

VIMz: Verifiable Image Manipulation using Folding-based zkSNARKs

Stefan Dziembowski^{1,2}, Shahriar Ebrahimi¹, and Parisa Hassanizadeh^{1,3}

{*firstname.surname*} @ideas-ncbr.org¹ @crypto.edu.pl²

¹IDEAS NCBR, Poland

²University of Warsaw, Poland

³Polish Academy of Science, Poland

Abstract—With the rise of generative AI technology, the media’s credibility as a source of truth has been significantly compromised. This highlights the need to verify the authenticity of media and its originality. Ensuring the integrity of media during capture using the device itself presents a straightforward solution to this challenge. However, raw captured media often require certain refinements or redactions before publication. Zero-knowledge proofs (ZKP) offer a solution by allowing attestation of the correctness of specific transformations applied to an authorized image. While shown to be feasible, previous approaches faced challenges in practice due to their high prover complexity.

In this paper, we aim to develop a practical framework for efficiently proving the authenticity of HD and 4K images on commodity hardware. Our goal is to minimize prover complexity by utilizing the folding-based zkSNARKs technique, resulting in VIMz, the first practical verifiable image manipulation system of this kind. VIMz leverages Nova’s folding scheme to achieve low complexity recursive zkSNARK proofs of authentic image manipulation. Our implementation results demonstrate a substantial reduction in prover complexity—up to a 3× speedup in time and a 96× reduction in memory (from 309 GB in [Kang et al., arXiv 2022] to only 3.2 GB). Moreover, the low memory consumption allows VIMz to prove the correctness of multiple chained transformations simultaneously, further increasing the performance (up to 3.5×). Additionally, we propose a trustless smart contract system that autonomously verifies the proofs of media authenticity, achieving trustless copyright and ownership management, aligning with the standards of the Coalition for Content Provenance and Authenticity (C2PA). Such a system serves as a foundational infrastructure for constructing trustless media marketplaces with diverse applications.

I. INTRODUCTION

“Imagine this for a second: One man, with total control of billions of people’s stolen data, all their secrets, their lives, their futures.” This statement originates from a DeepFake video featuring Mark Zuckerberg, the CEO of Facebook, in 2019 [1]. Fast forward to 2023, where the recipient of the Sony World Photography Award declined the honor, unveiling that the winning image was generated by AI, a fact unbeknownst to the judges [2]. In the era of widespread AI applications, the investigation of media origin and authenticity has become increasingly important. Recent advancements in generative models facilitates development of realistic face-swapped videos and images using consumer-level GPUs and software libraries, such as DeepFaceLab [3].

Furthermore, popular text-to-image models such as Midjourney [4], DALL•E [5], and Stability AI [6] are accessible through relatively affordable monthly subscriptions. The accessibility of tools that can convincingly produce fake videos and images underscores the escalating potential for spreading misinformation and manipulating public opinion. Therefore, it is necessary to critically assess the sources of information and possess the ability to distinguish between original and synthetically produced or altered media.

To this end, leading corporations are actively engaged in research on detecting deepfakes. Noteworthy among these efforts are three benchmark datasets: DFD by Google [7], DFDC by Facebook [8], and Celeb-DF [9], all released in 2019. As reported in [10], the AUC scores achieved by deepfake detectors on these datasets were up to 0.86, 0.76, and 0.66, respectively. These results emphasize the ongoing challenges in the field, indicating that there is considerable work to be done for these models to reach a level of maturity.

As AI researchers focus on training and refining models to create tools for discerning between authentic and synthetic media, alternative approaches are being explored by researchers in other areas. One 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 [11]. A notable initiative of this approach is the Coalition for Content Provenance and Authenticity (C2PA) project that establishes technical standards for verifying the origin and history (provenance) of media content [11].

This process, however, requires a substantial level of trust in the authorities and certified validators within the protocol. Furthermore, users are restricted to specific software and applications, which must also be deemed trustworthy. Indeed, this trust model, particularly in software and third parties, has been shown to introduce direct and serious attack vectors when relying on software for secure digital signatures [12], [13], [14], [15], [16], [17].

A more ideal and secure approach would involve the ability to publicly 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 to generate a succinct proof for the edited image that is publicly verifiable by everyone [18], [19], [20]. SNARKs, or Succinct Non-Interactive Arguments of Knowledge, are

cryptographic tools that offer concise proofs of knowledge, verifying the correctness of a statement. Unlike the approach of C2PA, the SNARKs proving systems do not require any kind of trust during proof generation and, therefore, cannot be threatened by previous attacks. Consequently, anyone can publicly verify whether the refined image aligns with the provided proof or not.

However, a drawback of this approach is the computational complexity of the prover, particularly with the increasing resolutions of media to FHD and 4K. Consequently, the proof generation process becomes resource-intensive, especially in terms of RAM usage. For example, in related work [20], complete¹ proof generation (proof of correct transformation along with proof of integrity) for one convolution-based transformation on an HD-resolution image demands up to 309 GB of RAM and over 21 minutes on an AWS server with 64 vCPU cores. This limits the applicability and renders this approach nearly impractical for images with higher resolutions, such as FHD or 4K.

To tackle this challenge, we propose decomposing each transformation into smaller functions that collectively produce the transformed image. These smaller functions are then implemented within a folding-based zkSNARKs framework, wherein the system verifies the proof from the previous step before generating a new cumulative proof. Folding-based zkSNARKs have demonstrated greater efficiency compared to traditional ones by enforcing uniform circuit requirements across all recursive steps in the proof [21], [22], [23], [24]. Notably, Nova proving system [21] offers a robust open-source implementation, leading to a highly efficient prover [25].

Therefore, our approach involves breaking down the entire transformation process of an authentic image into recursive steps within the Nova protocol to leverage its low-complexity prover. However, the requirement that all steps must adhere to the same circuit presents a non-trivial task. Nonetheless, the outcome is VIMz² (Verifiable Image Manipulation using folding-based zkSNARKs), which, to the best of our knowledge, is the first system of its kind to leverage the higher efficiency of folding-based zkSNARKs. Implementation of VIMz is entirely open-source³ and comprises various programming stacks, including Python, Circom, Rust, and Solidity.

VIMz outperforms the state-of-the-art primarily due to its significantly enhanced memory efficiency. Specifically, when proving transformations in HD resolution and calculating the complete hash of both the original and the resulting transformed image, VIMz utilizes a peak memory of only 3.2 GB of RAM. This represents over 96× improvement in memory efficiency compared to [20]. Moreover, VIMz achieves up to 3× speedup in proving time compared to [20], depending on the type of transformation applied.

This notable improvement positions VIMz as the first proving platform for secure image transformations that can be executed on commodity hardware. The increased memory

efficiency also enables the simultaneous execution of multiple VIMz instances, even on standard hardware, resulting in significantly improved performance compared to the state-of-the-art. Moreover, while proving transformations on higher image resolutions, such as FHD and 4K, was impractical in previous work, VIMz can provide such proofs on a mid-range laptop with only 16 GB of RAM.

Thanks to the low computational demands of VIMz on the prover side, its applications can be extended to a broader range of scenarios. In light of this, we have designed a decentralized and autonomous protocol to trustlessly manage copyright and ownership of authenticated media content. This protocol can be effectively realized as an infrastructure for constructing a trustless C2PA-compatible media market.

The key contributions of this paper can be summarized as follows:

- 1) **VIMz:** We introduce VIMz, the first practical proving system for authentic refinements on verified images using folding-based zkSNARKs. VIMz is open-source and offers:
 - **Optimized proofs of integrity:** We introduce and implement various optimizations, including lossless compression of pixel data over the Pallas/Vesta field, resulting in a nearly 10× reduction in circuit complexity.
 - **Memory efficiency:** VIMz has a peak memory usage of just 3.2 GB when proving transformations in HD resolution, a significant improvement of over 96× compared to the state-of-the-art.
 - **Low complexity:** VIMz introduces up to a 3× speedup in proving time compared to related work.
 - **Practicality:** Capable of running on commodity hardware and proving transformations on 4K images even on a mid-range laptop.
 - **Parallel Execution:** The low memory requirements of VIMz allow running multiple instances in parallel, resulting in up to 3.5× additional speedup compared to a single execution.
- 2) **Trustless, C2PA-compatible marketplace:** We propose a decentralized and autonomous smart contract system based on the proofs generated by VIMz as an infrastructure to trustlessly manage copyright and ownership of authentic images.
- 3) **Security Analysis:** We formally analyze the security of proofs generated by VIMz with respect to the probabilistic polynomial time (PPT) adversary model. Additionally, we extend the analysis to encompass the proposed trustless C2PA-compatible marketplace.

The remainder of the paper is structured as follows. Section II presents a review of related work, highlighting the gaps between ideal solutions and the current state-of-the-art. Section III offers the necessary background to comprehend the paper along with our trust and adversary model. Section IV details design principals behind VIMz and analyzes the soundness of the generated proofs. Section V outlines optimizations applied to VIMz at both the circuit and protocol levels. Section VI presents experimental analysis of VIMz during the proof generation for authentic transformations in both HD and 4K resolutions, and compares its performance

¹In 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.

²Pronunciation: // wimzi/, like the word *whimsy*!

³Github link: <https://github.com/zero-savvy/vimz>

TABLE I. COMPARISON WITH PREVIOUS WORK

	Foundation	Trust Assumption	Proofs of Integrity [♠]	Prover Setting		Public Verifier	Confidential Original Source	Maximum Resolution	Implementation [‡]	
				Untrusted	Complexity					
Trusted Editor [11], [26]	Digit. Signature	Origin [♠] & Editor/TEE	✗	✗	Very Low	✗ [⊙]	✓ [⊙]	unlimited	-	
Deepfake Detection	ML/AI	AI Model	N/A	✓	N/A	✗	✓	unlimited	-	
Verifiable Computation	[18]	PCD [27]	Origin [♠] & TEE [□]	✗ [▼]	✓ [⊙]	Very High	✓	✗	128 × 128	N/A
	[19]	Groth16 [28]		✗ [▼]	✓	High	✓	✗	4K & higher	Github
	[29]	GKR[30]		✗ [▼]	✓	Low [*]	✓	✗	HD	Github
	[31]	SNARKs		✓	✓	Low [*]	✓	✓	FHD [▼]	Github
	[20]	Halo2 [32]		✓	✓	High	✓	✓	HD	N/A
	This Work	Nova [21], [25]		✓	✓	Low	✓	✓	4K & Higher	Github

[♠] Requires full realization of hash calculations for both the original and refined images within the VC circuit. While this incurs significant complexity, it is the foundation for preserving the confidentiality of the original image. [▼] Although authors discuss the idea of calculating the hash of input and output images, it is not implemented.

^{*} Status/platform of the implementation and its availability. ^{*} Limited supported transformations: [29] only supports convolution-based, while [31] only focuses on redaction.

[♠] The original image is untampered and its signature is trustworthy. [□] Optional. [⊙] Conditional. [▼] 16 × 16 block size.

to the state-of-the-art. In Section VII, we propose a trustless and autonomous smart contract system as the foundation of an authentic media marketplace, utilizing proofs of valid image transformations and analyzing its security against the adversary model. Finally, in Section VIII, we conclude the paper by summarizing our contributions, discussing the implications of VIMz, and outlining potential future research directions.

II. RELATED WORK

In this section, we provide a thorough examination of three primary strategies to combat disinformation in media. While these approaches may not always be directly comparable, we point out their individual strengths and weaknesses. Table I offers a detailed comparison, outlining the distinctive features and advantages of this work compared to previous work.

a) Trusted Editor: From a cryptographic standpoint, approaches like the C2PA solution rely on digital signatures. In such methods, when a user captures a photo, the camera or a specific software application (e.g., Truepic on mobile phones) adds metadata to the photo and signs it on behalf of the user. The photo is then refined by some trusted editing software tools and applications (e.g., Adobe Photoshop). As edits are applied to previous versions of the image, new metadata is appended to the existing records, enabling the entire edit history to be verified by trusted company applications. In an ideal scenario, 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 can lead to complete disasters on both the prover and verifier sides [13], [12], [14]. To this end, as suggested in the C2PA trust model [11], the software should be run on trusted execution environments (TEE), which, in turn, introduces an additional layer of trust assumption.

b) Deepfake detection models: Several studies aim to develop AI models capable of effectively distinguishing between real and artificial images. Some of this work concentrates on directly detecting manipulation artifacts or forensic noises in AI-generated media [33], [34], [35]. Meanwhile, other approaches explore alternative methods, such as the utilization of advanced watermarks [36]. Ideally, a perfect model would distinguish between authentic and artificially generated images without relying on trust assumptions for

the camera or metadata signatures. Nevertheless, recent reviews [10] emphasize a significant gap in accuracy and realism between generative AI models and deepfake detection models. This suggests that even on limited datasets, such techniques fall short of being a reliable option for real-world applications. Additionally, distinguishing between authentic refinements and potential misinformation transformations poses a challenge during the training process of such models.

c) Verifiable Computation (VC): The rationale for employing VC in this context is to develop a protocol that enables the proof 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 commitment method that binds the transformed image to the original source. It is essential that the prover can demonstrate the correctness of all calculations within the commitment scheme inside the VC environment. This ensures that the authenticity of the refined image can be proven without ever revealing the original image to any party.

Theoretically, this approach holds 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, limiting the practical adoption of this method. Photoproof [18] 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. A subsequent study in 2022 [19] further enhanced efficiency and demonstrated the feasibility of applying this approach to higher image resolutions. The latest advancement in this field is ZK-IMG [20], which utilizes the Halo2 proving system [32] to demonstrate transformations in HD resolution. Another study [29] employed a similar approach to prove the correctness of convolution function evaluations. However, their focus was limited to convolution functions and did not support other types of image manipulation. Moreover, [29] does not provide proofs of integrity and, therefore, does not fully implement the complete proof system required to bind the proofs to the original source, falling short of achieving zero-knowledge regarding the confidentiality of the original source. Another related work is [31], which provides verifiable image redaction. The main idea in [31] is to group pixels into larger blocks, such as 16x16, to improve overall performance.

Until now, ZK-IMG [20] and [31] were the only works capable of including the hash of both the original and transformed images within the proof, thereby achieving trustless integrity assurance. However, [31] only implements image redactions. On the other hand, performance evaluations in [20] reveal significant resource requirements, with peak memory usage exceeding 309 GB of RAM and proof generation times surpassing 1300 seconds for authentic refinements on an HD image.

Selection Rationale for Proof System: Since the introduction of practical zkSNARKs like Groth16 [28] and Plonk [37], considerable efforts have been made to enhance zk-SNARK efficiency. Some approaches distribute computations across multiple provers [38], [39], but these are not suitable for our use case, i.e. media authenticity, where a single prover with access to confidential image data needs to demonstrate specific refinements w.r.t. the original source. Alternatively, Elastic SNARKs, like Gemini [40], offer prover configurability, yet recent studies [22] suggest that folding-based schemes, such as Nova, generally tend to have lower prover complexity. Hence, we opted for a folding-based structure in designing VIMz, specifically leveraging the Nova proving system due to its available and extensive implementation [25]. We emphasize that due to our circuits alignment with the fundamental architecture of folding schemes, VIMz remains adaptable to future advancements in this domain like [22].

III. BACKGROUND

Table II presents the terminology used in the paper. We generally borrow the mathematical representations from [21]. Certain symbols and notations will be elaborated upon in their respective sections. Below, we provide an overview of the main concepts covered, including zero-knowledge proof (ZKP) systems in general, followed by the the folding-based zkSNARKs of Nova.

A. Succinct Non-Interactive Arguments of Knowledge

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 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.
- $\pi \leftarrow \mathcal{P}(pk, u, w)$: The proving algorithm stating that $(pp, s, u, w) \in \mathcal{R}$.
- $b \leftarrow \mathcal{V}(vk, u, \pi)$: The verification algorithm, where $b \in \{0, 1\}$.
- $\pi \leftarrow \mathcal{S}(pp, u, \tau)$: The simulator that outputs π given the trapdoor τ .

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

Definition III.1. A non-interactive argument for \mathcal{R} $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{S})$ is a SNARK if it satisfies:

- **Completeness:** An honest prover with valid witness should convince any verifier. Formally, for any PPT

TABLE II. TERMINOLOGY OF THE PAPER

Notation	Description
\mathbb{Z}_p	$\mathbb{Z}_p : \bigcup i, 0 \leq i < p, i \in \mathbb{N} = \{0, 1, \dots, p\}$
\mathbb{B}	Denotes one bit: $\mathbb{B} \in \mathbb{Z}_2$
α, β	We refer to the pixel matrices of the <i>original</i> and the <i>transformed</i> image as α and β , respectively.
$\alpha^R \alpha^G \alpha^B$	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 the image α .
H	Poseidon [41] hash function with 2 inputs and one value output. $H : \mathbb{Z}_p^2 \rightarrow \mathbb{Z}_p$
H_σ	Poseidon [41] hash value of an entire row with n pixels (recursively): $H_\sigma : \mathbb{Z}_p^n \rightarrow \mathbb{Z}_p = H(\alpha_{i,n-1} H(\alpha_{i,n-2} H(\alpha_{i,n-3} \dots H(\alpha_{i,2} H(\alpha_{i,1} H(\alpha_{i,0} 0)))) \dots)$.
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	Poseidon [41] hash value of an entire image with $n \times m$ pixels (recursively): $H_\phi : \mathbb{Z}_p^{n \times m} \rightarrow \mathbb{Z}_p = h^n$.
f_T	Transformation function: $\beta \leftarrow f_T(\alpha, U_{in})$
U_{in}^i, U_{out}^i	Public input and output of each step in recursive SNARK.
\mathcal{K}	Kernel matrix for convolution-based transformations.
tx_{msg}	Equivalent to the <code>msg</code> global variables in Solidity, such as <code>msg.sender</code> to access the public address of sender.

adversary \mathcal{A} :

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

- **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:

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

- **Zero-knowledge:** If the argument does 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:

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

$$D_2 = \left\{ \begin{array}{l} (pp, \rho) \leftarrow \mathcal{S}(1^\lambda) \\ (s, (u, w)) \leftarrow \mathcal{A}(pp) \\ (pp, s, u, w) \in \mathcal{R} \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ \pi \leftarrow \mathcal{S}(pp, u, \rho) \end{array} \right\}$$

B. Incrementally Verifiable Computation (IVC)

IVC allow verification of computations done by repeated application of the same function. More precisely, for a given function f , with initial input z_0 , IVC allows for generating a proof Π_i for stating that $z_i = f^i(z_0)$, given a proof Π_{i-1} for stating $z_{i-1} = f^{i-1}(z_0)$. An interesting property IVC schemes in general 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 III.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} :

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

- **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:

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

C. Nova Proving System

Nova [21] 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 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*⁴ prover and a relatively minimal verifier circuit of about 10,000 multiplication gates [21]. Fig. 1 presents an overall overview of the folding-based IVC structure in Nova.

The *Nova-rust* library [25] fully implements the Nova proving system and verifies the compressed output of recursive SNARKs within the *Spartan* [42] to achieve compact SNARK proofs. Additionally, the implementation offers support for

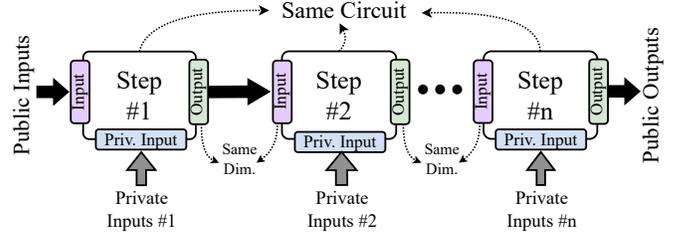


Fig. 1. The folding scheme structure of Nova [25].

multiple frontends to define recursive circuits in Nova. For example, the *Nova-Scotia* library enables the use of witness generator binaries of Circom [43] in Nova, acting as middleware between R1CS⁵ constraints in Circom and Nova prover [44]. This flexibility enhances the usability and integration capabilities of the Nova proving system in practice.

IV. 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 transformations and analyze its soundness. Lastly, we detail the design principles for folding-based image transformations within the IVC scheme.

A. Trust and Adversary Model

The trust model in our protocol relies on one foundational assumption, specifically that the original image captured by the camera 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 [45] or the Truepic Lens SDK for mobile devices [26]) or an authorized/trusted entity. This assumption aligns with prior work [20], [11] and forms a critical foundation for any protocol in this area. However, unlike the trust model of C2PA [11], we do not necessitate additional trust assumptions. Both the editor and the storage components in our protocol are considered untrusted. Notably, verification can be achieved without any 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:** Poseidon hash functions [41], which we specifically utilize, are proven to be secure and efficient within the SNARKs setting.
- **Digital signature schemes:** The security of digital signature schemes, such as ECDSA or EdDSA.
- **Nova and Spartan proving systems:** We assume that forging a false proof in such systems is not computationally possible by a PPT adversary.

⁴Transparent SNARKs do not require *trusted setup*.

⁵Rank-1 Constraint System

B. Overview of the Protocol

We introduce VIMz, a protocol that enables users to obtain verifiable transformations on their images. This protocol 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). The protocol 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 public 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 proof of the authentic transformation. The prover then sends the proofs along with the transformed image to the verifier.
- **Verification Phase:** The verifier checks the correctness of the original image's signature using verified public keys of the trusted entity and validates the proof of the transformed image.

For the rest of this section, we focus on details of the prover side and analyse the security of the proofs generated by the VIMz in prover side. In Section VII, we design a C2PA-compatible marketplace based on this protocol. Fig. 2 illustrates the overall architecture of the VIMz prover. To facilitate standard image transformations, we have designed a user-friendly 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 designed prover circuit based on Nova folding scheme. The prover subsequently outputs proofs w.r.t. the selected transformation. Specifically, the final π_{SNARK} is a valid proof of the following statement:

Definition IV.1. Let h_α and h_β be the hashes of images α and β , respectively. Let $U_{out} = \{h_\alpha, h_\beta\}$ and U_{in} 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:

$$S[f_T, U_{in}, U_{out}] = \left\{ \begin{array}{l} \text{I know } \alpha \text{ and } \beta, \\ \text{such that } \beta = f_T(\alpha, U_{in}), \\ \text{and } h_\alpha = H_\phi(\alpha) \text{ and } h_\beta = H_\phi(\beta) \end{array} \right\}$$

A notable feature of the statement in Definition IV.1 is the ability to 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.

As mentioned earlier, we use Nova for generating SNARKs proofs of media authenticity. Nova allows different front-ends to define ZK circuits. For this, we employ the Nova-Scotia variant [44], enabling circuit definition in the Circom [43] language. This choice allows us to leverage the expressive capabilities of Circom and its dependable and extensive libraries, such as circomlib [46]. For more details on the software and specific commands to execute the proof generation properly, refer to the Artifact Appendix.

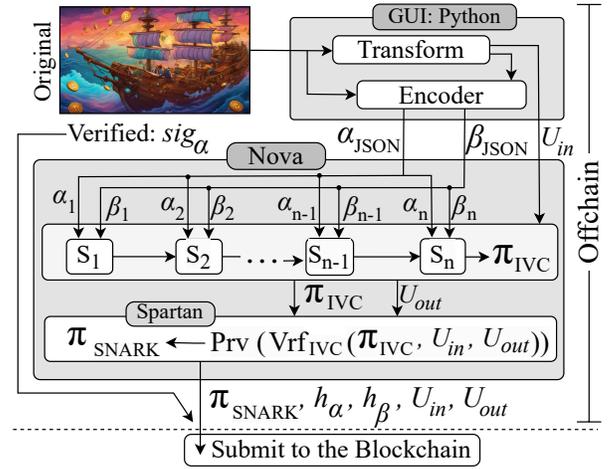


Fig. 2. Overview of the implemented architecture.

C. Folding-based Proofs of VIMz

In the proposed approach, while the original image remains confidential, each transformation mandates an associated zk-SNARKs proof that is publicly verifiable. This ensures that anyone can securely authenticate the edited image based on the soundness property of the accompanying proof w.r.t. the authenticated signature of the original image. Our goal is to provide zkSNARKs proofs ensuring both the accurate execution of a transformation and the computation of hashes for the original and transformed images to uphold integrity. However, as demonstrated by previous work [20], 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 efficient recursive proofs in Nova.

Fig. 3 illustrates the proposed method, employing 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, ensuring the integrity of input data. Consequently, the final step (step n) should produce the hash of both the original and transformed images, denoted as h_α^n and h_β^n respectively. Fig. 4 depicts the dataflow for constructing h_α^i in a row-by-row traversal. 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 Fig. 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 .

The prerequisites for each transformation may extend beyond the hash results of previous rows. Important information, 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 Fig. 3. Note that, due to the specific structure of the proof folding scheme, the data types and dimensions/sizes of public inputs in the circuit must align with the public outputs. The output of each intermediate step serves directly as the public input for the subsequent step.

To provide further clarification and adhere to the notation from [21], we redefine the following symbols: $\forall i \in \mathbb{N} : \omega_i =$

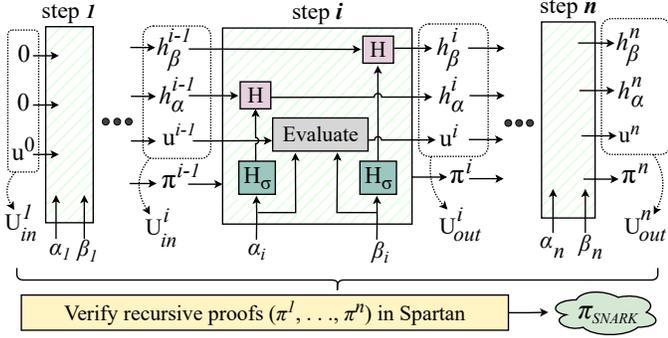


Fig. 3. Proving a transformation of an image in Nova, while evaluating the integrity of both original and transformed image with Poseidon hashing algorithm.

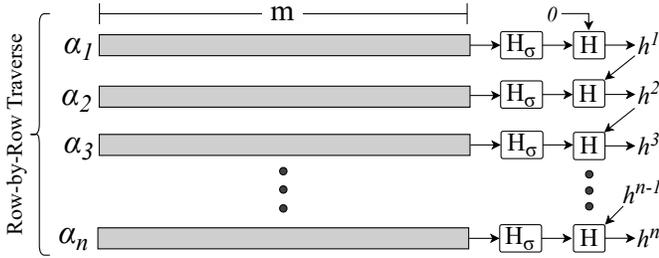


Fig. 4. Data flow of calculating the hash of an image in a row-by-row traverse setting.

(α_i, β_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 IV.2. 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:

$$\mathcal{F}_T : (\mathbb{Z}_{256}^{m \times 3}, \mathbb{Z}_p, \mathbb{Z}_p) \rightarrow (\mathbb{Z}_{256}^{m' \times k'}, \mathbb{Z}_p, \mathbb{Z}_p)$$

$$\mathcal{F}_T(\alpha_i, h_\alpha^{i-1}, h_\beta^{i-1}) = \begin{cases} \beta_i = \mathcal{F}_T(\alpha_i) \\ h_\alpha^i = H(h_\alpha^{i-1}, H_\sigma(\alpha_i)) \\ h_\beta^i = H(h_\beta^{i-1}, H_\sigma(\beta_i)) \end{cases}$$

For simplicity, we begin by formalizing a transformation that does not require any configurations, such as grayscale. This means we omit u^i values shown in Fig. 3, which are necessary for defining transformation-specific parameters, such as contrast factor or crop starting points. Thus, a valid VIMz proof's public input should only contain two zeros: $\{0, 0\}$, and the resulting public output should be $\{h_\alpha^n, h_\beta^n\}$. Now we can define the soundness of VIMz proofs as follows with respect to the soundness of general IVC defined in [21].

We reduce the knowledge soundness of VIMz proofs to two fundamental assumptions: 1) the soundness property of the IVC scheme, and 2) the collision resistance of the Poseidon hash function. 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:

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

Theorem 1. The probability of a PPT adversary breaking the soundness of VIMz proofs (the above-mentioned probability) 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 $H_\phi(\beta') = H_\phi(\beta) = h_\beta^n$ or $H_\phi(\alpha') = H_\phi(\alpha) = h_\alpha^n$. 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 $z = \{h_\alpha^n, h_\beta^n\}$ 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. ■

D. Details of Circuit Design

This section provides an in-depth detail for realizing each transformation within the folding scheme. Each circuit is designed and optimized based on principles of Circom language [43] and compiled to Nova-compatible RICS through Nova-Scotia [44].

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:

$$gray = (\alpha^R * 0.299) + (\alpha^G * 0.587) + (\alpha^B * 0.114) \quad (1)$$

Applying the *grayscale* effect to the image is straightforward. As illustrated in Fig. 3, the transformed image is evaluated against the original row-by-row. 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.

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 1. Notably, in line 3 of the algorithm, the standard constants in equation 1 are multiplied by 1000, resulting in `val` being $1000 \times$ larger than the actual grayscale value. 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

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  $j : 0 \rightarrow \text{len}(\alpha_i)$  do
2   /* Multiplied by 1000 to handle
   decimal values in  $\mathbb{F}_q$ , e.g., 0.299
    $\rightarrow 299. */$ 
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  $h_\beta^i \leftarrow H(h_\beta^{i-1} | H_\sigma(\beta_i))$ 
```

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).

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 2 below.

$$\text{cap}_0^k(x) = \text{Min}(\text{Max}(0, x), k) \quad (2)$$

Note on cap function: Capping values to integers within the range (0, 255) requires an approach that considers the peculiarities of working with \mathbb{F}_q . Determining the sign of the resulting value is the initial step, and if it is 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 Circom language [43].

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' transformed 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 image⁶. Fig. 5 represents resizing an HD to SD resolution in this setting. Given that the original and destination sizes are fixed during the benchmarks, constant

⁶Note that unlike rest of the transformations, proving *resize* functionality requires less steps. For instance, for downsizing from HD to SD resolution, we only require $240 = \frac{720}{3} = \frac{480}{2}$ steps.

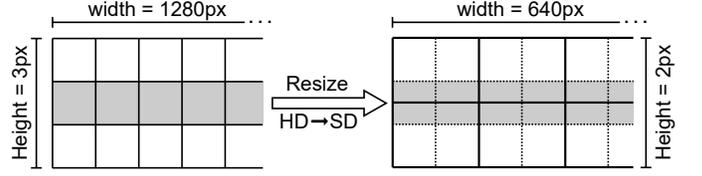


Fig. 5. Implementation of *resize* in Nova.

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).

Algorithm 2: Resize (Step i): HD \rightarrow SD

Public Input : $h_\alpha^{i-1}, h_\beta^{i-1}$
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  $i : 0 \rightarrow r_\beta$  do
4     for  $j : 0 \rightarrow \text{len}(\beta_i)$  do
5        $weight \leftarrow 2 - i$ 
6        $val \leftarrow (\alpha_{i,j*2}^c + \alpha_{i,j*2+1}^c) * weight$ 
7          $+ (\alpha_{i+1,j*2}^c + \alpha_{i+1,j*2+1}^c) * (3 - weight)$ 
8       assert  $6 > |val - \beta_{i,j}^c \times 6|$ 
9  $h_\alpha^i \leftarrow H(h_\alpha^{i-1} | H_\sigma(\alpha_i))$ 
10  $h_\beta^i \leftarrow H(h_\beta^{i-1} | H_\sigma(\beta_i))$ 
```

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. Fig. 6 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. However, we are obligated to maintain identical computations in each step within the folding scheme. To achieve this, we implemented the *crop* functionality 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 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 calculates the hash value of the crop area in the horizontal dimension. This result will be used

Algorithm 3: Crop (Step i)

Public Input : $h_\alpha^{i-1}, h_\beta^{i-1}, i_{row}, x, y$
Private Input : α_i
Public Output: $h_\alpha^i, h_\beta^i, i_{row}, x, y$

- 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** $y \leq i_{row} < y + height_{crop}$ **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 $i_{row} \leftarrow i_{row} + 1$
-

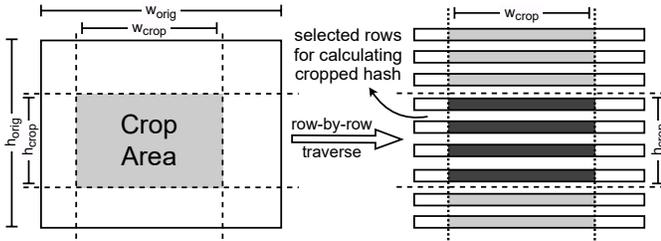


Fig. 6. Implementation of *crop* in a row-by-row setup.

if the current row (α_i) is within the vertical range of the crop area. Lines 4 to 8 implement this logic based on the index of the row. This way, h_β will only be updated if the current row is within the crop area. Finally, the next cumulative hash value of the original image, along with the index of the next step, is calculated in lines 8 and 9. It is important to note that since the initial public values (inputs of step 0) are visible in the final snark proof, any verifier can check whether these inputs were set correctly or not (e.g., the first i_{row} must be set to 0).

Note on runtime element selection based on inputs: Line 3 of Algorithm 3 indicates a dynamic subset selection of the array α_i based on runtime inputs given to the circuit. In the RICS setting, memory access cannot be decided at runtime and must be fixed during compilation, otherwise it will result in “*Non-quadratic constraints*” that are not allowed in Circom. Therefore, to implement such behavior, we have to employ $width_{crop}$ times multiplexers that support an input size of $|width_\alpha - width_{crop}|$ values. This results in an expensive RICS implementation that makes *crop* the most expensive transformation in our setup.

Note on If-Else statement: This design necessitates conditional calculations based on specific public inputs from the previous step. However, traditional if-else statements are not directly realizable in RICS format because they result in “*Non-quadratic constraints*”. To overcome this limitation, 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.

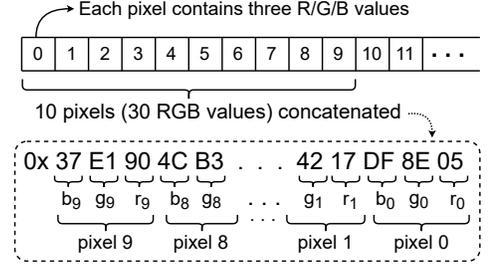


Fig. 7. Lossless compression of pixel values.

5) **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 calculating convolution and comparing the result against the transformed image. 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 \mathcal{K} of size $(2k + 1) \times (2k + 1)$.

Algorithm 4: Convolution (Step i)

Public Input : $h_\alpha^{i-1}, h_\beta^{i-1}, \mathcal{K}, h_{\alpha_{[(i-k) \rightarrow (i+k-1)]}}$
Private Input : $\alpha_{[(i-k) \rightarrow (i+k)]}, \beta_{[(i-k) \rightarrow (i+k)]}$
Public Output: $h_\alpha^i, h_\beta^i, h_{\alpha_{[(i-k+1) \rightarrow (i+k)]}}$

- 1 $weight \leftarrow \sum_{p,j:0 \rightarrow (2k+1)} \mathcal{K}_{p,j}$
 - 2 **foreach** $c \in [R, G, B]$ **do**
 - 3 **for** $j : 0 \rightarrow len(\alpha_i)$ **do**
 - 4 $val \leftarrow 0$
 - 5 **for** $m : 0 \rightarrow len(\mathcal{K})$ **do**
 - 6 **for** $n : 0 \rightarrow len(\mathcal{K})$ **do**
 - 7 $val += \alpha_{m,j+n}^c \times \mathcal{K}_{m,n}$
 - 8 $val \leftarrow \text{cap}_0^{255 \times weight}(val)$ // optional
 - 9 **assert** $weight > |val - \beta_{i,j}^c \times weight|$
 - 10 **for** $m : 0 \rightarrow 2k - 1$ **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 checksums are verified against each other. To this end, the first $2k$ input rows are compared with values passed by the previous step, and the checksum of the last $2k$ rows is passed to the next step (lines 10 to 12 of the algorithm).

V. OPTIMIZATION

A. Lossless Pixel Compression before Hashing

As noted in prior work [20], the calculation of the entire hash of inputs constitutes the majority of constraints in the circuit. To mitigate this, we propose a lossless compression technique for private inputs, reducing the number of constraints by nearly 30 times. The primary concept involves packing as many pixel values as possible within a field element.

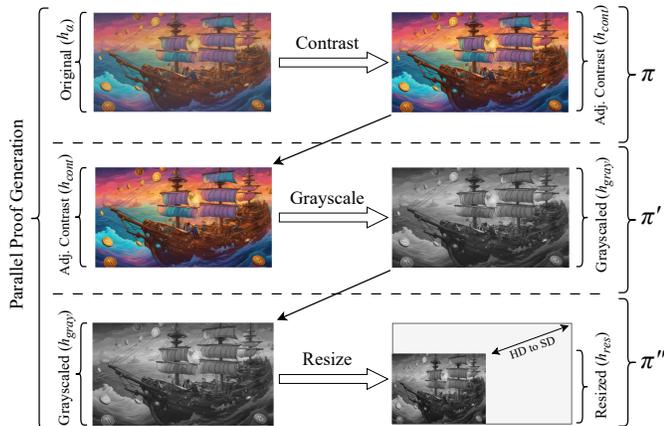


Fig. 8. Parallel generation of proofs for multiple transformations on an image.

Using Pallas/Vesta curves in our setup, each field element $e \in \mathbb{F}_q$ can range from 0 to $q-1$, where $q = 2^{254} + 455 \dots 353$. 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), resulting in a valid number in \mathbb{F}_q . Fig. 7 illustrates the compression method.

Now, in order to validate the integrity of an image represented in this structure, we require $30\times$ less number of hashes over the field \mathbb{F}_q . While the compression significantly reduces the number of required hashes, it necessitates "decompression" before validating the transformed pixel values. We provide more details regarding the decompression process in E.

B. Parallel Proof Generation

The implementation of VIMz leverages the memory and space efficiency provided by the Nova proving system. Our experimental results, detailed in Section VI, demonstrate a maximum memory peak of only 3.2 GB of RAM when proving transformations on an HD-resolution image and computing the hash of both the original and transformed images. This substantial advantage over previous work underscores the practicality of VIMz, even on consumer-level commodity hardware.

The modest memory requirements of VIMz enable simultaneous execution of multiple instances on commodity hardware, allowing for parallel proof generation of different transformation steps. Fig. 8 illustrates a scenario with three distinct transformations (*contrast*, *grayscale*, and *resize*) applied to an image. VIMz can concurrently run multiple instances, proving each effect separately but simultaneously. 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 Fig. 8), ensuring integrity between steps. The final proof comprises all hashes and their corresponding partial proofs (π , π' , and π''). Section VI includes experimental results for running multiple instances of VIMz in parallel on various platforms (Table V).

TABLE III. EXPERIMENTAL SETUP CONFIGURATION

		Dell Laptop Latitude 5531	Desktop	Dedicated Server
CPU	Model	12th Gen Intel® Core i5-12500H	AMD Ryzen 9 5900X	AMD Ryzen 9 7950X3D
	Freq.	3.3~4.5 GHz	3.7~4.8 GHz	4.2~5.7 GHz
	Cores* [‡]	12	12	16
	Cache [‡]	18 MB	70 MB	145 MB
Memory		16 GB	64 GB	128 GB
SSD		512 GB	512 GB	2 TB
OS		Ubuntu 22.04.2 LTS		

* Number of physical cores. [‡] Total cache size (L1+L3+L3).

VI. EXPERIMENTAL RESULT

In this section, we detail the setup of our benchmark platforms, outlining three distinct systems used for our experiments. We then evaluate and compare VIMz with prior work, considering prover performance and complexity, verifier complexity, and communication cost (proof size).

A. System Setup

Our setup can operate seamlessly without the need for extensive memory modules enabling us to practically generate proofs for various transformations, even on commodity hardware. To showcase the practicality of VIMz, we have executed multiple benchmarks across a diverse set of hardware, ranging from a mid-range Dell Latitude laptop to a dedicated server equipped with the latest Ryzen 9 7950X3D CPU. Table III outlines the systems used for benchmarking in our experiments.

B. Prover Complexity

a) HD Resolution: Table IV focuses on proof generation performance for each transformation independently. The key advantage of VIMz over prior work is its higher efficiency, resulting in significantly lower memory consumption, coupled with up to more than $3\times$ speedup while proving a transformation and calculating the hash of both the input and final image. VIMz, also has significantly lower key generation time (up to $33\times$). This aspect is particularly crucial in scenarios where the prover's storage is limited, and the capability to swiftly reproduce keys based on applied transformations is essential.

VIMz's proving time generally outperforms results of [20], except for the *crop* transformation. This issue arises from failures encountered with the C++ witness generator binary for the *crop* circuit produced by the Circom compiler; leading to runtime execution failures and necessitating the use of the less efficient web-Assembly (*wasm*) witness generator. This also results to nearly uniform performance across devices and longer proving times. Despite this, VIMz maintains significant memory efficiency even in the case of *crop*, requiring $55\times$ less RAM, highlighting its practicality on commodity hardware even without available optimizations.

One notable observation from Table IV is that VIMz performs similarly on a laptop equipped with a core i5 CPU and 16 GB of RAM compared to a more costly Desktop PC featuring 64 GB of RAM. The results suggest that the

TABLE IV. KEY AND PROOF GENERATION PERFORMANCE FOR HD RESOLUTION

	ZK-IMG [20]			VIMz (This work)						
	Time (s)		Peak	Time (s)						Peak Memory
	Key. Gen.	Proving	Memory	Laptop		Mid-range Desktop		High-end Server		
Transformation [♦]				Key. Gen.	Proving	Key. Gen.	Proving	Key. Gen.	Proving	
Crop [*]	432.6	557.1	139.1 GB	82.2	914.5 [‡]	80.6	923.5 [‡]	77.1	898.8 [‡]	3.2 GB
Resize [*]	431.8	556.8	139.1 GB	32.1	187.0	32.7	183.7	25.1	135.7	2.5 GB
Contrast [⊖]	823.8	1029.1	284.3 GB	29.9	479.4	28.8	462.4	22.9	371.7	2.4 GB
Grayscale [□]	-	-	-	21.9	279.6	21.0	271.2	17.8	240.6	1.3 GB
Brightness [⊖]	839.4	1027.9	287.1 GB	29.5	474.0	27.1	452.5	23.2	372.5	2.4 GB
Sharpness				35.6	614.1	34.8	594.2	28.7	455.8	2.8 GB
Blur	897.9 [*]	1300.3 [*]	305.3 [*] GB	33.6	555.3	30.7	538.1	26.8	406.0	2.5 GB

[♦]All measurements encompass proving the accurate execution of transformations, alongside computing the hash values of both the original and transformed images, ensuring trustless and privacy-preserving proofs of integrity. ^{*}HD to SD. [⊖]Contrast and brightness adjustments with an accuracy of 0.1. [□]Grayscale transformation with an accuracy of 0.001. [‡]The C++ witness generator for the `crop` circuit, produced by Circom, faced execution failures due to internal memory access bugs. Thus, we employed the less efficient web-Assembly (`wasm`) witness generator, lacking CPU architecture optimization. This led to almost uniform performance measurements across devices. ^{*}The authors of ZK-IMG [20] only report benchmarks for general convolution transformation.

TABLE V. PARALLEL PROOF GENERATION IN VIMz (HD RESOLUTION)

# of Trans.	List of Trans. [*]	Bottleneck [□]	Time (s)				Peak Memory [*]	
			Laptop [♦]		High-end Server		Laptop [♦]	High-end Server
			Max	Avg	Max	Avg		
2	Cont. Gray.	Cont.	573.2	287	378.3	190	3.0 GB	2.8 GB
	Cont. Sharp.	Sharp.	801.0	400	482.6	242	4.3 GB	4.4 GB
3	Cont. Gray. Resz.	Cont.	672.4	225	395.1	132	5.2 GB	5.0 GB
	Cont. Sharp. Resz.	Sharp.	883.1	295	501.7	168	6.1 GB	6.0 GB
4	Cont. Sharp. Resz. Gray.	Sharp.	1013.8	254	532.4	134	7.1 GB	6.7 GB
5	Cont. Sharp. Resz. Gray. Bright.	Sharp.	1234.7	247	577.3	116	8.8 GB	8.6 GB

^{*}Cont., Gray., Resz., Sharp., and Bright. stand for Contrast, Grayscale, Resize, Sharpness, and Brightness, respectively. [□]Indicates the transformation with the highest prove time, resulting in the *Max* time. [♦]Since the performance of the laptop is comparable to that of the mid-range desktop, we excluded the desktop results to avoid redundancy. ^{*}The peak memory reported here represents the total (summation) peak memory usage across all instances of VIMz executed in parallel.

performance is largely determined by the processing power of the CPU. This implies the potential for optimization by running multiple instances of VIMz concurrently to further improve efficiency.

b) Parallel Proof Generation: Utilizing parallelization techniques, as discussed in Section V-B, enables us to boost the performance further with minimal overhead. Table V presents comprehensive performance metrics for executing simultaneous proof generations in VIMz. 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 the *Max* time divided by the number 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 consumption results in individual transformations.

A notable observation from the results presented in Table V 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 IV, the overall average performance (*Avg* in Table V) increases by up to $3.5\times$, demonstrating the potential of parallel execution of VIMz instances in practical scenarios. This suggests that in the context of integrating VIMz with real-world editors such as Adobe Photoshop or GIMP, the proof generation process for each authenticated transformation could be initiated in the background immediately, providing a

seamless user experience.

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

c) 4K Resolution: Because of the memory efficiency of the Nova proving system, we can easily increase the resolution of the supported images to higher numbers, such as 3840×2160 (4K). Table VI provides performance measurements of prover in terms of proof generation time and peak memory usage 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.

C. Verifier Complexity

Table VII compares verification complexity of VIMz against ZK-IMG [20]. Our implementation requires verifying a SNARKs proof generated by Spartan [42], taking less than a second on a laptop. While the verification process of Halo2 proofs, used in [20], achieves faster verification times, there is little practical difference between the two approaches. Both benefit from zkSNARKs-based proving systems, which generally provide efficient verifiers in polynomial time. It is worth noting that both types of proofs are verifiable in Solidity [47],

TABLE VI. PROOF GENERATION PERFORMANCE FOR 4K RESOLUTION

Transform.	Laptop*		High-end Server	
	Proving (s)	Memory	Proving (s)	Memory
Crop	7259	9.3 GB	7302	10.3 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

* Since the performance of the laptop is comparable to that of the mid-range desktop, we excluded the desktop results to avoid redundancy.

TABLE VII. VERIFIER COMPLEXITY

VIMz		Crop	Resize	Cont.	Bright.	Gray.	Sharp.	Blur
		[20]	6 ms	6 ms	8 ms	9 ms	-	9 ms
VIMz	Server	0.5 s	0.3 s	0.3 s	0.3 s	0.2 s	0.3 s	0.3 s
	Laptop*	0.9 s	0.5 s	0.5 s	0.5 s	0.3 s	0.5 s	0.5 s

* Since the performance of the laptop is comparable to that of the mid-range desktop, we excluded the desktop results to avoid redundancy.

[48], enabling autonomous verification of the protocols built upon them.

D. Communication Cost

Table VIII compares proof sizes for various transformations with previous work. Unlike ZK-IMG [20], our implementation using Nova maintains nearly constant proof sizes, around 10KB for HD resolution transformations. This represents a general improvement, with proofs being at least 20% smaller across transformations, except for *crop* and *resize*, which see approximately 70% overhead. This is primarily due to Plonkish architecture of ZK-IMG, which excels in area selection. In contrast, VIMz generates final SNARKs proofs using Spartan, resulting in smaller sizes, up to 31% smaller for convolution-based transformations. Notably, proof sizes in Halo2 increase with image resolution, unlike VIMz, which maintains around 10 KB even at 4K resolution.

VII. TRUSTLESS C2PA-COMPATIBLE MARKETPLACE

In this section, we demonstrate how the proofs generated by VIMz can facilitate the creation of a trustless and decentralized marketplace compliant with C2PA standards. We propose a distributed infrastructure based on smart contracts for the autonomous and trustless management of ownership rights for both authentic images and their modified versions.

A. Authentic Media Marketplace

The cryptographic ability to prove edits on an image opens up possibilities for trustless copyright and ownership management. Assuming the original image remains untampered and is signed with a trusted and authorized key linked to a real or organizational entity, 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

TABLE VIII. PROOF SIZE (IN KB)

	Crop	Resize	Cont.	Bright.	Gray.	Sharp.	Blur
[20]	6.0	6.0	12.3	12.4	-	14.9*	
VIMz	10.5	10.2	10.3	10.3	9.9	10.3	10.3

* The authors of ZK-IMG [20] only report benchmarks for general convolution transformation.

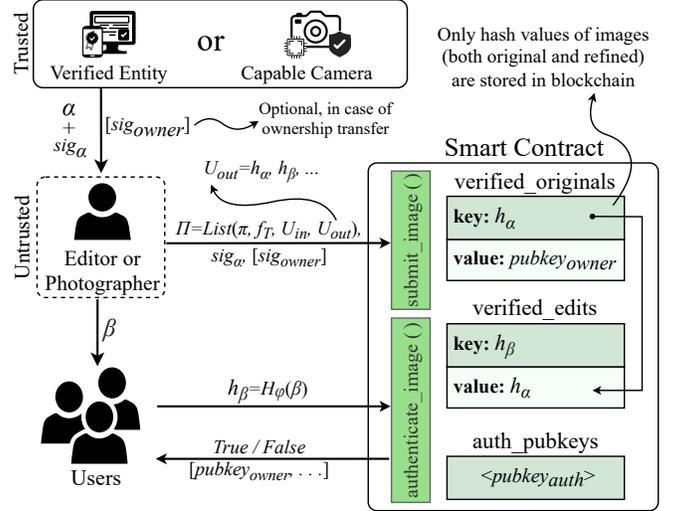


Fig. 9. Overview of the proposed protocol as a trustless infrastructure for a C2PA-compatible media market.

refined image. More precisely, the prover is required to submit a claim following the format outlined in Definition IV.1.

Fig. 9 offers an overview of the proposed protocol, illustrating two potential scenarios for submitting a new edited version of an original source:

- 1) **Sender has direct rights to the original image:** If the sender of the message, which includes both the edited image hash value and the proof, possesses the rights to the original image, the smart contract verifies the proof and completes the transaction with a successful status.
- 2) **Sender lacks rights to the original image:** They must supplement the message with a signed declaration from the entity holding the rights to the original image. This declaration confirms that the new editor of the image is the rightful owner. The contract processes the proof and authorization, finalizing the transaction accordingly.

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. Thus, transferring ownership of an image or an edited version also transfers ownership of all edits and versions of that image. In our ownership model, all edited versions of an image reference the original image, and the original image, in turn, points to its owner. By changing ownership, all edited versions are automatically transferred to the new 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 updating⁷.

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 idea. To address this, we propose a method allowing honest users to register commitments to an original image simultaneously with submitting the finalized edited version.

Algorithm 5 outlines the abstract functionality of the `submit_edited_image` method. The method takes a list of transformations and their respective proofs (Π), the signature of the original image (sig_α), and an optional signed message regarding ownership transfer (sig_{owner}). The algorithm iterates through the given list, retrieves the verifying key based on the specified function, and verifies the provided proof with respect to the given public inputs and outputs (U_{in} and U_{out}).

Algorithm 5: Autonomous Verifier Smart Contract

Data: tx_{msg} , $\Pi = \text{List}(\pi_{SNARK}, f_T, U_{in}, U_{out})$, sig_α ,
 $[sig_{owner}]$
Result: *Boolean* ($True|False$)

```

1 for  $i : 0 \rightarrow |\Pi|$  do
2    $\langle \pi_{SNARK}, f_T, U_{in}, U_{out} \rangle \leftarrow \Pi_i$ 
3    $d_v \leftarrow \text{retrieve\_SNARK\_key}(f_T)$ 
4   Assert  $\text{verify\_spartan}(d_v, \pi_{SNARK}, U_{in}, U_{out})$ 
5    $(h_\alpha, h_\beta) \leftarrow \text{get\_hashes}(U_{out})$ 
6    $id_\beta \leftarrow h_\beta$ 
7   if  $i == 0$  then
8      $pk_{sig} \leftarrow \text{verify\_and\_recover\_pk}(sig_\alpha, h_\alpha)$ 
9     if  $h_\alpha \notin \text{verified\_originals}$  then
10      Assert  $pk_{sig} \in \text{authorized\_pubkeys}$ 
11     if  $pk_{sig} \neq tx_{msg}.sender$  then
12       $\text{ownership}_{msg} \leftarrow \text{"transfer } h_\alpha \text{ to \{pk\}"}$ 
13      /* {pk} is the actual value
14      of  $tx_{msg}.sender$  */
15       $pk'_{sig} \leftarrow \text{verify\_and\_recover\_pk}(sig_{owner},$ 
16       $\text{ownership}_{msg})$ 
17      Assert  $pk'_{sig} == pk_{sig}$ 
18      Assert ( $\text{verified\_originals}[h_\alpha] == pk_{sig}$  or
19       $h_\alpha \notin \text{verified\_originals}$ )
20       $id_{owner} \leftarrow tx_{msg}.sender$ 
21       $id_\alpha \leftarrow h_\alpha$ 
22     else
23       $\langle \pi'_{SNARK}, f'_T, U'_{in}, U'_{out} \rangle \leftarrow \Pi_{i-1}$ 
24       $(h'_\alpha, h'_\beta) \leftarrow \text{get\_hashes}(U'_{out})$ 
25      Assert  $h'_\beta == h_\alpha$ 
26
27 Update  $\text{verified\_originals}[id_\alpha] \leftarrow id_{owner}$ 
28 Update  $\text{verified\_edits}[id_\beta] \leftarrow id_\alpha$ 
29 return True

```

The algorithm ensures that the first edit in the list of transformations is applied directly to the original image (α). It verifies the h_α value against the signature sig_α and checks

⁷We 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.

if h_α is listed as one of the `verified_originals`. If not, the signer of sig_α must be one of the `authorized_pubkey` defined in the contract. For the subsequent transformations in the list, the algorithm verifies the correctness of the given proof π_{SNARK} and ensures a chained sequence of inputs and outputs, where the outcome of each transformation serves as the input for the next transformation. If all checks pass and none of the assertions fail, the algorithm sets the sender of the message as the owner of α and adds the output of the last transformation as a new edited version of α .

B. Security Analysis

a) Denial of Service (DoS): Traditional client-server architectures, as seen in approaches like C2PA, are susceptible to DoS attacks due to the reliance on at least one available trusted server throughout the protocol. In contrast, the proposed method is non-interactive, eliminating the need for trusted parties to run servers at any stage. Users can simply query the status of a refined image from the blockchain. Moreover, the proposed market imposes no restrictions on the blockchain, and any permissionless blockchain, such as Ethereum, can serve as the foundation for running the smart contract system. Such blockchains inherently possess countermeasures against DoS attacks, as running a full node is feasible for any participant. It is worth noting that although DoS attacks are shown to be possible in some scenarios on EVM-based blockchains to overwhelm smart contracts with time-consuming transactions, such attacks, and their countermeasures are beyond the scope of this paper.

b) False Proof Generation: This scenario involves adversaries attempting to propose proof without possessing a correct witness. VIMz addresses this by employing unique circuits for each transformation, ensuring a precise and ordered list of transactions on the original source. Additionally, depending on the transformation, all configurations (e.g., contrast factor or starting point position in crop) are set as public inputs, enforcing correctness during the verification process. The chained calculation of the hash of the input image and the output result in each transformation also compels the prover to provide a legitimate path from the original source to the final refined image. Therefore, forging a set of proofs for a transformation is impossible while assuming the soundness property of the underlying proof systems, such as Nova [21] and Spartan [42].

c) Replay Attacks: In the proposed ZK statement, both h_α and h_β serve as public outputs of the proving circuit, acting as binding properties for each claim broadcasted to the blockchain. Consider an entity with the public key pk_1 successfully submitting a transaction adding h_β as the hash of a refined version of the original source with the hash h_α . In the case of an adversary attempting to replay such a claim without possessing pk_1 , they would need to provide a signed message sig_{owner} declaring the ownership transfer of h_α to the new owner. This requires forging the signature, which is rendered impossible within our defined adversary model. Moreover, if the owner of pk_1 submits the same successful message, it results in no additional gain other than incurring a wasted transaction fee.

d) Message Integrity: Full nodes on the blockchain verify the integrity of submitted messages through digital

signatures. Utilizing a public permissionless blockchain in our setup ensures the security of transactions against message manipulation, relying on the security of the underlying cryptographic primitives.

e) Malformed ZK Circuit Execution: A crucial advantage of zkSNARKs proving systems lies in their ability to ensure precise circuit execution without relying on the execution environment itself. Successful verification of a zkSNARKs proof guarantees that the prover has accurately and successfully executed the circuit. Consequently, any attempt to manipulate the proof generation process will lead to failure during the verification process due to the soundness property inherent in the underlying proving systems. It is important to note that the verifier possesses the verification key independently of the prover, thereby enforcing the proof to align with the exact expected proving circuit during the verification process.

f) Double-Spending: Owners of an image might attempt to transfer ownership to multiple entities. The designated infrastructure inherently prevents such actions. After the first successful transaction, which includes the ownership transfer, the owner of the original source gets updated within the contract. As a result, any subsequent ownership claims originating from the previous owner will be rejected.

VIII. DISCUSSION AND FUTURE WORK

This paper demonstrates the practicality of generating proofs for valid image manipulations using folding-based zkSNARKs, particularly the Nova proving system [21]. We present VIMz, an open-source platform as the first practical prover for authentic image transformations (in HD and 4K resolution) that can efficiently operate on consumer-level hardware. In comparison to existing methods, VIMz exhibits a peak memory usage of only 3.2 GB while proving the transformations in HD resolution. This results in a substantial reduction from the over 300 GB requirement in previous work [20] for the same task. In addition to this, VIMz achieves up to $3\times$ speedup in proof generation time. By leveraging parallelization, VIMz instances can concurrently prove multiple chained transformations, resulting in an additional $3.5\times$ speedup on average. Furthermore, we propose a trustless and distributed platform for copyright and media ownership, serving as a C2PA-compatible infrastructure for establishing authentic marketplace on public blockchains like Ethereum.

Moving forward, one key future objective of this work is to seamlessly integrate VIMz into established software platforms like Adobe Photoshop or GIMP, aligning with C2PA standards. Another direction in future work is to employ ongoing advancements in folding-based schemes [22], [23], [24], some of which have already been integrated into the original underlying Nova protocol [23]. Moreover, future efforts can optimize VIMz for higher efficiency to be more suitable for running on resource-constrained devices like mobile phones. Finally, given solid performance of VIMz, expanding its capabilities to support audio and video processing is a rewarding opportunity for further exploration.

REFERENCES

- [1] Samantha Cole. This deepfake of mark zuckerberg tests facebook’s fake video policies. <https://www.vice.com/en/article/ywyx/deepfake-of-mark-zuckerberg-facebook-fake-video-policy>. Accessed: 2023-10-05.
- [2] Paul Glynn. Sony world photography award 2023: Winner refuses award after revealing ai creation, Apr 2023.
- [3] DeepFaceLab library. <https://github.com/iperov/DeepFaceLab>. Accessed: 2023-10-05.
- [4] Midjourney. <https://www.midjourney.com/home/>. Accessed: 2023-10-04.
- [5] DALL•E. <https://openai.com/research/dall-e>. Accessed: 2023-10-04.
- [6] Stability AI. <https://stability.ai/home>. Accessed: 2024-01-02.
- [7] Nick Dufour and Andrew Gully. Contributing data to deepfake detection research, 2019. Accessed: 2024-01-05.
- [8] Mike Schroepfer. Creating a data set and a challenge for deepfakes. *Facebook artificial intelligence*, 5, 2019.
- [9] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-*df*: A large-scale challenging dataset for deepfake forensics. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3207–3216, 2020.
- [10] Yisroel Mirsky and Wenke Lee. The creation and detection of deepfakes: A survey. *ACM Computing Surveys (CSUR)*, 54(1):1–41, 2021.
- [11] C2PA Technical Specification. <https://c2pa.org/specifications/specifications/1.2/index.html>. Accessed: 2024-01-03.
- [12] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stempf. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy*, 18(2):56–60, 2020.
- [13] Simon Rohlmann, Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. Breaking the specification: Pdf certification. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1485–1501. IEEE, 2021.
- [14] Christian Mainka, Vladislav Mladenov, and Simon Rohlmann. Shadow attacks: Hiding and replacing content in signed pdfs. In *NDSS*, 2021.
- [15] Jens Müller, Dominik Noss, Christian Mainka, Vladislav Mladenov, and Jörg Schwenk. Processing dangerous paths. In *Network and Distributed Systems Security Symposium*. NDSS, 2021.
- [16] Jens Müller, Fabian Ising, Vladislav Mladenov, Christian Mainka, Sebastian Schinzel, and Jörg Schwenk. Practical decryption exfiltration: Breaking pdf encryption. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 15–29, 2019.
- [17] Vladislav Mladenov, Christian Mainka, Karsten Meyer zu Selhausen, Martin Grothe, and Jörg Schwenk. 1 trillion dollar refund: How to spoof pdf signatures. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1–14, 2019.
- [18] Assa Naveh and Eran Tromer. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 255–271. IEEE, 2016.
- [19] Dan Boneh Trisha Datta. Using zk proofs to fight disinformation, 2022.
- [20] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. ZK-IMG: Attested images via zero-knowledge proofs to fight disinformation. *arXiv preprint arXiv:2211.04775*, 2022.
- [21] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*, pages 359–388. Springer, 2022.
- [22] Wilson Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based snarks. *Cryptology ePrint Archive*, 2024.
- [23] Wilson Nguyen, Dan Boneh, and Srinath Setty. Revisiting the nova proof system on a cycle of curves. *Cryptology ePrint Archive*, Paper 2023/969, 2023. <https://eprint.iacr.org/2023/969>.
- [24] Abhiram Kothapalli and Srinath Setty. Hypernova: Recursive arguments for customizable constraint systems. *Cryptology ePrint Archive*, 2023.
- [25] Nova high-speed recursive arguments from folding schemes. <https://github.com/microsoft/Nova>. Accessed: 2024-01-03.
- [26] TruePic: Secure content transparency with C2PA. <https://truepic.com/>. Accessed: 2024-01-03.

- [27] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79:1102–1160, 2017.
- [28] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
- [29] David Balbás, Dario Fiore, Maria Isabel González Vasco, Damien Robissout, and Claudio Soriente. Modular sumcheck proofs with applications to machine learning and image processing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1437–1451, 2023.
- [30] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. Delegating computation: interactive proofs for muggles. *Journal of the ACM (JACM)*, 62(4):1–64, 2015.
- [31] Hankyung Ko, Ingeun Lee, Seunghwa Lee, Jihye Kim, and Hyunok Oh. Efficient verifiable image redacting based on zk-snarks. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pages 213–226, 2021.
- [32] The Halo2 zero-knowledge proving system. <https://zcash.github.io/halo2/>. Accessed: 2023-10-31.
- [33] Tianyi Wang and Kam Pui Chow. Noise based deepfake detection via multi-head relative-interaction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 14548–14556, 2023.
- [34] Luca Guarnera, Oliver Giudice, and Sebastiano Battiato. Deepfake detection by analyzing convolutional traces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 666–667, 2020.
- [35] Junke Wang, Zuxuan Wu, Wenhao Ouyang, Xintong Han, Jingjing Chen, Yu-Gang Jiang, and Ser-Nam Li. M2tr: Multi-modal multi-scale transformers for deepfake detection. In *Proceedings of the 2022 international conference on multimedia retrieval*, pages 615–623, 2022.
- [36] Amna Qureshi, David Megías, and Minoru Kuribayashi. Detecting deepfake videos using digital watermarking. In *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1786–1793. IEEE, 2021.
- [37] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.
- [38] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. *Cryptology ePrint Archive*, 2023.
- [39] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. {DIZK}: A distributed zero knowledge proof system. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 675–692, 2018.
- [40] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orru. Gemini: Elastic snarks for diverse environments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 427–457. Springer, 2022.
- [41] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security Symposium*, volume 2021, 2021.
- [42] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. *Cryptology ePrint Archive*, Paper 2019/550, 2019. <https://eprint.iacr.org/2019/550>.
- [43] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2022.
- [44] nova-scotia. <https://github.com/nalinbhardwaj/Nova-Scotia>. Accessed: 2024-01-03.
- [45] Sony unlocks in-camera forgery-detection technology. <https://www.sony.eu/presscentre/sony-unlocks-in-camera-forgery-proof-technology>, 2022. Accessed: 2024-01-08.
- [46] Circomlib. <https://github.com/iden3/circomlib/>. Accessed: 2024-02-07.
- [47] solidity-verifier. <https://github.com/lurk-lab/solidity-verifier/>. Accessed: 2024-02-07.
- [48] Halo2 solidity verifier. <https://github.com/privacy-scaling-exploration/s/halo2-solidity-verifier/>. Accessed: 2024-02-07.

APPENDIX A CONTRAST ADJUSTMENT ALGORITHM

Algorithm 6: Contrast (Step i)

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

```

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

```

APPENDIX B BRIGHTNESS ADJUSTMENT ALGORITHM

Algorithm 7: Brightness (Step i)

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

```

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

```

APPENDIX C CAP FUNCTIONALITY IMPLEMENTATION IN CIRCOM

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

```

1 template Cap(n) {
2
3   // n must be equal to ceil(log(max_limit))
4   signal input max_limit
5   signal input calced_value;
6   signal output final_value;
7
8   component lt[4];
9   component selector;
10  component gt_selector;
11
12  // find sign of calced_value
13  lt[0] = LessEqThan(n);
14  lt[1] = LessEqThan(n);
15  lt[0].in[1] <== 0 - calced_value;
16  lt[0].in[0] <== calced_value;
17  lt[1].in[0] <== max_limit;

```

```

18 lt[1].in[1] <== calced_value;
19
20 gt_selector = Mux1();
21 gt_selector.c[1] <== max_limit;
22 gt_selector.c[0] <== calced_value;
23 gt_selector.s <== lt[1].out;
24
25 selector = Mux1();
26 selector.c[0] <== gt_selector.out;
27 selector.c[1] <== 0;
28 selector.s <== lt[0].out;
29
30 final_value <== selector.out;
31 }

```

Listing 1. Cap functionality implementation in Circom.

APPENDIX D

IMPLEMENTATION OF IF-ELSE STATEMENT IN RICS

Algorithm 8 provides the logic to realize If-Else statement using multiplexer and comparison gates in RICS setting. Listing 2 implements this algorithm in Circom.

Algorithm 8: IF-Else Statements in RICS

```

1  $MUX.in[0] \leftarrow val_1$  //if  $i < y \vee i > y + h_{crop}$ 
2  $MUX.in[1] \leftarrow val_2$  //if  $y \leq i < y + h_{crop}$ 
3  $gte \leftarrow \text{Circuit (GreaterThanOrEqualTo)}$ 
4  $gte.in[0] \leftarrow i_{row}$ 
5  $gte.in[1] \leftarrow y$ 
6  $lt \leftarrow \text{Circuit (LessThan)}$ 
7  $lt.in[0] \leftarrow i_{row}$ 
8  $lt.in[1] \leftarrow y + height_{crop}$ 
9  $MUX.s \leftarrow gte.out \times lt.out$ 
10  $next\_crop\_hash \leftarrow MUX.out$ 

```

```

1 // if the row is within the cropped area
2 component selector = Mux1();
3 selector.c[0] <== prev_crop_hash;
4 selector.c[1] <== trans_hasher.hash;
5
6 component gte = GreaterEqThan(12);
7 gte.in[0] <== row_index;
8 gte.in[1] <== crop_start_y;
9
10 component lt = LessThan(12);
11 lt.in[0] <== row_index;
12 lt.in[1] <== crop_start_y + heightCrop;
13
14 selector.s <== gte.out * lt.out;
15 next_crop_hash = selector.out;

```

Listing 2. If-Else statement implementation in Circom.

APPENDIX E

DECOMPRESSION CIRCUIT

```

1 template Decompressor() {
2   signal input in;
3   signal output out[10][3];
4   component toBits = Num2Bits(240);
5   component toNum[10][3];
6   toBits.in <== in;
7   for (var i=0; i<10; i++) {
8     for (var j=0; j<3; j++) {
9       toNum[i][j] = Bits2Num(8);
10      toNum[i][j].in[0] <== toBits.out[i*24+j*8];
11    }
12  }
13 }

```

```

17 toNum[i][j].in[7] <== toBits.out[i*24+j*8+7];
18 out[i][j] <== toNum[i][j].out;
19 }
20 }
21 }

```

Listing 3. Decompressor circuit in circom.

ARTIFACT APPENDIX

The rise of deepfake technology has eroded the trustworthiness of media sources. However, before publication, raw media often requires refinements. Introducing VIMz, an efficient tool that verifies the authenticity of transformed or edited images relative to a verified original source. Built on recursive zkSNARKs from the Nova protocol [25] and Nova-Scotia [44] frontend (utilizing Circom for internal circuit definition), VIMz supports resolutions up to 4K (3840 × 2160) and beyond. Compared to the state-of-the-art, VIMz demonstrates significant reductions in prover complexity, achieving up to a 3× speedup in time and a 96× reduction in memory consumption, rendering it feasible on commodity hardware.

VIMz is fully open-source and available on a public GitHub repository. 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:

- **circuits**: Contains the underlying ZK circuits of VIMz in circom language.
- **contracts**: Contains high-level Solidity smart contracts (see Section VII) 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 [44].
- 3) **nova/src**: updating the main.rs file accordingly.

A. Description & Requirements

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>.

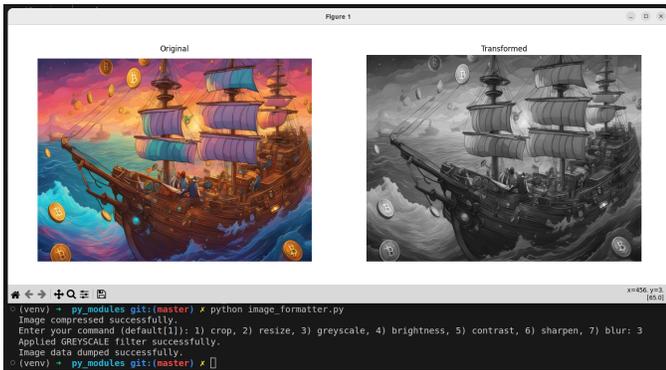


Fig. 10. Python GUI of VIMz.

2) Hardware dependencies: None.

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.

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.

B. 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.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash
```

```
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.git
```

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
```

C. 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 IV, Table V, and Table VI 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
```

D. Major Claims

In Section VI, 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.

E. Evaluation

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
```

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 IV). Once these results are verified, you can proceed to run multiple transformations in parallel to replicate the results shown in Table V.

[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 Fig. 11. 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.

F. 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
```



```
Verifiable Image Manipulation Using Recursive zkSNARKs
v1.1.0

Input file: samples/JSON/HD/transformation_contrast.json
Output file: contrast_hd.json
Selected function: contrast
Circuit file: circuits/contrast_step.r1cs
Witness generator: circuits/contrast_step_cpp/contrast_step
Image resolution: HD

Running NOVA with witness generator: circuits/contrast_step_cpp
Creating keys from R1CS took 17.342093067s
Number of constraints per step (primary circuit): 363101
Number of constraints per step (secondary circuit): 10347
Number of variables per step (primary circuit): 347737
Number of variables per step (secondary circuit): 10329
Creating a RecursiveSNARK...
RecursiveSNARK creation took 559.937217231s
Verifying a RecursiveSNARK...
RecursiveSNARK::verify: Ok(. . .), took 432.033612ms
Generating a CompressedSNARK using Spartan with IPA-PC...
CompressedSNARK::prove: true, took 12.995761847s
Data has been written to output.json
----- Load Data -----
Verifying a CompressedSNARK...
CompressedSNARK::verify: true, took 464.070025ms
```

Fig. 11. Example standard output generated by VIMz prover.

```
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 [44].
- 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.