, a capability not offered by competitors such as [22] or [38].

Since VIMz operates without trust assumptions on the prover, it preserves the privacy of all editors. In contrast, approaches like C2PA require the editor to sign edits, which in turn, reveals the signer's identity during verification. Our protocol goes further by not only preserving the editor's identity but also anonymizing the identity of the original image signer. We achieve this while ensuring that the original signer possesses an authorized private key.

The key contributions of this paper are as follows:

- **IVC-based proofs of image manipulation:** We introduce the first proving framework for image manipulation using folding-based zkSNARKs[4], namely VIMz, that enables efficient IVC proofs of image provenance.
- **Succinct proofs of image manipulation:** We achieve a proof size of around 10 KB for images with 8K (33 MP) resolution,which is 92% smaller than the competition. This characteristic makes VIMz well-suited to be integrated into blockchain-based applications.
- **Optimized proofs of integrity:** We propose a folding-friendly image hashing method that outperforms the competition [22] in both proving time and memory efficiency. This enables proving image hashes both before and after transformation. To further increase the efficiency, we introduce a lossless compression technique for pixel data within the Pallas/Vesta field, reducing the number of hashes by 30×.
- **Memory efficiency:** Our analysis measured a peak memory usage of just 2.9 GB, 5.6 GB, and 9.5 GB when proving transformations in HD, 4K, and 8K resolutions, respectively. This allows VIMz to prove such transformations on a midrange laptop with only 16 GB of RAM.
- **Fastest Prover:** VIMz provides the fastest prover compared to the competition.
- **Chained edits proved in parallel:** Unlike the competition [22, 38], VIMz allows unlimited chained transformations. The prover is only required to disclose the last refined image, while all intermediate versions remain confidential. This feature, combined with low memory requirements, enables running multiple instances of VIMz in parallel to speedup the process by more than 3.5× on average.
- **C2PA Compatibility:** We demonstrate that VIMz can be viewed as an enhanced version of C2PA, providing higher security guarantees with fewer trust assumptions, while also preserving the privacy of both the image signer (creator/owner) and the editors.
- **Privacy-preserving proofs of image manipulation:** VIMz allows to efficiently anonymize both the original image signer and subsequent editors.

The remainder of the paper is structured as follows. Section 2 offers the necessary background to comprehend the paper. In Section 3, we analyze potential approaches for image provenance and provide the rationale behind the construction of VIMz. Section 4 offers an overview of the protocol, detailing the statements and analyzing the soundness of the proofs generated by VIMz. Section 5 details the circuit design along with optimizations in both circuit and execution levels. Section 6 presents experimental analysis of VIMz during the proof generation for transformations different resolutions up to 8K (33 MP). Section 7 compares VIMz against related work. Finally, in Section 8, we conclude the paper.

2 Background

Table 1 presents the terminology of the paper. We generally borrow the mathematical representations from [32, 33]. The rest of the section overviews the main concepts employed to build VIMz.

2.1 SNARKs

Let $\mathcal{R}$ be a binary relation for an NP language $L_{\mathcal{R}}$, where λ is the security parameter. The argument system for $\mathcal{R}$ is defined as a

[2] In this context, a "complete" proof involves not only proofs for correct transformation but also hashing both the original and resulting images, serving as *proofs of integrity*. The hash acts as a binding factor for verification against a claimed refined image.
[3] Pronunciation: `/'wimzi/`, like the word *whimsy*!
[4] Github link: https://github.com/zero-savvy/vimz

Table 1: Terminology of the Paper

Notation	Description
α, β	We refer to the pixel matrices of the *original* and the *transformed* image as α and β, respectively.
$\alpha^{\{R\|G\|B\}}$	*Red*, *Green*, and *Blue* color plains of the image α.
α_i, β_i	i-th row of images α and β.
$\alpha_{i,j}$	Pixel value in the i-th row and j-th column of α.
H	Poseidon [26] hash function. $H : \mathbb{Z}_p^2 \to \mathbb{Z}_p$
H_σ	Hash value of an entire image row with n pixels: $H_\sigma : \mathbb{Z}_p^n \to \mathbb{Z}_p = H(\alpha_{i,n-1}\|H(\alpha_{i,n-2}\|H(\alpha_{i,n-3}\| \ldots \|H(\alpha_{i,2}\|H(\alpha_{i,1}\|H(\alpha_{i,0}\|0))) \ldots))$.
h^i	Cumulative hash value of rows of an image, e.g., $h^i \leftarrow H(h^{i-1}\|H_\sigma(\alpha_i))$
H_ϕ	Hash value of an entire image with $n \times m$ pixels: $H_\phi : \mathbb{Z}_p^{n\times m} \to \mathbb{Z}_p = h^n$.
f_T	Transformation function: $\beta \leftarrow f_T(\alpha, U_{in})$
$\mathcal{F}_T$	Folding-friendly verifiable image transformer (*Definition* 4.4)
U_{in}^i, U_{out}^i	Public input and output of i-th step in IVC.
κ	Kernel matrix in convolution.
O_p	Merkle proof checker $O_p : \mathbb{Z}_q^{m+\lceil \log n \rceil} \to \mathbb{B}$, which takes as input the Merkle path, the root, and m additional values to open the commitment at a leaf.
$\mathcal{V}_{sig}$	Verifies a signature against the given message and public key: $\mathcal{V}_{sig} : \mathbb{Z}_q^3 \to \mathbb{B}$.

quadruple probabilistic polynomial algorithms $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{S})$ and a deterministic encoder $\mathcal{K}$, where:

- $pp \leftarrow \mathcal{G}(1^\lambda)$: The generator samples the public parameter pp w.r.t. the security parameter λ.
- $(pk, vk) \leftarrow \mathcal{K}(pp, s)$: The prover and verifier key pair is derived from the commonly defined structure s and the public parameter pp using the deterministic encoder.
- $\pi \leftarrow \mathcal{P}(pk, u, w)$: Proving algorithm stating $(pp, s, u, w) \in \mathcal{R}$.
- $b \leftarrow \mathcal{V}(vk, u, \pi)$: Verification algorithm, where $b \in \{0, 1\}$.
- $\pi \leftarrow \mathcal{S}(pp, u, \tau)$: Simulator outputs π given trapdoor τ.

Formally, the properties of [zk]SNARKs are as follows.

Definition 2.1. A non-interactive argument for $\mathcal{R}$ is a SNARK if it satisfies:

- **Completeness**: An honest prover with valid witness should convince any verifier. Formally, for any PPT adversary $\mathcal{A}$:

$$Pr\left[\begin{array}{l|l} \mathcal{V}(vk, u, \pi) = 1 & \begin{array}{l} pp \leftarrow \mathcal{G}(1^\lambda) \\ (s, (u, w))) \leftarrow \mathcal{A}(pp) \\ (pp, s, u, w) \in \mathcal{R} \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ \pi \leftarrow \mathcal{P}(pk, u, w) \end{array} \end{array}\right] = 1$$

- **Knowledge Soundness**: A dishonest prover (adversary), should not be able to convince any verifier. To formally define this we require that for all PPT adversaries $\mathcal{A}$ there exists an extractor $\mathcal{E}$ that can compute witness given any randomness ρ, such that:

$$Pr\left[\begin{array}{l|l} \mathcal{V}(vk, u, \pi) = 1, & \begin{array}{l} pp \leftarrow \mathcal{G}(1^\lambda) \\ (s, (u, w))) \leftarrow \mathcal{A}(pp) \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ w \leftarrow \mathcal{E}(pp, \rho) \end{array} \\ (pp, s, u, w) \notin \mathcal{R} & \end{array}\right] = \text{negl}(\lambda)$$

- **Zero-knowledge**: If the argument dos not reveal anything beyond the truth of the statement, we label it as zero-knowledge. Formally, there must exist a PPT simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$ following distributions are indistinguishable:

$$\mathcal{D}_1 = \left\{ (pp, s, u, \pi) \;\middle|\; \begin{array}{l} pp \leftarrow \mathcal{G}(1^\lambda) \\ (s, (u, w))) \leftarrow \mathcal{A}(pp) \\ (pp, s, u, w) \in \mathcal{R} \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ \pi \leftarrow \mathcal{P}(pk, u, w) \end{array} \right\}$$

$$\mathcal{D}_2 = \left\{ (pp, s, u, \pi) \;\middle|\; \begin{array}{l} (pp, \rho) \leftarrow \mathcal{S}(1^\lambda) \\ (s, (u, w))) \leftarrow \mathcal{A}(pp) \\ (pp, s, u, w) \in \mathcal{R} \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ \pi \leftarrow \mathcal{S}(pp, u, \rho) \end{array} \right\}$$

2.2 Incrementally Verifiable Computation (IVC)

IVC allows verification of computations done by repeatedly applying the same function. More precisely, for a given function f, with initial input z_0, IVC allows for generating a proof Π_i stating that $z_i = f^i(z_0)$, given a proof Π_{i-1} stating $z_{i-1} = f^{i-1}(z_0)$. An interesting property of IVC schemes is that they support additional auxiliary inputs for f. While the main input for each iteration of applying f comes from previous step, the auxiliary input ω_i in each step is independent from other steps. Therefore, IVC schemes further extend the completeness and soundness properties as follows.

Definition 2.2. We can define IVC by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic encoder $\mathcal{K}$ satisfying:

- **Completeness**: For any PPT adversary $\mathcal{A}$:

$$Pr\left[\begin{array}{l|l} \begin{array}{l} \mathcal{V}(vk, i, z_0, \\ z_i, \Pi_i) = 1 \end{array} & \begin{array}{l} pp \leftarrow \mathcal{G}(1^\lambda) \\ f, (i, z_0, z_{i-1}, z_i, w_{i-1}, \Pi_{i-1}) \leftarrow \mathcal{A}(pp) \\ z_i = f(z_{i-1}, \omega_{i-1}) \\ (pk, vk) \leftarrow \mathcal{K}(pp, f) \\ \mathcal{V}(vk, i-1, z_0, z_{i-1}, \Pi_{i-1}) = 1 \\ \Pi_i \leftarrow \mathcal{P}(pk, i, z_0, z_i; z_{i-1}, \omega_{i-1}, \Pi_{i-1}) \end{array} \end{array}\right] = 1$$

- **Knowledge Soundness**: $\forall n \in \mathbb{N}$, and expected polynomial time adversaries $\mathcal{P}^*$, there exists expected polynomial time extractor $\mathcal{E}$, such for any randomness ρ, following probability is negligible:

$$Pr\left[\begin{array}{l|l} \begin{array}{l} z_n \neq z, \\ \mathcal{V}(vk, n, z_0, z, \Pi) = 1 \end{array} & \begin{array}{l} pp \leftarrow \mathcal{G}(1^\lambda) \\ f, (z_0, z, \Pi) \leftarrow \mathcal{P}^*(pp; \rho) \\ (pk, vk) \leftarrow \mathcal{K}(pp, f) \\ (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(pp, z_0, z; \rho) \\ z_i \leftarrow f(z_{i-1}, \omega_{i-1}) \;\; \forall i \in \{1, \ldots, n\} \end{array} \end{array}\right]$$

2.3 Nova Proving System

Nova [33] employs an efficient folding-based SNARK to achieve efficient IVC. Folding schemes are cryptographic primitives that simplify the verification of two NP statements into checking a single NP statement. This allows the prover to incrementally prove

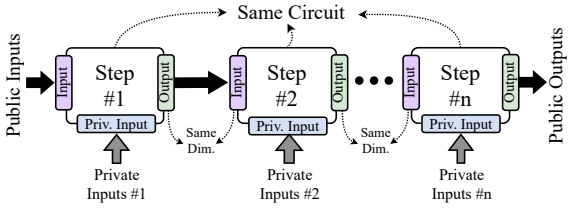

Figure 1: The folding scheme structure in Nova [33].

correct execution for sequential computations represented by the form $y = F^l(x)$, where F is the computation, x is the input, and $l > 0$. Notably, Nova provides one of the fastest *transparent*[5] prover and a relatively minimal verifier circuit of about 10,000 multiplication gates [33]. Figure 1 presents an overall overview of the folding-based IVC structure in Nova.

The `Nova-rust` library [10] fully implements the Nova proving system and verifies the compressed output of IVC in the Spartan [48] to achieve compact SNARKs proof. Additionally, the implementation offers support for multiple frontends to define steps in Nova, such as Circom [18] through the nova-scotia library [11].

3 Motivation and Design Strategies

Over the past decade, various strategies have been proposed to combat disinformation in media. While these approaches may not always be directly comparable, we aim to highlight their individual strengths and weaknesses and provide the rationale behind the design of VIMz. Table 2 outlines the distinctive features and advantages of these approaches.

3.1 Possible Techniques for Media Provenance

C2PA. From a cryptographic perspective, approaches like the C2PA solution rely on digital signatures. In these methods, when a user captures a photo, the camera [1] or a specific software application (e.g., Truepic [16] on mobile phones) adds metadata and signs it. When the photo is edited using trusted software tools (e.g., Adobe Photoshop), new metadata is appended to the existing records, enabling verifiable edit history. In an ideal scenario, trusting the software to handle the signatures and private keys would make the C2PA protocol complete. However, in practice, such reliance on software can lead to severe vulnerabilities for both the prover and verifier sides [29, 35, 46]. To mitigate this, the C2PA trust model [8] suggests executing the software within a trusted execution environment (TEE), which introduces an additional layer of trust assumptions and limits performance. Furthermore, specific attacks on well-known TEEs, such as Intel SGX [41, 50, 51], can compromise the overall protocol.

ML-based detection models. Several studies focus on developing AI models that can effectively differentiate between real and artificial images. Some of this work targets detecting manipulation artifacts or forensic traces in AI-generated media [28, 52, 53], while others explore alternative methods, such as using advanced watermarks [24, 45]. Ideally, a perfect model would distinguish between authentic and artificially generated images without relying on trust assumptions for the camera or metadata signatures. However, recent reviews [36] highlight a significant gap in accuracy and realism between generative AI models and detection models. This suggests that, even on limited datasets, such techniques are not reliable for real-world applications. Additionally, distinguishing between genuine refinements and misinformation transformations presents challenges during the training process of these models.

[5]Transparent SNARKs do not require *trusted setup*.

Table 2: Different Approaches for Media Provenance

	Tech.	Trust Assumption	Guarantee			Complexity	
			Intg.♦	Priv.♦	Acc.♦	Prove	Verify
C2PA	Dig. Sig.	Origin★ & Editor/TEE❂	✓	✗	✓	low	low
ML	ML/AI	Model & TEE❂	N/A	✓	✗	N/A	high
VC	SNARKs	Origin★	✓	✓	✓	high	low

♦ *Intg.*, *Priv.* and *Acc.* stand for *integrity*, *privacy*, and *accuracy* assurance, respectively. ★ The original image is untampered. ❂ At some point, the prover or the verifier require trusted execution environments (TEE).

Verifiable Computation (VC). This approach aims to achieve proofs of authentic image refinements on an original source without requiring trust in the prover or the proving mechanism. To achieve this, the proofs must incorporate a opening of a commitment that binds the witness to the original source, such as a hashing method. This ensures that the authenticity of the refined image can be proven without ever revealing the original image to any party. To further enhance this approach, each proof can also incorporate computations of creating a new commitment for the transformed image. This means that the proofs not only prove knowledge of an opening for the previous commitment (for the original image), but also calculate a new commitment for the transformed image. While this increases the complexity on the prover side, it allows chaining proofs and transformations, thus reducing overall verification complexity. Theoretically, VC-based approaches hold the potential for ultimate accuracy in detecting artificially generated media while preserving the confidentiality of the original source. However, the major hurdle lies in the high complexity of the prover, which limits the practical adoption of this method. We note that the assumption of trustworthy signatures on the original source is not limited to photographed images. Different digital media content sources, such as generative models, can also sign the images they produce, demonstrating that the VC approach is not limited to specific hardware or software.

3.2 Selection Rationale of Proof System

As shown in previous studies [30], the main limitation of VC-based approaches is the memory complexity, especially on the prover side. This is because the entire original image is treated as witness data for the proofs, and the proof system requires access to it as a whole. However, we noticed that certain types of proof systems can iterate over the witness data and verify the entire witness incrementally, i.e., IVC. Folding schemes, such as Nova [33] or HyperNova [32], are efficient for achieving IVC in strictly recursive functions, where the prover can only apply the exact same function in each iteration. In these schemes, the time complexity for the prover at any step i is proportional to the size of the function applied at that step and the

total number of functions l, and the proof size remains independent of the number of iterations [32]. As discussed in the rest of this section, we argue that a majority of authentic and permissible transformations in photography and the news industry [7] can potentially be implemented iteratively.

3.3 Compatibility with C2PA

Our approach, like C2PA [9], relies on the assumption that a trustworthy entity, such as a tamper-proof camera, signs the original image. This assumption is currently practical, with companies like Sony [1] supporting signatures in professional camera line-ups and applications like Truepic [16] claiming trustworthy handling of camera modules in smartphones. Recent advancements in device attestation protocols [23] further support the achievement of verified signatures on the original image.

However, VIMz can be seen as an updated version of C2PA, where the requirement for trusted editors is eliminated. This removes restrictions on the proving or verifying software/environment, making VIMz far easier to adopt and integrate into available frameworks compared to C2PA. Consequently, VIMz enables image verification without disclosing the editor's identity, thereby preserving editor privacy. Furthermore, VIMz preserves the confidentiality of the original image and the identity of the image signer (creator/owner) while securely verifying the signer's authenticity through membership proofs (Section 4.4).

4 VIMz Protocol

This section begins by overviewing our trust model and security assumptions, followed by an overview of the VIMz protocol. We then define a folding-based circuit design for proofs of image transformation and analyze its soundness.

4.1 Target Application and Security Goals

We target scenarios where images undergo edits while retaining verifiable authenticity, which is essential for applications like journalism, social media, and intellectual property management. A typical use case envisions a trusted entity (e.g., camera) signing a high-resolution image at the moment of capture/creation for authenticity, followed by an editor refining the image through transformations such as brightness adjustments. The goal is to ensure the correctness of these edits while safeguarding privacy by linking the final edited image to the original version without exposing intermediate edits, the editor's identity, or the signer's public key, while being authenticated anonymously. Therefore, the desired system should guarantee: (1) the integrity of all transformations, (2) secure linkage of the final edited image to the original version, (3) while preserving the privacy of intermediate edits and protecting the editor's workflow. Additionally, (4) the editor's identity and (5) the original image signer's public key must remain anonymous to the verifier. Finally, the system should (6) efficiently handle high-resolution images and multiple transformations, ensuring practicality for real-world applications.

4.2 Trust and Adversary Model

The trust model in our protocol relies on the basic assumption that the original image is untampered. This assumption is upheld through a valid signature associated with the image. The signature is generated either by a tamper-proof camera (like the Sony Alpha 7 IV camera [1] or the Truepic Lens SDK for mobile devices [16]) or an trusted entity. This assumption aligns with prior work [9, 30, 42] and forms a critical foundation for any protocol in this area. However, unlike the trust model of C2PA [9], we do not necessitate additional trust assumptions. Both the editor and the storage components in our protocol are considered untrusted. Precisely, anyone can publicly verify the proofs without prior knowledge of the original image beyond its public signature.

Adversarial Model. We consider a PPT adversary $\mathcal{A}$ with the capability to eavesdrop, intercept, or manipulate any number of messages. We also assume that the adversary can compromise the functionality of any software, including VIMz, thereby gaining full control over it. However, the following cryptographic tools remain secure under any PPT adversary:

- **Collision-resistance hash functions:** Such as Poseidon, which we specifically utilize [26].
- **Digital signature schemes:** Such as ECDSA or EdDSA.
- **Nova and Spartan proving systems:** Forging a false proof in such systems is not possible by a PPT adversary [33, 48].

4.3 Overview of the Protocol

VIMz allows users to demonstrate that specific refinements have been applied to an original image α (with hash h_α) to produce image β (with hash h_β). Figure 2 presents the high-level overview of the protocol that consists of three phases:

- **Commitment Phase:** A trusted entity (such as a camera or an application like Truepic) signs the hash of the original image with its authenticated key. This signature serves as the commitment to the image α.
- **Proving Phase:** The prover, with access to the original image, applies a set of transformations and generates proofs for each transformation. The prover then sends the proofs along with the final transformed image to the verifier.
- **Verification Phase:** The verifier checks correctness of the original image's signature using verified public keys of the trusted entity and validates the proof of transformed image.

For the rest of this section, we focus on the proving phase and analyze the security of the proofs generated by VIMz. In Appendix F, we design a C2PA-compatible marketplace on top of this protocol.

Definition 4.1. Let h_α and h_β be the hashes of images α and β, respectively. We set $U_{out} = \{h_\alpha, h_\beta\}$ and consider U_{in} to be some configuration parameter for the image transformation function f_T. let $S[f_T, U_{in}, U_{out}]$ represent a statement with public values f_T, U_{in}, and U_{out}, proving that:

$$S\left[f_T, U_{in}, U_{out}\right] = \Big\{\exists\, \alpha, \beta \;\Big|\; \beta = f_T(\alpha, U_{in}) \text{ and } h_\alpha = H_\phi(\alpha) \text{ and } h_\beta = H_\phi(\beta)\Big\} \tag{1}$$

The statement in Definition 4.1 benefits from two main principals:

(1) It can be sequentially chained for multiple transformations performed on an original source, resulting in a final refined image. In this process, the output of each transformation serves as the input for the next one.

(2) The original image, as well as any intermediate images in the transformation chain, can be kept private. The prover is only required to reveal the final image to the verifier.

4.4 Anonymization of the Image Signer

The protocol outlined in Figure 2 provides the option to ensure the privacy of both the original image signer and the prover. In VIMz, proofs are securely bound to the original (α) and transformed (β) images solely through their hashes. Consequently, when proving the relationship between the two images, there is no association with the identities of the prover or the original signer. This design removes the need for the prover to authenticate their identity, as verification focuses exclusively on the validity of the proof, not on the identity of the prover.

To anonymize the identity of the original image signer, we assume the existence of a commitment to a set of n verified public keys, which was established during a secure ceremony between the manufacturer of authenticated cameras and some trusted authorities. We assume the commitment is in form of a Merkle tree with root $\mathcal{T}$ and a height of $\lceil \log n \rceil$. Instead of disclosing the sig_α to the verifier or any other party, the prover would generate a zkSNARKs proof to demonstrate the following statement.

Definition 4.2. Assume that all verified public keys are committed to a Merkle tree with root $\mathcal{T}$, where each public key pk_i is hidden using a random nonce r_i. We define the function $O_p : \mathbb{Z}_q^{3+\lceil \log n \rceil} \to \mathbb{B}$ as follows: $O_p(pk_i, r_i, \mathcal{T}, MP_i)$ verifies whether the combination of pk_i and r_i is a correct opening of a commitment that belongs to a Merkle tree with root $\mathcal{T}$, using the Merkle path MP_i. We also define $\mathcal{V}_{sig} : \mathbb{Z}_q^3 \to \mathbb{B}$ such that $\mathcal{V}_{sig}(m, sig, pk)$ verifies signature sig against message m and the public key pk. Let $S[h_\alpha, \mathcal{T}]$ represent a statement with public values $\mathcal{T}$ and $h_\alpha = H_\phi(\alpha)$, demonstrating that:

$$S[h_\alpha, \mathcal{T}] = \{\exists\ sig_\alpha, MP_i, r_i, pk_i \mid \mathcal{V}_{sig}(h_\alpha, sig_\alpha, pk_i) = 1 \text{ and } O_p(pk_i, r_i, \mathcal{T}, MP_i) = 1\} \quad (2)$$

Theorem 4.3. *The statement in Definition 4.2 ensures that the image α is signed by an authentic public key, while preserving the anonymity of the signer identity.*

Proof Sketch. The correctness of Theorem 4.3 is based on the collision resistance of H, the security of the digital signature scheme, the binding and hiding properties of the commitment scheme, and the soundness and zero-knowledge properties of the underlying zkSNARKs. Since h_α and $\mathcal{T}$ are public inputs of S, a valid claim successfully binds h_α to $\mathcal{T}$. The soundness property of the zkSNARKs ensures that h_α is correctly signed by pk_i, which is an authentic public key given that the prover provides a valid opening of the commitment using MP_i as the witness. Furthermore, due to the zero-knowledge property of the zkSNARKs, no information about the auxiliary inputs of the statement (such as pk_i, MP_i, sig_α, and r_i) is revealed to any parties. □

4.5 Folding-based Proofs of VIMz

Our goal is to provide proofs for ensuring both the accurate execution of transformations and the computation of hashes for the original and transformed images to uphold integrity. However, as demonstrated by previous work [30], this approach leads to complex circuits and computationally demanding proof generation. To address this challenge, we employ a row-by-row folding technique for each image, leveraging the IVC in Nova.

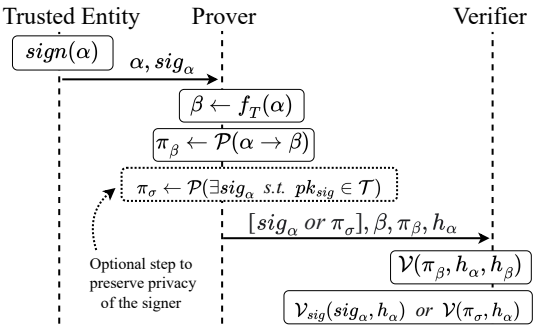

Figure 2: Overview of the protocol.

Figure 3 illustrates the proposed method for row-by-row traversal to validate image transformations within VIMz. During this traversal, cumulative hash values up to each step (h_α^i and h_β^i) are passed to the subsequent step to ensure the integrity of auxiliary inputs. The final step (step n) produces the hash of both the original and transformed images, denoted as h_α^n and h_β^n respectively. Figure 4 depicts the dataflow for constructing h_α^i in VIMz. Since each step must adhere to the same behavior (due to the folding constraint), the hash result of the first row ($H_\sigma(\alpha_1)$ in Figure 4) must also be hashed with a value from the previous state. Therefore, we set the initial given hash value as 0. The calculation of h_β^i follows the exact same method as h_α^i.

Some configuration values, such as the *contrast* adjustment factor or the starting point position in *cropping*, demand additionally publicly verifiable data. This supplementary data is denoted as u^i in Figure 3.

To provide further clarification and adhere to the original notation of Nova [33], we redefine the following symbols: $\forall i \in \mathbb{N} : \omega_i = (\alpha_i, \beta_i)$ and $z_i = U_{in}^{i+1} = U_{out}^i$. Additionally, we define a function to better describe the computations done in each step as follows:

Definition 4.4. A ***folding-friendly verifiable image transformer*** is a function with three inputs. This function can be applied in each step of the folding scheme as illustrated in Figure 3, as follows:

$$\mathcal{F}_T : (\mathbb{Z}_{256}^{m \times 3}, \mathbb{Z}_p, \mathbb{Z}_p) \to (\mathbb{Z}_{256}^{m' \times k'}, \mathbb{Z}_p, \mathbb{Z}_p)$$
$$\mathcal{F}_T(\alpha_i, h_\alpha^{i-1}, h_\beta^{i-1}) = \begin{cases} \beta_i = f_T(\alpha_i) \\ h_\alpha^i = H(h_\alpha^{i-1}, H_\sigma(\alpha_i)) \\ h_\beta^i = H(h_\beta^{i-1}, H_\sigma(\beta_i))) \end{cases} \quad (3)$$

For simplicity, we begin by formalizing a transformation that does not require any configuration data. This means we omit u^i values shown in Figure 3. Thus, a valid VIMz proof's public input should only contain two zeros: $\{0, 0\}$, and the resulting public output should be $\{h_\alpha^n, h_\beta^n\}$.

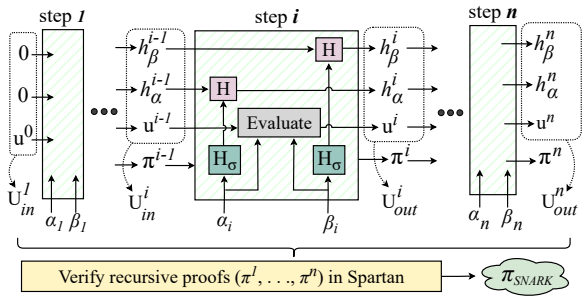

Figure 3: Folding steps of VIMz.

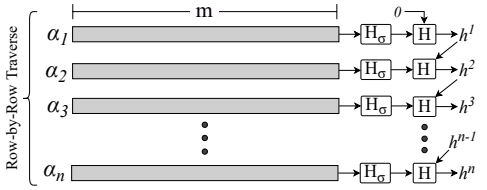

Figure 4: Calculating the hash in a row-by-row traversal.

We reduce the knowledge soundness of VIMz proofs to two fundamental assumptions: 1) the soundness property of the IVC scheme [33], and 2) the collision resistance of the Poseidon hash function [26]. Let β' represent the transformed image finally revealed by the PPT adversary $\mathcal{P}^*$, and let β denote the truly transformed image from the original source α. Therefore, we can compute the probability that an adversarial prover breaks the soundness of proofs generated by VIMz as follows:

$$Pr\left[\begin{array}{l|l} \Big((\alpha' \neq \alpha \wedge H_\phi(\alpha') = h^n_\alpha) & pp \leftarrow \mathcal{G}(1^\lambda) \\ \vee (\beta' \neq \beta \wedge H_\phi(\beta') = h^n_\beta) & \alpha', \beta', \mathcal{F}_T, (z_0, z, \Pi) \leftarrow \mathcal{P}^*(pp; \rho) \\ \vee \ (z_0 = \{0,0\} & (pk, vk) \leftarrow \mathcal{K}(pp, \mathcal{F}_T) \\ \wedge \ z \neq \{h^n_\alpha, h^n_\beta\})\Big) & (\omega_0, \ldots, \omega_{n-1}) \leftarrow \mathcal{E}(pp, z_0, z; \rho) \\ \wedge \ \mathcal{V}(vk, n, z_0, z, \Pi) = 1 & z_i \leftarrow \mathcal{F}_T(z_{i-1}, \omega_{i-1}) \ \ \forall i \in \mathbb{N}_n \end{array}\right] \quad (4)$$

Theorem 4.5. *The probability of a PPT adversary breaking the soundness of VIMz proofs (Equation 4) is negligible.*

Proof sketch. Arguing that the above probability is negligible in the PPT adversarial model is straightforward. This follows from the collision resistance of the hash function H and the soundness property of Nova. Our argument proceeds in two parts: If the adversary submits a valid proof with valid public parameters but manages to verify with either $\beta' \neq \beta$ or $\alpha' \neq \alpha$, such that $H_\phi(\beta') = H_\phi(\beta) = h^n_\beta$ or $H_\phi(\alpha') = H_\phi(\alpha) = h^n_\alpha$. In either case, the adversary must find a collision in H, which has a negligible probability due to the collision resistance of H. Alternatively, if the adversary manages to successfully verify a malformed proof Π' using public parameters other than $z = \{h^n_\alpha, h^n_\beta\}$ and $z_0 = \{0,0\}$, the probability of this for a PPT adversary is negligible according to the soundness property of the employed IVC scheme. Therefore, the overall probability of a PPT adversary breaking the soundness of proofs generated by VIMz is negligible. □

Note on Supported Transformations. We realize that image transformers can be classified based on their compatibility with row-based folding iteration. Therefore, we label one group as *"folding-friendly"* transformations that rely on local pixel values or their immediate neighbors (the rows above and below) for computation. Examples include convolution-based transformers (e.g., blur), grayscale, contrast, brightness, crop and resize (using interpolation). These transformations fit well within a row-by-row processing model, as the results depend only on nearby pixels, meaning the same transformation function can be applied repeatedly across the image rows without needing information from distant parts of the image. In contrast, *"non-folding-friendly"* transformations —such as *rotation*, *affine transformations*, and *shear*— introduce a challenge with row-by-row processing. Although these transformations could theoretically be applied row by row, the issue lies in the spatial relocation of pixels in the output. For example, in *rotation*, pixels from a single row may be relocated to entirely different positions across the image, but the row-based function can only generate output row by row. This mismatch makes these transformations unsuitable for this approach, as they require a more global understanding of pixel positions to ensure accurate evaluations.

However, the majority of common image transformations are folding-friendly. As shown in Section 5.1, VIMz can efficiently prove all permitted image manipulation according to the news industry standards [7]. Additionally, most widely-used image enhancements in applications and social media, such as convolution-based transformations and color-space mappings, are folding-friendly and efficiently supported by VIMz.

5 Implementation

In this section, we detail the design and implementation of an efficient prover based on folding schemes. We note that while VIMz primarily focuses on proving image transformations, other types of data with a bounded two-dimensional, iterable structure, such as computations over large matrices or convolutional neural networks, can also be adapted for this approach.

Figure 5 illustrates the overall architecture of the VIMz prover. To support standard image transformations, we have developed a Python interface that allows a prover to apply their desired list of transformations to the original source. The software then generates suitable inputs for the circuits based on Nova's folding scheme. The final π_{SNARK} is a valid proof of the statement in Equation 1. We use Nova to fold proofs of media authenticity and the Nova-Scotia [11] to define steps for Nova in Circom language [18]. For more details on the software and specific commands to execute the proof generation properly, refer to the Artifact Appendix[6].

5.1 Details of Circuit Design

5.1.1 ***Grayscale.*** The *grayscale* filter is a process that transforms the color planes of an image into the gray spectrum, as depicted in the following equation:

$$gray = (\alpha^R * 0.299) + (\alpha^G * 0.587) + (\alpha^B * 0.114) \quad (5)$$

[6]VIMz is available as open-source in: https://github.com/zero-savvy/vimz

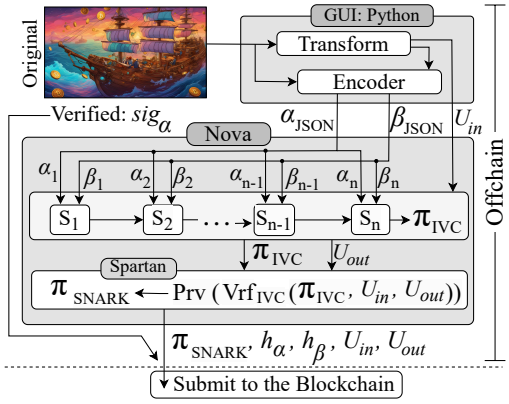

Figure 5: Overview of the VIMz prover.

Applying the *grayscale* effect to the image is straightforward. As illustrated in Figure 3, the transformed image is evaluated row-by-row against the original. Algorithm 1 outlines the details of the i-th step during the iterations for evaluating the *grayscale* transformation. The algorithm takes two types of inputs, private and public. The private inputs α_i and β_i contain pixel values of the i-th row of the original and transformed image, respectively. On the other hand, public inputs h_α^{i-1} and h_β^{i-1} represent the cumulative hash results of the original and transformed images, respectively, up to the $(i-1)$-th step.

Algorithm 1: Grayscale (Step i)

Public Input : $h_\alpha^{i-1}, h_\beta^{i-1}$
Private Input : α_i, β_i
Public Output : h_α^i, h_β^i

1 **for** $j : 0 \rightarrow len(\alpha_i)$ **do**
2 `/* Multiplied by 1000 to handle decimal values in` $\mathbb{F}_q$`, e.g., 0.299 → 299. */`
3 $val \leftarrow (\alpha_{i,j}^R * 299) + (\alpha_{i,j}^G * 587) + (\alpha_{i,j}^B * 114)$
4 **assert** $1000 > |val - \beta_{i,j} \times 1000|$
5 $h_\alpha^i \leftarrow H(h_\alpha^{i-1} | H_\sigma(\alpha_i))$
6 $h_\beta^i \leftarrow H(h_\beta^{i-1} | H_\sigma(\beta_i))$

An essential consideration in Algorithm 1 is the multiplication by 1000 to ensure accurate decimal calculations within the integer format of elements in the field $\mathbb{F}_q$ in equation 5. To compare `val` with the given grayscale values ($\beta_{i,j}$ in the algorithm), we assert that the distance between `val` and $\beta_{i,j} \times 1000$ is less than 1000. The allowance for a distance larger than 0 (but less than 1000) accounts for rounding errors when capping the grayscale value to an integer. Ultimately, the circuit updates the cumulative hash values by digesting its private inputs (α_i and β_i), preparing them for the subsequent step (lines 5 and 6).

5.1.2 ***Contrast And Brightness Adjustments.*** The approach for realizing brightness or contrast adjustments is similar to that for grayscale. To avoid repetition, we do not discuss them here, but provide complete algorithms in Appendix A and Appendix B. It's important to note that applying contrast or brightness adjustments can result in pixel values exceeding the valid range (0-255). Therefore, it is essential to cap the result within this standard range, as depicted in Equation 6 below.

$$cap_0^k(x) = Min(Max(0,\ x),\ k) \tag{6}$$

To implement `cap` functionality over $\mathbb{F}_q$, we need to start by determining the sign of the resulting value. If the sign was negative, the value must be capped at zero. However, in $\mathbb{F}_q$, $-x$ is equivalent to $q - x$, and there is no inherent sign. To identify the sign, a comparison is made directly with $q - x$. If $q - x$ is greater than x, the value is positive; otherwise, it is negative and should be capped at zero. Additionally, when the value is positive, a comparison with 255 is necessary to cap it at 255 if it exceeds this value. Appendix C provides circuit-level code for implementing `cap` in VIMz.

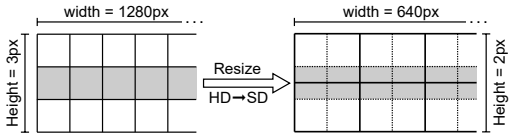

Figure 6: Implementation of *resize* in Nova.

5.1.3 ***Resize.*** Resizing an image differs from other transformations as it cannot be executed exactly row-by-row. Depending on the vertical resize ratio $\frac{\text{height}(\alpha)}{\text{height}(\beta)} = \frac{k}{k'}$, a set of k rows from the original image compresses into k' rows, where $k' < k$. For instance, when resizing an HD image to SD resolution, 720 rows of the original image compress into 480 rows, resulting in a simplified resize ratio of $\frac{720}{480}$ or $\frac{3}{2}$. Consequently, in each step of the designed circuit for resizing, three rows of the original image are evaluated against two rows of the transformed image[7]. Figure 6 represents resizing an HD to SD resolution in this setting. Given that the original and destination sizes are fixed during the benchmarks, constant weights can be used for bilinear interpolation in the *resize* algorithm.

Algorithm 2 provides the abstract functionality implemented in each step of the *resize* transformation. In the down-scaling based on bilinear interpolation, at most, four pixels from the original image have an effect on a pixel in the resized image. Lines 5 and 6 of the algorithm calculate the resulting pixel values based on the weights in the bilinear interpolation down-scaling. Similar to previous transformations, the inability to use float values in $\mathbb{F}_q$ necessitates asserting that the resulted weighted value `val` has a distance of less than 6 from $\beta_{i,j} \times 6$ (line 7 of the Algorithm 2).

5.1.4 ***Crop.*** When applying *crop* in a row-by-row traverse, it becomes crucial for each step to determine whether its corresponding row (α_i) falls within the crop area. Figure 7 provides a high-level overview of this approach. Specifically, if the current step is within the crop area, the *crop* evaluation takes effect; otherwise, it needs to be skipped as illustrated in Algorithm 3.

Unlike the rest of the transformations, this algorithm has only one private input, α_i, in each step. This is because the values of the cropped image must exactly match the one for the corresponding

[7] *resize* circuit requires less steps. e.g., HD to SD requires only $\frac{720}{3} = \frac{480}{2} = 240$ steps.

Algorithm 2: Resize (Step i): HD→SD

Public Input : h_α^{i-1}, h_β^{i-1}
Private Input : $\alpha_{[i..i+r_\alpha]}$, $\beta_{[i..i+r_\beta]}$
Public Output: h_α^i, h_β^i

1 /* $r_\alpha = 3$, $r_\beta = 2$*/
2 **foreach** $c \in [R, G, B]$ **do**
3 **for** $i : 0 \to r_\beta$ **do**
4 **for** $j : 0 \to len(\beta_i)$ **do**
5 $weight \leftarrow 2 - i$
6 $val \leftarrow (\alpha^c_{i,j*2} + \alpha^c_{i,j*2+1}) * weight$
 $+(\alpha^c_{i+1,j*2} + \alpha^c_{i+1,j*2+1}) * (3 - weight)$
7 **assert** $6 > |val - \beta^c_{i,j} \times 6|$
8 $h_\alpha^i \leftarrow H(h_\alpha^{i-1} | H_\sigma(\alpha_i))$
9 $h_\beta^i \leftarrow H(h_\beta^{i-1} | H_\sigma(\beta_i))$

pixels from the original image. The algorithm takes three additional public inputs besides h_α^{i-1} and h_β^{i-1}. The i_{row} indicates the index of the current row, while x and y represent the column and row of the starting point of the crop, respectively.

Line 3 of the algorithm computes the hash value of the cropped area along the horizontal dimension. This operation involves a dynamic subset selection from the array α_i based on runtime inputs provided to the circuit. In the R1CS setting, memory access must be predetermined at compilation. A possible workaround is to fix the crop location before compiling the circuit, treating it as a constant, i.e., x and y will be fixed as part of the circuit's template.

Lines 4 to 8 check whether to use the h_{temp} value based on the index of the row. This way, h_β will only be updated if the current row is within the crop area. Similar to the dynamic array selection, traditional `if-else` statements are not directly realizable in R1CS setting. Therefore, we must compute all potential conditional results and subsequently select the appropriate one based on the inputs. The actual implementation of the `if-else` statement in Algorithm 3 involves the code described in Appendix D.

5.1.5 ***Selective Crop.*** A drawback of the previous approach, which involves fixing x and y, is that it requires a unique verification key (vk) for each crop location. While this isn't a major issue in most cases and is standard practice in related works [22, 38], as verifying or generating the vk by compiling the public ZK circuit is relatively inexpensive on the verifier's side, it becomes problematic in constrained environments like when a Solidity-based smart contract serves as the verifier.

To achieve a dynamic subset selection of size $width_{crop}$ from the given row, we require $width_{crop}$ times `multiplexers` each supporting an input size of $|width_\alpha - width_{crop}|$ values. These multiplexers enable selection of all possible subsets for a given row, then determine which of these subsets should be propagated based on the given x value.

5.1.6 ***Convolution-based Transformations.*** Two widely used effects based on convolution are *blur* and *sharpness* adjustment. Algorithm 4 provides an abstraction of the designated circuit for

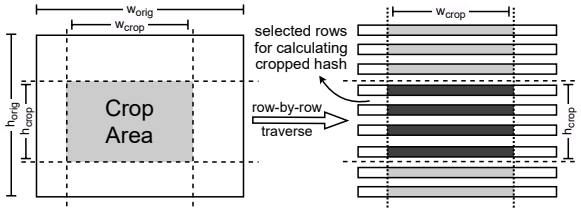

Figure 7: *Crop* in a row-by-row setup.

Algorithm 3: Selective Crop (Step i)

Public Input : h_α^{i-1}, h_β^{i-1}, i_{row}, x, y
Private Input : α_i
Public Output: h_α^i, h_β^i, i_{row}, x, y

1 $width_{crop} \leftarrow width(Crop\ Resolution)$
2 $height_{crop} \leftarrow height(Crop\ Resolution)$
3 $h_{temp} \leftarrow H_\sigma(\alpha_i[x : x + width_{crop}])$
4 **if** $y \le i_{row} < y + height_{crop}$ **then**
5 $h_\beta^i \leftarrow H(h_\beta^{i-1} | h_{temp})$
6 **else**
7 $h_\beta^i \leftarrow h_\beta^{i-1}$
8 $h_\alpha^i \leftarrow H(h_\alpha^{i-1} | H_\sigma(\alpha_i))$
9 $i_{row} \leftarrow i_{row} + 1$

convolution. It is crucial to note that while calculations are done row-by-row, values from k previous and next rows are needed to apply a kernel matrix κ of size $(2k + 1) \times (2k + 1)$.

Algorithm 4: Convolution (Step i)

Public Input : h_α^{i-1}, h_β^{i-1}, κ, $h_{\alpha_{[(i-k)\to(i+k-1)]}}$
Private Input : $\alpha_{[(i-k)\to(i+k)]}$, $\beta_{[(i-k)\to(i+k)]}$
Public Output: h_α^i, h_β^i, $h_{\alpha_{[(i-k+1)\to(i+k)]}}$

1 $weight \leftarrow \sum_{p,j:0\to(2k+1)} \kappa_{p,j}$
2 **foreach** $c \in [R, G, B]$ **do**
3 **for** $j : 0 \to len(\alpha_i)$ **do**
4 $val \leftarrow 0$
5 **for** $m : 0 \to len(\kappa)$ **do**
6 **for** $n : 0 \to len(\kappa)$ **do**
7 $val += \alpha^c_{m,j+n} \times \kappa_{m,n}$
8 $val \leftarrow \mathbf{cap}_0^{255\times weight}(val)$ `///optional`
9 **assert** $weight > |val - \beta^c_{i,j} \times weight|$
10 **for** $m : 0 \to 2k - 1$ **do**
11 **assert** $H_\sigma(\alpha_{i-k+m}) == h_{\alpha_{i-k+m}}$
12 $h_{\alpha_{i-k+m+1}} \leftarrow H_\sigma(\alpha_{i-k+m+1})$
13 $h_\alpha^i \leftarrow H(h_\alpha^{i-1} | H_\sigma(\alpha_i))$
14 $h_\beta^i \leftarrow H(h_\beta^{i-1} | H_\sigma(\beta_i))$

To enforce consistency of input rows between steps, the common input rows hashes are verified against each other. To this end, the

first $2 \times k$ input rows are compared with values passed by the previous step, and the hashes of the last $2 \times k$ rows is passed to the next step (lines 10 to 12 of the algorithm).

5.2 Private Proofs of Authentic Image signer

The statement defined in Definition 4.2 aligns precisely with the scenario implemented in [6]. Their library provides an interface to prove membership in a group of ECDSA public keys without disclosing information about the specific public key, which was derived from an ECDSA signature. According to their benchmarks [6], proving the statement in Definition 4.2 takes only 4 seconds on a midrange laptop, demonstrating that for a given hash value, a signature exists from a key within a group of $2^{20} \approx 1$ million authentic keys. Verifying the proof requires just 300 ms, and the proof size is approximately 2 KB.

5.3 Optimization

5.3.1 ***Lossless Pixel Compression before Hashing.*** As noted in prior work [30], calculating the entire hash of auxiliary inputs adds up to the majority of constraints in the ZK circuit. To mitigate this, we propose a lossless compression technique for pixel values to reduce the number of constraints by nearly 30×. The primary concept involves packing as many pixel values as possible within a field element.

Using Pallas/Vesta curves in our setup, each field element $e \in \mathbb{F}_q$ can range from 0 to $q-1$, where $q = 2^{254} + 455 \ldots 353$. Consequently, each field element can carry up to 254 bits of information. Given that every RGB value spans from 0 to 255 (equivalent to 8 bits or one byte), we can concatenate up to 10 complete pixel values (three R/G/B values for each pixel) that translate to a valid number in $\mathbb{F}_q$ as is shown in Figure 8.

Now, in order to validate the integrity of an image, we require 30× less number of hashes over the field $\mathbb{F}_q$. However, this method necessitates *decompression* before validating the transformed pixel values. Appendix E implements the *decompression* circuit in Circom.

5.3.2 ***Parallel Proof Generation.*** The implementation of VIMz leverages the memory and space efficiency of the Nova proving system. Our experimental results, detailed in Section 6, demonstrate a maximum memory peak of only 3.2 GB of RAM when proving transformations on an HD-resolution image, while also computing the hash of both the original and transformed images. This underscores the practicality of VIMz, even on consumer-level commodity hardware and enables simultaneous execution of multiple instances of VIMz, allowing for parallel proof generation. Figure 9 illustrates a scenario with three distinct transformations applied to an image. VIMz can concurrently run multiple instances, proving each effect separately but in parallel. Calculating the hash of both input and transformed images in each step establishes a chain of hashes (e.g., h_α, h_{cont}, h_{gray}, and h_{res} in Figure 9) that ensures integrity between steps. The final proof comprises all of the hashes and their corresponding partial proofs (π, π', and π''). Section 6 includes experimental results for running multiple instances of VIMz in parallel on various platforms (Table 5).

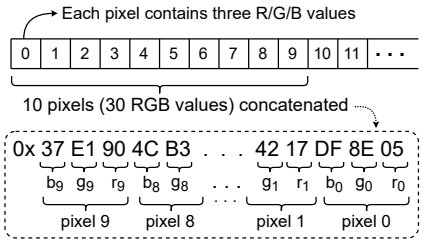

Figure 8: Lossless compression of pixel values.

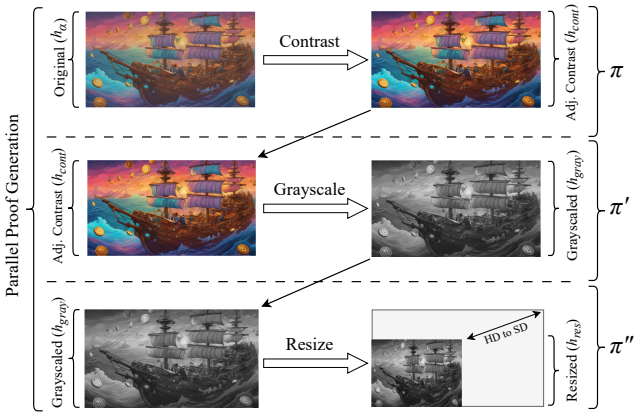

Figure 9: Proving multiple transformations in parallel.

5.4 Configurability

We denote the hyperparameters of transformations as part of the U^i_{in} and U^i_{out} values in the folding scheme. In the current implementation, several transformations have at least one hyperparameter, such as *contrast* or *brightness* factors, or the starting position (x, y) in *selective crop*. Each folding step passes these values to the next step unchanged. As these hyperparameters are part of U^i_{in} and U^i_{out}, they will ultimately be visible to the verifier as public inputs and outputs of the VIMz proofs. Consequently, the number of hyperparameters affects prover and verifier complexity; however, this overhead is negligible compared to the overall computation and communication complexity.

In terms of support for new transformations, defining them in VIMz is straightforward. Depending on the computations, a transformation may require a few hyperparameters, which will become part of U^i_{in} and U^i_{out}. The remaining step involves defining the calculation in `Circom` for one iteration. For transformations requiring access to neighboring rows (e.g., convolution-based transformations like blur and sharpness), techniques outlined in Algorithm 4 can be applied to enforce integrity of shared rows between steps. For more technical details on adding new transformations in VIMz, refer to Appendix G.

6 Experimental Result

To showcase the practicality of VIMz, we have executed multiple benchmarks across two set of hardware, a midrange Dell Latitude

Table 3: Experimental Setup Configuration

		Dell Latitude 5531	Dedicated Server
CPU	Model	Intel® Core i5-12500H	AMD Ryzen 9 7950X3D
	Freq.	3.3~4.5 GHz	4.2~5.7 GHz
	Cores✳	12	16
	Cache♦	18 MB	145 MB
Memory		16 GB	128 GB
SSD		512 GB	2 TB
OS		Ubuntu 22.04 LTS	

✳ Number of physical cores. ♦ Total cache size (L1+L2+L3).

Table 4: Proving HD Resolution (1280×720)

	Time (s)				Peak Memory
	midrange Laptop		Server		
Trans.♦	Key. Gen.	Proving	Key. Gen.	Proving	
Crop★	3.8	187.1	3.5	133.0	0.7 GB
Resize★	11.5	187.0	6.6	135.7	2.5 GB
Contrast❂	11.7	479.4	6.5	371.7	2.4 GB
Grayscale❑	8.2	279.6	3.7	240.6	1.3 GB
Brightness❂	11.3	474.0	6.5	372.5	2.4 GB
Sharpness	11.8	614.1	6.8	455.8	2.8 GB
Blur	11.5	555.3	6.6	406.0	2.5 GB
Sel. Crop★	11.9	914.5	7.4	898.8	3.2 GB

♦ All measurements include proving the accurate execution of transformations, alongside computing the hash values of both the α and β to achieve trustless and private proofs of integrity. ★ HD to SD.
❂ *Contrast* and *brightness* adjustments with an accuracy of 0.1.
❑ *Grayscale* transformation with an accuracy of 0.001.

laptop versus a dedicated server. Table 3 outlines the systems used for benchmarking in our experiments.

6.1 Prover Complexity

All of our benchmarks report the overhead of proving the complete proofs of VIMz (proving both the transformation along with the hash of the input and output images).

HD Resolution. Table 4 provides the proof generation performance for various transformations. Key takeaways include VIMz's low memory consumption, fast proving process and a very short key generation time. As indicated by the results, generating complete proofs[8] for a single HD transformation requires a maximum of 3.2 GB of RAM, which is practical even on constrained systems.

Parallel Proof Generation. Utilizing parallelization technique, as discussed in Section 5.3.2, enables us to boost the performance further with minimal memory overhead. Table 5 provides performance metrics for executing parallel proof generations. In this setup, chained transformations are proven concurrently, with the longest proof generation time reported as the *Max* time, while the *Avg* time is computed as $\frac{Max\ \text{time}}{\#\ \text{of transformations}}$. Due to parallel execution, overall memory consumption exceeds that of a single transformation. However, as VIMz instances share libraries, memory usage is less than the sum of memory requirements of Table 4.

[8]Proving transformation along with hashes of both α and β.

Table 5: Parallel Proof Generation for HD (1280×720)

Transaction List★	Time (s)				Peak Memory♣	
	Laptop		Server		Laptop	Server
	Max	Avg	Max	Avg		
C-G	573.2	287	378.3	190	3.0 GB	2.8 GB
C-S	801.0	400	482.6	242	4.3 GB	4.4 GB
C-G-R	672.4	225	395.1	132	5.2 GB	5.0 GB
C-S-R	883.1	295	501.7	168	6.1 GB	6.0 GB
C-S-R-G	1013.8	254	532.4	134	7.1 GB	6.7 GB
C-S-R-G-B	1234.7	247	577.3	116	8.8 GB	8.6 GB

★ *C*, *G*, *R*, *S*, and *B* stand for Contrast, Grayscale, Resize, Sharpness, and Brightness, respectively. The underline indicates the transformation with the highest prove time, i.e., resulting in the *Max* time.
♣ The peak memory reported here represents the total (summation) peak memory usage across all instances of VIMz executed in parallel.

A notable observation from Table 5 is that as the number of transformations performed in parallel increases, the overall performance improves significantly on average. For instance, compared to the results for a single transformation from Table 4, the overall average performance increases by up to 3.5×. This suggests that in the context of integrating VIMz with real-world editors such as Photoshop or GIMP, the proof generation process for each transformation could be initiated in the background immediately.

Another takeaway is the fact that unlike the scenario of proving a single transformation, the performance gap between the laptop and the server widens when proving in parallel. Here, the advantage of larger cache size and higher number of physical cores in Ryzen 9 CPU demonstrates its superiority over the laptop CPU.

4K Resolution. Table 6 provides performance measurements of prover for 4K resolution. These results imply that it is possible to provide proofs of authentic image manipulations using folding-based zkSNARKs even for large images. We note that the same parallelization method can also be applied to the 4K proving scenario that will further boost performance.

6.2 Verifier and Communication Complexity

Table 7 provides proofs size and the verification time, which correspond to the size and verification time of a Spartan SNARKs proof [48]. It takes less than a second on the laptop and to verify VIMz proofs. VIMz maintains nearly constant proof sizes around 10 to 11 KB regardless of resolution. Notably, these proofs can be verified on-chain in Solidity [13, 14] and are well within Ethereum's transaction size limits. Therefore, it is possible to have a transaction containing multiple chained VIMz proofs to support building a trustless C2PA-compatible marketplace as we show in Appendix F.

7 Related Work

The rationale for employing VC in the context of media provenance is to generate secure proofs of authentic image refinements without requiring trust in the prover. By opening a commitment that binds the image to its original source (e.g., using a hash), we prove the authenticity without revealing the original image. Additionally, each proof can compute a new commitment for the transformed

Table 6: Proving 4K Resolution (3840×2160)

	Laptop		Server	
Transform.	Proving (s)	Memory	Proving (s)	Memory
Crop★	1541	2.5 GB	1210	2.8 GB
Resize★	1301	4.0 GB	974	4.7 GB
Contrast	3525	5.5 GB	2834	6.2 GB
Brightness	3384	5.4 GB	2823	6.2 GB
Grayscale	2169	3.1 GB	1596	3.1 GB
Sharpness	4599	8.0 GB	3657	8.4 GB
Blur	4097	7.4 GB	3231	7.5 GB
Sel. Crop★	7259	9.3 GB	7302	10.3 GB

★ 4K to 2K.

Table 7: Verifier Complexity

	Verification Time (s)		Proof Size (KB)
	Server	Laptop	
Crop	0.5 s / 0.5 s	0.9 s / 0.9 s	10.5
Resize	0.3 s	0.5 s	10.2
Grayscale	0.2 s	0.3 s	9.9
Cont. / Bright. / Sharp. / Blur	0.3 s	0.5 s	10.3

image, allowing for chained proofs and reduced verification complexity, albeit with a significant increase in prover complexity. Even without commitment generation for the transformed image, the high prover complexity in most of the related work posed a serious challenge for practical adoption.

Photoproof [42] pioneered this idea by demonstrating the usability of cryptographic proofs in general for verification of operations like *crop*, *flip*, or adjustments to *contrast* and *brightness*. However, their underlying proof system was not efficient enough and as a result, their experiments where limited to low resolution images of only 128, which is not practical in real world. A subsequent study in 2022, ZK-IMG [30] utilized the Halo2 [4], which is a more efficient method of achieving SNARKs based on Plonky2 [12] proof system. As a result, ZK-IMG was able to achieve higher efficiency and was able to generate complete proofs in HD resolution (1280×720). However, as elaborated in Section 3.1, the main drawback of works like [30] is that the entire original and transformed images are given to the proof system, such as Halo2, at once. This means that in practice, the prover must execute a large amount of arithmetic in one round of proving, resulting in very high memory complexity. Consequently, ZK-IMG reported a peak memory usage of more than 300 GB of RAM while generating a complete proof for images in HD resolution. In contrast, VIMz addresses the problem of high memory complexity by utilizing folding-based proof systems. VIMz's peak memory consumption is bound to the computations done in only one row of an image, achieving a far lower memory requirement of just 2.8 GB for generating complete proofs of HD image transformations. In addition to lower memory consumption, VIMz achieves up to 3× faster proving times compared to [30]. We compared VIMz to [30] in the final section of Table 8.

Another study [17] employed sumcheck protocol [20] to prove the correctness of convolution function evaluations. However, their focus was on convolution functions and did not report any analysis for other types of image manipulation. Moreover, [17] does not incorporate concrete steps for providing proofs for opening any kind of commitment to the input data and as a result does not provide proofs of integrity. Therefore, [17] does not fully implement the complete proof system required to preserve the confidentiality of the original source. In contrast, VIMz, similar to other related work [30], incorporate proofs of hash for both the original and transformed image to securely bind the zkSNARKs proofs to both images, while preserving confidentiality of the original image.

In another work [31], authors provide verifiable image redaction by grouping pixels into larger blocks, such as 16×16, to improve overall performance. In contrast to [31], VIMz proves precise (per pixel) transformation evaluations. Moreover, while [31] focuses solely on proofs of redactions, VIMz provides a wider range of image transformations, such as *grayscale*, *sharpness*, and *contrast*. We acknowledge that the method in [31], which involves increasing the granularity of the blocks, improves the prover's performance by lowering precision. However, this approach cannot be applied to most image transformations, as the commitment does not have enough data to cover all pixel values in the witness.

Two concurrent works to our paper, [22] and [38], also aim to reduce prover complexity. We compare the performance of VIMz to these works in Table 8 in terms of both prover and verifier complexity. In [22], the authors propose proving construction of a polynomial commitment for an original image source that was previously committed using a specific lattice-based hash. The new commitment can then be used to efficiently prove a transformation on the original image. The main drawback of [22] is the commitment performance, which results in either longer proving times or significantly higher memory usage compared to the competition. To provide a more detailed comparison with their hashing mechanism, we benchmark the performance of hash proofs in Table 9. Our results indicate that folding-based hash proofs are generally more efficient than lattice-based hash proofs of [22], offering up to 2× faster proving times and requiring 97% less memory.

In [38], the authors divide the image into smaller tiles to reduce prover complexity. This approach generates multiple independent proofs for each tile, necessitating separate verification processes. Both the total proof size and verification time increase linearly with the number of tiles. Moreover, implementing transformations like resizing or convolution-based operations (e.g., blur or sharpness) is challenging since each tile lacks access to neighboring pixels, leading to potential errors, as noted by the authors in [38]. In contrast, VIMz uses folding schemes to effectively handle such transformations without errors. As explained in Section 5.1 and Algorithm 4, VIMz can incorporate neighboring pixel data and verify it securely, ensuring accurate convolution implementations. In benchmarks, as shown in Table 8, for 33 MP transformations, VIMz achieves faster proving times, over 170× faster verification times, and over 90% smaller proof sizes compared to [38]. We also note that their underlying proving system is Groth16 [27], which necessitates a trusted setup for each specific circuit. Therefore, their experimental approach is impractical for real-world scenarios, as the community would need to conduct a separate MPC ceremony for every transformation and tile size they wish to support. In contrast, our proofs are transparent and do not require any trusted setup ceremony.

Table 8: Comparison with Related Work

Trans.	Work	Prover Cost: proof gen. (min)	Prover Cost: Mem. (GB)	Verifier Cost: Verify (s)	Verifier Cost: proof size (KB)	Verifier Cost: Public✢ inp. (KB)
(6632 × 4976) 33 MP						
resize★	[38]	56.5	3.4	107.7	140	>8000
	[22]❂	24×3 + 5	>57	2 + 196	150 + 5	>8000
	op [22]❂	60×3 + 5	>72	2 + 0.2	150 + 28	>8000
	VIMz	**49**	**9.5**	**0.3**	**10.5**	**0.064**
crop★	[38]	62.25	3.4	107.7	140	>8000
	[22]❂	24×3 + 8×3	>57	2 + 196	150 + 5	>8000
	op [22]❂	60×3 + 8×3	>72	2 + 0.2	150 + 28	>8000
	VIMz	**46**	**3.9**	**0.6**	**10.5**	**0.064**
(2048 × 1365) Resolution						
gray.	[38]	14.9	4.5	21.9	28	>2666
	[22]❂	4×3 + 3.37	>10	2 + 18	150 + 5	>2666
	op [22]❂	6×3 + 3.37	>10	2 + 0.2	150 + 28	>2666
	VIMz	**11.5**	**2.4**	**0.4**	**10**	**0.064**
(1280 × 720) HD Resolution						
sharp.	[30]	21.67	305.3	0.009	14.9	0.064
	VIMz	**7.58**	**2.8**	**0.3**	**10.3**	**0.064**

★ 33 MP to 2048×1365. ✢ Size of public inputs required for verification. ❂ In [22], reported proof times for commitment are for 30 MP not 33 MP. Moreover, there are two proofs required to be verified: commitment (of size 5 or 28 KB) and transformation (of size 100∼200 KB, avg=150 KB). We applied a factor of ×3 because [22] only reports results for a single color channel. Here, we opted to multiply the time rather than the RAM usage, as the latter would exceed the memory limit of their own setup.

Beyond general performance metrics and significantly smaller proof size, VIMz offers several advantages over [22] and [38]. Both of those work do not incorporate calculations of the hash value for the transformed images in their proofs. Consequently, the verifier circuit requires the entire transformed image as input, which can be up to 8 MB, as seen in Table 8. Therefore, their proof size, combined with the public inputs for the proof (i.e., the transformed image), far exceeds the limitations of common block sizes in public blockchains, e.g., Ethereum block size is around 100 KB. This makes their work incompatible for future adoptions in blockchain-based applications.

In contrast, VIMz proves the hash of the transformed image within the proof itself, generating a compact commitment for the transformed image. As a result, the verification process of the proof only involves the hash of the transformed image for verification, not the entire image data, making it more compatible with constrained verifiers. Another advantage of proving transformed image hashes within the main proof is the capability to support chained transformations. This allows multiple edits to be proven in sequence without revealing intermediate images. Additionally, it enables proving chained edits in parallel, which, as shown in Section 5.3.2, can improve the prover's performance by an additional 3.5× speedup. We note that the comparisons in Table 8 do not include the results of parallel execution of VIMz instances for a fair comparison.

Table 9: Hash proofs in VIMz Compared to Related Work

The reported results reflect proving all three RGB channels together.

	Time (s): midrange Laptop: Key. Gen.	Time (s): midrange Laptop: Proving	Time (s): Server: Key. Gen.	Time (s): Server: Proving	Peak Memory
VIMz HD❂	2.4	123	2.4	103	0.5 GB
VIMz 4K✸	6.0	724	5.3	631	1.2 GB
VIMz 33 MP✯	11.1	2661	9.4	2196	2.2 GB
VIMz 33 MP (monochrome)▲	11.0	875	9.2	743	1.9 GB
[22] 30 MP (monochrome)	–	–	-	1440	72 GB

❂ 1280×720 ✸ 3840×2160 ✯ 6632×4976 ▲ Only one color plane for the sake of comparison with results reported in [22]. Unlike the hash proof results in [22], VIMz doesn't require proving additional commitments by relying solely on image hashes in transformation proofs, which leads to overall lower memory and computational complexity.

8 Discussion and Future Work

This paper showcases the practicality of generating proofs for valid image manipulations using folding-based zkSNARKs. We present VIMz, an open-source platform that can efficiently operate on consumer-level hardware. It can prove the transformations on 8K (33MP,i.e., 100MB) images under 50 minutes, reaching to a peak memory of only 10 GB, while proof size is just around 10 KB and verification time is under 1 second. VIMz's succinct proofs enable the development of a privacy-preserving, trustless marketplace for authentic media. Appendix F provides further design details for such a marketplace. As a complete proof system, VIMz proves the integrity of both the original and edited images, as well as the correctness of the transformation without revealing intermediate images within a chain of edits—only the final result is disclosed. We further introduce a privacy-preserving option to the protocol, where the original image owner's identity remains confidential, while still enabling a trustless authenticity check of the signer.

VIMz, similar to C2PA and other VC-based methods, assumes a trustworthy signature on the original image. However, VIMz extends C2PA by eliminating the need for trusted editors, enabling easier integration into existing frameworks, while guaranteeing editor and original signer's privacy. As VIMz is built on folding schemes, it is particularly suited for folding-friendly transformations, where the final value of the resulting image is determined by neighboring pixel values. Other types of image manipulation like affine transformations or rotations, which involve shifting pixel positions, can be expensive or incompatible with VIMz's row-by-row commitment scheme. Thus, a potential future direction is to explore efficient folding-based mechanisms that support such transformations. Currently, VIMz supports a limited set of folding-friendly transformations, but this can be expanded to include more configurability with additional hyperparameters, such as dynamic convolution kernels, to accommodate arbitrary global image effects. Future work may also incorporate advancements in folding-based schemes [32, 43, 44], some of which have already been integrated into the underlying Nova protocol [43]. Finally, optimizing VIMz for greater efficiency could make it more suitable for deployment on mobile devices.

Acknowledgments

The authors would like to thank reviewers for providing us with constructive and insightful reviews. The authors used ChatGPT4 only for grammatical revision of the text in Sections 1, 3, 7, and 8 to correct any typos, grammatical errors, and awkward phrasing. This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666).

References

[1] 2022. Sony Unlocks In-Camera Forgery-Detection Technology. https://www.sony.eu/presscentre/sony-unlocks-in-camera-forgery-proof-technology. Accessed: 2024-01-08.
[2] 2023. DALL•E. https://openai.com/research/dall-e. Accessed: 2023-10-04.
[3] 2023. DeepFaceLab Library. https://github.com/iperov/DeepFaceLab. Accessed: 2023-10-05.
[4] 2023. The Halo2 zero-knowledge proving system. https://zcash.github.io/halo2/. Accessed: 2023-10-31.
[5] 2023. Midjourney. https://www.midjourney.com/home/. Accessed: 2023-10-04.
[6] 2023. Spartan-ecdsa. https://github.com/personaelabs/spartan-ecdsa. Accessed: 2023-10-04.
[7] 2023. Standards and practices in The Associated Press News. https://www.ap.org/about/news-values-and-principles/telling-the-story/. Accessed: 2023-10-04.
[8] 2024. C2PA Technical Specification. https://c2pa.org/specifications/specifications/1.2/index.html. Accessed: 2024-01-03.
[9] 2024. Coalition for Content Provenance and Authenticity (C2PA). https://c2pa.org/. Accessed: 2024-01-03.
[10] 2024. Nova High-speed recursive arguments from folding schemes. https://github.com/microsoft/Nova. Accessed: 2024-01-03.
[11] 2024. nova-scotia. https://github.com/nalinbhardwaj/Nova-Scotia. Accessed: 2024-01-03.
[12] 2024. Plonky2. https://github.com/0xPolygonZero/plonky2/. Accessed: 2024-07-26.
[13] 2024. solidity-verifier. https://github.com/lurk-lab/solidity-verifier/. Accessed: 2024-02-07.
[14] 2024. Sonobe. https://github.com/privacy-scaling-explorations/sonobe. Accessed: 2024-07-26.
[15] 2024. Stability AI. https://stability.ai/home. Accessed: 2024-01-02.
[16] 2024. TruePic: Secure Content Transparency with C2PA. https://truepic.com/. Accessed: 2024-01-03.
[17] David Balbás, Dario Fiore, Maria Isabel González Vasco, Damien Robissout, and Claudio Soriente. 2023. Modular Sumcheck Proofs with Applications to Machine Learning and Image Processing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1437–1451.
[18] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. 2022. Circom: A Circuit Description Language for Building Zero-knowledge Applications. *IEEE Transactions on Dependable and Secure Computing* (2022), 1–18. https://doi.org/10.1109/TDSC.2022.3232813
[19] Sanchuan Chen, Zhiqiang Lin, and Yinqian Zhang. 2023. Controlled data races in enclaves: Attacks and detection. In *32nd USENIX Security Symposium (USENIX Security 23)*. 4069–4086.
[20] Alessandro Chiesa, Michael A Forbes, and Nicholas Spooner. 2017. A zero knowledge sumcheck and its applications. *arXiv preprint arXiv:1704.02086* (2017).
[21] Intel Corporation. 2024. Intel® 64 and IA-32 Architectures Software Developer Manuals. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
[22] Trisha Datta, Binyi Chen, and Dan Boneh. 2024. VerITAS: Verifying Image Transformations at Scale. *Cryptology ePrint Archive* (2024).
[23] Shahriar Ebrahimi and Parisa Hassanizadeh. 2024. From Interaction to Independence: zkSNARKs for Transparent and Non-Interactive Remote Attestation. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS'24)*. San Diego, CA.
[24] Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. 2023. The stable signature: Rooting watermarks in latent diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 22466–22477.
[25] Paul Glynn. 2023. Sony World Photography Award 2023: Winner refuses award after revealing AI Creation. https://www.bbc.com/news/entertainment-arts-65296763
[26] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems.. In *USENIX Security Symposium*, Vol. 2021.
[27] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology – EUROCRYPT 2016*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 305–326.
[28] Luca Guarnera, Oliver Giudice, and Sebastiano Battiato. 2020. Deepfake detection by analyzing convolutional traces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 666–667.
[29] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2020. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy* 18, 2 (2020), 56–60.
[30] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. 2022. ZK-IMG: Attested Images via Zero-Knowledge Proofs to Fight Disinformation. *arXiv preprint arXiv:2211.04775* (2022).
[31] Hankyung Ko, Ingeun Lee, Seunghwa Lee, Jihye Kim, and Hyunok Oh. 2021. Efficient verifiable image redacting based on zk-snarks. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 213–226.
[32] Abhiram Kothapalli and Srinath Setty. 2023. HyperNova: Recursive arguments for customizable constraint systems. *Cryptology ePrint Archive* (2023).
[33] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. 2022. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*. Springer, 359–388.
[34] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. 2020. Celeb-df: A large-scale challenging dataset for deepfake forensics. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 3207–3216.
[35] Christian Mainka, Vladislav Mladenov, and Simon Rohlmann. 2021. Shadow Attacks: Hiding and Replacing Content in Signed PDFs.. In *NDSS*.
[36] Yisroel Mirsky and Wenke Lee. 2021. The creation and detection of deepfakes: A survey. *ACM Computing Surveys (CSUR)* 54, 1 (2021), 1–41.
[37] Vladislav Mladenov, Christian Mainka, Karsten Meyer zu Selhausen, Martin Grothe, and Jörg Schwenk. 2019. 1 Trillion Dollar Refund: How To Spoof PDF Signatures. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1–14.
[38] Pierpaolo Della Monica, Ivan Visconti, Andrea Vitaletti, and Marco Zecchini. 2024. Trust Nobody: Privacy-Preserving Proofs for Edited Photos with Your Laptop. Cryptology ePrint Archive, Paper 2024/1074. https://eprint.iacr.org/2024/1074
[39] Jens Müller, Fabian Ising, Vladislav Mladenov, Christian Mainka, Sebastian Schinzel, and Jörg Schwenk. 2019. Practical decryption exfiltration: Breaking pdf encryption. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 15–29.
[40] Jens Müller, Dominik Noss, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. 2021. Processing dangerous paths. In *Network and Distributed Systems Security Symposium*. NDSS.
[41] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based fault injection attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1466–1482.
[42] Assa Naveh and Eran Tromer. 2016. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 255–271.
[43] Wilson Nguyen, Dan Boneh, and Srinath Setty. 2023. Revisiting the Nova Proof System on a Cycle of Curves. Cryptology ePrint Archive, Paper 2023/969. https://eprint.iacr.org/2023/969 https://eprint.iacr.org/2023/969.
[44] Wilson Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. 2024. Mangrove: A Scalable Framework for Folding-based SNARKs. *Cryptology ePrint Archive* (2024).
[45] Amna Qureshi, David Megías, and Minoru Kuribayashi. 2021. Detecting deepfake videos using digital watermarking. In *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, 1786–1793.
[46] Simon Rohlmann, Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. 2021. Breaking the specification: Pdf certification. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1485–1501.
[47] Mike Schroepfer. 2019. Creating a data set and a challenge for deepfakes. *Facebook artificial intelligence* 5 (2019). https://ai.meta.com/blog/deepfake-detection-challenge/
[48] Srinath Setty. 2019. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2019/550. https://eprint.iacr.org/2019/550 https://eprint.iacr.org/2019/550.
[49] Dan Boneh Trisha Datta. 2022. Using ZK Proofs to Fight Disinformation. https://medium.com/@boneh/using-zk-proofs-to-fight-disinformation-17e7d57fe52f
[50] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lippi, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. 2020. LVI: Hijacking transient execution through microarchitectural load value injection. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 54–72.
[51] Stephan Van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Daniel Genkin, Andrew Miller, Eyal Ronen, Yuval Yarom, and Christina Garman. 2024. Sok: Sgx. fail: How stuff gets exposed. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4143–4162.
[52] Junke Wang, Zuxuan Wu, Wenhao Ouyang, Xintong Han, Jingjing Chen, Yu-Gang Jiang, and Ser-Nam Li. 2022. M2tr: Multi-modal multi-scale transformers for deepfake detection. In *Proceedings of the 2022 international conference on multimedia retrieval*. 615–623.

[53] Tianyi Wang and Kam Pui Chow. 2023. Noise based deepfake detection via multi-head relative-interaction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 14548–14556.

A Contrast Adjustment Algorithm

contrast adjustment algorithm as follows:

Algorithm 5: Contrast (Step i)

Public Input : $h_\alpha^{i-1}, h_\beta^{i-1}, cf$
Private Input : α_i, β_i
Public Output: $h_\alpha^i, h_\beta^i, cf$

/* repeat for each color (R, G, B) */
foreach $c \in [R, G, B]$ **do**
  for $j : 0 \rightarrow len(\alpha_i)$ **do**
    $val \leftarrow (\alpha_{i,j}^c - 128) * cf + 12800$
    $val \leftarrow \mathbf{cap}_0^{25500}(val)$
    assert $100 > |val - \beta_{i,j} \times 100|$
$h_\alpha^i \leftarrow H(h_\alpha^{i-1} | H_\sigma(\alpha_i))$
$h_\beta^i \leftarrow H(h_\beta^{i-1} | H_\sigma(\beta_i))$

B Brightness Adjustment Algorithm

Algorithm 6: Brightness (Step i)

Public Input : $h_\alpha^{i-1}, h_\beta^{i-1}, bf$
Private Input : α_i, β_i
Public Output: $h_\alpha^i, h_\beta^i, bf$

/* repeat for each color (r, g, b) */
foreach $c \in [r, g, b]$ **do**
  for $j : 0 \rightarrow len(\alpha_i)$ **do**
    $val \leftarrow \alpha_{i,j}^c * bf$
    $val \leftarrow \mathbf{cap}_0^{25500}(val)$
    assert $100 > |val - \beta_{i,j}^c \times 100|$
$h_\alpha^i \leftarrow H(h_\alpha^{i-1} | H_\sigma(\alpha_i))$
$h_\beta^i \leftarrow H(h_\beta^{i-1} | H_\sigma(\beta_i))$

C Cap Functionality Implementation in Circom

Listing 1 provides circuit-level code for implementing cap in Circom language.

```
template Cap(n) {

    // n must be eual to ceil(log(max_limit))
    signal input max_limit
    signal input calced_value;
    signal output final_value;

    component lt[4];
    component selector;
    component gt_selector;

    // find sign of calced_value
    lt[0] = LessEqThan(n);
    lt[1] = LessEqThan(n);
    lt[0].in[1] <== 0 - calced_value;
    lt[0].in[0] <==  calced_value;
    lt[1].in[0] <== max_limit;
    lt[1].in[1] <==  calced_value;

    gt_selector = Mux1();
    gt_selector.c[1] <== max_limit;
    gt_selector.c[0] <== calced_value;
    gt_selector.s <== lt[1].out;

    selector = Mux1();
    selector.c[0] <== gt_selector.out;
    selector.c[1] <== 0;
    selector.s <== lt[0].out;

    final_value <== selector.out;
}
```

Listing 1: Cap functionality implementation in Circom.

D Implementation of IF-Else Statement in R1CS

Algorithm 7 provides the logic to realize `If-Else` statement using `multiplexer` and `comparison` gates in R1CS setting. Listing 2 implements this algorithm in Circom.

Algorithm 7: IF-Else Statements in R1CS

$MUX.in[0] \leftarrow val\_1$ //if $i < y \ \vee \ i > y + h_{crop}$
$MUX.in[1] \leftarrow val\_2$ //if $y \leq i < y + h_{crop}$
$gte \leftarrow Circuit\ (GreaterThanEqual)$
$gte.in[0] \leftarrow i_{row}$
$gte.in[1] \leftarrow y$
$lt \leftarrow Circuit\ (LessThan)$
$lt.in[0] \leftarrow i_{row}$
$lt.in[1] \leftarrow y + height_{crop}$
$MUX.s \leftarrow gte.out \times lt.out$
$next\_crop\_hash \leftarrow MUX.out$

```
    // if the row is within the cropped area
    component selector = Mux1();
    selector.c[0] <== prev_crop_hash;
    selector.c[1] <== trans_hasher.hash;

    component gte = GreaterEqThan(12);
    gte.in[0] <== row_index;
    gte.in[1] <== crop_start_y;

    component lt = LessThan(12);
    lt.in[0] <== row_index;
    lt.in[1] <== crop_start_y + heightCrop;

    selector.s <== gte.out * lt.out;
    next_crop_hash = selector.out;
```

Listing 2: If-Else statement implementation in Circom.

E Decompression Circuit

```
template Decompressor(){
  signal input in;
  signal output out[10][3];
  component toBits = Num2Bits(240);
  component toNum[10][3];
  toBits.in <== in;
  for (var i=0; i<10; i++) {
    for (var j=0; j<3; j++) {
      toNum[i][j] = Bits2Num(8);
      toNum[i][j].in[0]<== toBits.out[i*24+j*8];
      ...
      toNum[i][j].in[7]<== toBits.out[i*24+j*8+7];
      out[i][j] <== toNum[i][j].out;
    }
  }
}
```

Listing 3: Decompressor circuit in circom.

F C2PA-Compatible Marketplace

Assuming the original image remains untampered and is signed with a trusted and authorized key linked to a real or organizational entity[9], we propose an approach that eliminates the need for pre-registration of the original image before publishing the edited version. To achieve this, we force any editor to prove their knowledge of specific transformations performed on an authentic original source, resulting in the final refined image. More precisely, the prover is required to submit a claim following the format outlined in Definition 4.1.

Figure 10 provides an overview of the proposed protocol. There are two potential scenarios for submitting a new edited version of an original source:

(1) **Sender has direct rights to the original image:** In this case the smart contract verifies the proof and finalizes the transaction accordingly.
(2) **Sender lacks rights to the original image:** The sender must additionally provide a signed declaration from the entity holding the rights to the original image to confirm ownership transfer.

In the proposed model, each original content can have only one owner, implying that all edited versions of an original image belong to a single owner. In our ownership model, all edited versions of an image reference the original image, and the original image points to its owner. This structure ensures that the cost of ownership transfer remains independent of the number of edited versions, as only one storage field in the contract needs updating[10].

Another consideration in the protocol design is that the percentage of photos taken by artists that ultimately get published is typically low. Therefore, registering every unedited image to the blockchain before editing and finalizing is not a scalable solution. To address this, we propose a method allowing honest users to register commitments to an original image simultaneously with submitting the finalized edited version.

[9]The trusted entity could be of any type, such an authentic tamper-proof camera, a generative AI model, or a DAO of digital artists.
[10]We acknowledge that there are various approaches to address ownership of an original source and its refined versions, depending on the application and target scenario. However, in this paper, our focus is on ensuring the accountability of media in compliance with C2PA.

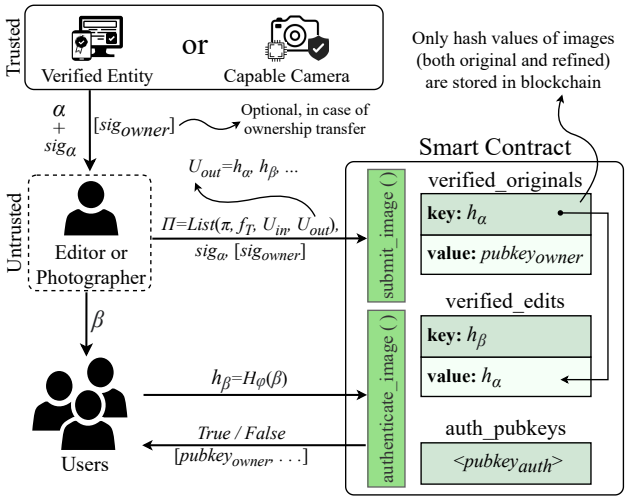

Figure 10: Trustless and C2PA-compatible marketplace.

G Artifact Appendix

Making sure that images shared online are authentic and trustworthy is a big challenge. But let's be real: most images need some tweaking before they go public. Zero-knowledge proofs (ZKPs) can help by verifying edited images without needing to reveal the original. The problem? ZKPs are often costly, especially when it comes to prover complexity and proof size. That's where VIMz comes in. VIMz is a framework designed to prove the authenticity of high-resolution images efficiently using folding-based zkSNARKs (powered by the Nova proving system) [32, 33, 43]. With VIMz, we can verify that both the original and edited images are legit, along with the correctness of the transformations, all without revealing any intermediate versions—only the final image is exposed. Plus, VIMz keeps the identities of the original creator and subsequent editors private while proving the final image's authenticity, making it ideal for privacy-preserving, trustless marketplaces compatible with C2PA standards. It's efficient enough to handle 8K (6632×4976) images on a mid-range laptop with minimal memory and proof size, offering fast verification and parallel processing capabilities. We formally prove the security of VIMz on Section 4 of paper.

Our tests show that VIMz is fast and efficient on both the prover and verifier sides. For example, you can prove transformations on 8K (33MP) images using just a mid-range laptop, hitting a peak memory usage of 10 GB. Verification takes less than 1 second, and proof sizes come in at under 11 KB no matter the resolution. Plus, the low complexity of VIMz means you can prove multiple transformations in parallel, boosting performance by up to 3.5× on the same machine.

VIMz is fully open-source and available on a public GitHub repository[11]. Within this repository, you will find all the code necessary for implementing zero-knowledge circuits in the Circom language, which can be used with Nova. The repository is organized into four directories:

[11]https://github.com/zero-savvy/vimz

- **circuits**: Contains the underlying ZK circuits of VIMz in circom language.
- **contracts**: Contains high-level Solidity smart contracts (see Appendix F) that provide the infrastructure for a C2PA-compatible marketplace on EVM-based blockchains.
- **nova**: Contains the main cargo-based package for building and installing VIMz using nova protocol.
- **py_modules**: Houses the Python interface (GUI) of VIMz, facilitating image editing and preparation of input files for the VIMz prover.
- **samples**: Holds images in standard resolutions (e.g., SD, HD, 4K) along with pre-built JSON files of supported edits to be fed into the VIMz prover.

To further assist developers, we have provided scripts for building Circom circuits and running VIMz in both single-threaded and multi-threaded modes to benchmark its performance on any commodity hardware with minimal effort.

Further Developments: [**Note**: This process requires knowledge of ZKP and familiarity with the Circom language and Nova proving system.] If someone wishes to customize the protocol, following changes must be made in the respected directories:

(1) **py_modules**: update image_formatter.py.
(2) **circuits**: Add the new .circom circuit w.r.t. to necessary properties of the Nova-Scotia [11].
(3) **nova/src**: updating the main.rs file accordingly.

G.1 Description & Requirements

G.1.1 How to access. VIMz is publicly accessible in open-source format via Github: https://github.com/zero-savvy/vimz and Zenodo DOI: https://zenodo.org/doi/10.5281/zenodo.12516127.

G.1.2 Hardware dependencies. None.

G.1.3 Software Dependencies. All experiments in our research are reproducible using commonly available commodity hardware running Linux operating systems. To simplify the benchmarking process, we have included sample input JSON files for the VIMz prover in the samples/JSON directory. Furthermore, we have provided several scripts to streamline the installation, building and benchmarking process.

G.1.4 Benchmarks. To streamline the benchmarking process, we have included scripts in the repository that automate the execution of individual or multiple (parallel) instances of VIMz. These scripts utilize the sample JSON files available in the repository for testing purposes.

G.2 Artifact Installation & Configuration

VIMz relies on several libraries and packages for proper execution, including rust, NodeJS, and Python3. Below, we outline general commands to install the main dependencies required by VIMz.

- For Installing **Node JS**:

```
curl -o- https://raw.githubuserconten
t.com/nvm-sh/nvm/v0.39.3/install.sh |
bash
```

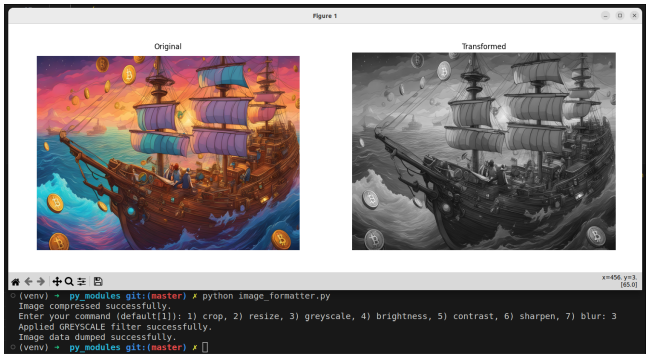

Figure 11: Python GUI of VIMz.

```
source ~/.bashrc
```

```
nvm install v16.20.0
```

- For Installing **rust**:

```
curl --proto '=https' --tlsv1.2 -sSf
https://sh.rustup.rs | sh -s --
--default-toolchain none -y
```

- Additional build-essential libraries and packages:

```
sudo apt install gcc
```

```
sudo apt install build-essential
nlohmann-json3-dev libgmp3-dev nasm
```

- For installing **circom**:

```
git clone https://github.com/iden3/
circom.git
```

```
cd circom
```

```
cargo build --release
```

```
cargo install --path circom
```

- For installing **snarkjs**:

```
npm install -g snarkjs
```

Once you have installed these dependencies, you can proceed with setting up and running VIMz. To obtain the latest version of VIMz, clone its GitHub repository using the following command:

```
git clone https://github.com/zero-savvy/vimz
```

Head to the nova directory:

```
cd vimz/nova
```

build and install vimz using cargo:

```
cargo build
cargo install --path .
```

verify installation of vimz:

```
vimz --help
```

G.3 Experiment Workflow

To streamline the evaluation process, we have provided pre-generated sample input JSON files for VIMz prover along with automated scripts to execute them. Our performance evaluations of VIMz prover, as presented in Table 4, Table 5, and Table 6 in the paper, can be reproduced with minimal effort using the provided scripts and samples. In general, the `vimz` command requires the following inputs:

```
vimz --function <FUNCTION>
--resolution <RESOLUTION> --input <FILE>
--circuit <R1CS FILE> --output <FILE>
--witnessgenerator <BINARY/WASM FILE>
```

You can access more detailed information about the required inputs by running the following command:

```
vimz --help
```

G.4 Major Claims

In Section 6, we claim certain proving times of VIMz on different platforms, including a commodity hardware (DELL Latitude laptop). These claims can be easily verified and reproduced using the provided sample files and scripts.

G.5 Evaluation

G.5.1 Experiment (E1). [Proofs of HD resolution] [2 human-minutes + 10 compute-minutes]

[How to] Using the samples provided in the `samples/JSON/HD/` directory and the provided `benchmark.sh` script in the main directory.

[Preparation] Follow the steps below:

(1) go to the circuits directory:

```
cd vimz/circuits
```

(2) build ZK circuits using the provided script in this directory:

```
./build_circuits.sh
```

[Execution] Go to the main directory of vimz repo and run any number of transformations as you prefer using the provided script:

```
./benchmark.sh [list-of-transformations]
```

- *Example 1*: benchmarking a single transformation:

```
./benchmark.sh contrast
     or
./benchmark.sh blur
     or
./benchmark.sh grayscale
```

- *Example 2*: benchmarking parallel execution of multiple transformations:

```
./benchmark.sh contrast blur
     or
./benchmark.sh resize blur shapness
```

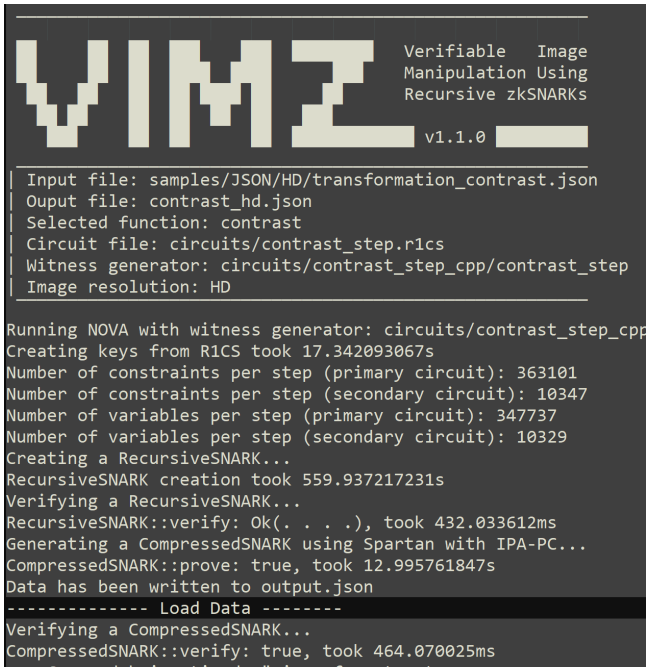

Figure 12: Example standard output generated by VIMz prover.

NOTE: Since the proof generation process can be time consuming, it is recommended to initially benchmark with only one transformation at a time (replicating the results presented in Table 4). Once these results are verified, you can proceed to run multiple transformations in parallel to replicate the results shown in Table 5.

[Results] The script generates a file (or multiple files, one per given transformation) with a `.output` suffix in the same directory. These files contain the standard output of running the `vimz` command directly, as shown in Figure 12. The output includes various performance metrics. The total proof generation time can be calculated as the sum of two numbers: `RecursiveSNARK creation` and `CompressedSNARK::prove:` from the output.

G.6 Customization

For running the **`python`**-based GUI and applying different transformations other than the ones given in `samples` directory, following steps must be taken:

```
cd vimz/py_modules/
virtualenv venv; source venv/bin/activate
pip install -r requirements.txt
python python_formatter.py
```

we recommend the following steps to redesign or add a new transformation to the VIMz process:

(1) **`py_modules`**: Edit or add the preferred transformation to the Python file `image_formatter.py`. This file contains useful utility functions, such as `compress()`, which handle the creation of a pre-processed suitable JSON file input for the VIMz prover.

(2) **`circuits`**: Define the circuit that verifies and calculates the hash of both the original and transformed images in Circom. Ensure that the circuit follows necessary properties of the Nova-Scotia framework [11].
(3) **`nova/src`**: Define the new method in the `main.rs` Rust file to ensure proper execution by VIMz. This phase is responsible for executing steps and proving them recursively using witness generators from Circom inside the Nova protocol.