HyperVerITAS: Verifying Image Transformations at Scale on Boolean Hypercubes

Garrett Greiner*
University of Utah
garrett.greiner@utah.edu

Toshi Mowery*†
University of Notre Dame
emowery@nd.edu

Pratik Soni
University of Utah
psoni@cs.utah.edu

Abstract

We present **HyperVerITAS**, a new zero-knowledge proof (ZKP) system for image provenance that enables scalable, efficient, and privacy-preserving verification of image transformations. HyperVerITAS builds upon the same minimal trust model as VerITAS (IEEE S&P '25), requiring trust only in the image source device, while treating the editing software as untrusted. Unlike VerITAS, which relies on FFT-intensive SNARKs and suffers from high memory overhead (up to 120 GB), HyperVerITAS leverages multilinear polynomial encodings over the Boolean hypercube to dramatically reduce both proving time and memory usage. Our design cleanly separates signature verification from image transformation, supports modular integration of multiple polynomial commitment schemes (including post-quantum constructions) and naturally extends to a wide range of affine image transformations.

We implement HyperVerITAS with two distinct commitment schemes (Brakedown and multilinear KZG) and evaluate it on full-system pipelines involving cropping and grayscaling. On commodity hardware (Apple M3, 36 GB RAM), HyperVerITAS generates proofs for 33 MP images using only 27 GB of RAM and 6.6 minutes of proving time, whereas VerITAS fails to scale beyond 4 MP. These results establish HyperVerITAS as a practical and scalable ZKP system for secure and efficient image provenance.

Keywords

zero-knowledge proofs, privacy-preserving image provenance, digital signatures

1 Introduction

In today's digital landscape, verifying the authenticity and provenance of images has become a critical concern. From scientific publishing to generative AI and secure documentation, there is an increasing demand to reliably determine where an image originated and how it may have been altered. This need becomes even more pressing as digital media is widely disseminated across platforms, often undergoing multiple rounds of processing and editing.

To address this, the Coalition for Content Provenance and Authenticity (C2PA) [15] has introduced a standard that allows trusted devices – such as cameras and machine learning servers – to cryptographically sign images at the point of capture or generation. These digital signatures, implemented using standard schemes like Schnorr [35], RSA [34], or ECDSA [20], attest to the authenticity and integrity of the image at the time of creation. Major camera manufacturers, including Sony, Leica, and Nikon, have begun integrating this functionality into their devices [25, 36]. Meanwhile, companies such as OpenAI – also a C2PA partner – have initiated efforts to embed attestations in AI-generated media [30].

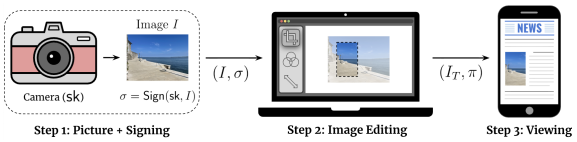

Figure 1: Image Editing Pipeline with a Trusted Camera

These cryptographic attestations empower downstream users or verifiers to validate the origin and integrity of an image—so long as the image remains unaltered. *However, images are rarely used or published as-is.* Transformations such as compression and resizing are common for performance and bandwidth optimization, while redaction, cropping, and filtering are applied for content refinement or hiding sensitive information. These transformations are often performed using third-party or open-source image editing software. Unfortunately, even benign changes invalidate the original signature, rendering it impossible to verify the edited image's provenance within the existing C2PA framework.

To accommodate post-capture transformations, the C2PA standard proposes extending the *chain of trust* to include the editing software. In this model, each editing application is responsible for re-signing the image after applying a transformation. While this preserves verifiability, it introduces new security risks. Specifically, it expands the trust model to include potentially untrustworthy editing tools and devices, requiring them to manage cryptographic signing keys securely. This assumption is particularly fragile in the case of open-source software or consumer-grade applications, where such guarantees are difficult, if not impossible, to enforce.

Threat Model. To address this limitation, Datta, Chen, and Boneh [12] propose a more realistic and minimal trust model for the image editing pipeline in Figure 1. In their approach, only the original source device (e.g., camera or ML server) is trusted to sign the image. *The editing software, on the other hand, is treated as untrusted.* Notably, this vision is already becoming a reality – Leica, for example, is developing cameras that integrate secure enclaves to safeguard signing keys within trusted execution environments [25].

The VerITAS System. Datta, Chen, and Boneh present an image provenance system under this threat model called VerITAS, which

*Equal Contribution.
†Work done while at the University of Utah as an undergraduate student.

Proceedings on Privacy Enhancing Technologies 2026(2), 58–75

https://doi.org/10.56553/popets-2026-0036

leverages zero-knowledge proofs (ZKPs) to prove that an output image I_T is the result of a valid transformation T applied to an input image I that was signed by a trusted source. ZKPs provide both *soundness*, eliminating the need to trust the editor, and *zero knowledge*, preserving the privacy of the input image. By leveraging a highly efficient ZKP known as a ZK-SNARK (zero-knowledge succinct non-interactive argument of knowledge), the resulting proof π – as well as the verification cost incurred by the end user – remains significantly smaller than the input image and the complexity of the transformation. But rather than using a monolithic general-purpose ZK-SNARK, VerITAS decomposes the proof into two components: (a) proof of knowledge of a valid signature, and (b) proof of correct application of a specified image transformation.

For the signature verification component, VerITAS assumes a hash-then-sign digital signature scheme using a lattice-based linear hash function. Leveraging standard sigma protocols [10, 35], designing such a ZKP then reduces to proving knowledge of a *low-norm* pre-image vector x under a compressing linear map such as $y = \mathbf{A} \cdot x$ for a finite-field matrix $\mathbf{A}$. The authors design a custom proof system for this step using the interactive oracle proof (IOP) and polynomial commitment paradigm. For the image transformation component, they use PLONK [16], an efficient general-purpose SNARK also based on the IOP framework. This hybrid approach enables VerITAS to handle high-resolution images—up to 30 megapixels – and a range of real-world transformations with sub-15-minute proving time on high-end machines.[1] Prior work on ZKPs for image transformations either predates the C2PA standard [29] or scales only to lower resolution images such as 720p (= 0.92 MP) [21].

Limitations of VerITAS. Despite its impressive scalability and modular architecture, VerITAS exhibits several key limitations w.r.t. scalability. Most notably, the prover's memory consumption is prohibitively high. Their experiments were conducted on a server with 131 GB of RAM, and memory usage during proof generation ranged from 75 GB to 120 GB for a single 30 MP (∼90 MB) image. Such requirements are far beyond what is available on typical commodity hardware – e.g., laptops running standard image editing software. Even with these substantial resources, the reported proof generation time remains in the sub-15-minute range. While proofs may only need to be generated once per transformation, a scalable system should strive for much faster runtimes to support real-world deployment at scale.

Upon closer analysis, much of this overhead stems from the use of *univariate* IOPs and *univariate* polynomial commitment schemes, which incur substantial computational and memory costs due to the extensive use of large FFTs for operations like polynomial division, multiplication, and interpolation.

Secondly, although the image transformation proofs are comparatively lightweight, the reliance on general-purpose SNARKs like PLONK [16] introduces additional costs. In particular, translating image transformation logic into rank-1 constraint systems (R1CS) constitutes a significant portion of the total proving time.[2]

Another scalability bottleneck arises when an editor applies a sequence of transformations – e.g., cropping followed by grayscale conversion. In VerITAS, such a pipeline must be decomposed into individual proofs with commitments to intermediate images, further compounding the complexity, size, and runtime of the overall proof.

These limitations motivate the central question we investigate:

Can we design scalable zero-knowledge proof systems for image transformations that support 30-megapixel images—while remaining practical on commodity hardware?

In this paper, we explore new techniques for achieving this goal, focusing on reducing prover complexity while preserving cryptographic soundness and real-world applicability.

1.1 Our Contributions

We present **HyperVerITAS**, a zero-knowledge proof (ZKP) system for image provenance that supports scalable verification of image transformations while enabling efficient proof generation even on commodity hardware. The key innovation in HyperVerITAS is its use of *multilinear polynomials over the Boolean hypercube*, in contrast to the univariate polynomials employed in prior work. This design choice is motivated by growing evidence that multilinear encodings lead to dramatically lower memory usage for SNARK provers, particularly in scenarios where the witness is large [4, 7–9].

Our work is inspired by the architecture of VerITAS but takes a different technical path. In particular, HyperVerITAS builds on the insight of HyperPlonk [11], which showed how to generalize the PLONK protocol to the Boolean hypercube, avoiding expensive FFTs and significantly reducing prover memory. We adopt a similar strategy to adapt VerITAS's two-component design (signature verification and transformation verification) to the hypercube domain—hence the name **HyperVerITAS**.

Formally, HyperVerITAS is a ZK-SNARK for the following relation, parameterized by a verification key vk of a digital signature scheme $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ issued by a trusted image capture or generation device:

$$R_{\mathsf{vk}} = \{((I_T, T); (I, \sigma)) : I_T = T(I) \wedge \mathsf{Vrfy}(\mathsf{vk}, I, \sigma) = 1\} \ . \quad (1)$$

As in VerITAS, we split the relation into two sub-proofs:

(1) A ZKP of knowledge of a valid signature (I, σ) under vk.
(2) A ZKP that a transformation T was correctly applied to obtain $I_T = T(I)$.

Despite the conceptual similarity, the underlying proof systems in HyperVerITAS differ significantly from those in VerITAS, especially in their reliance on multilinear polynomials encoded over the Boolean-hypercube. Below, we describe both components in detail.

Proving Knowledge of a Signature. In our model, the image source (camera or ML server) signs a hash of the original image. Although SHA-256 is a natural candidate for this hash function, proving knowledge of a large (e.g., 90 MB) preimage under SHA-256 in ZK is computationally prohibitive. Even SNARK-friendly hashes like Poseidon do not scale well: VerITAS reports memory exhaustion for the PLONK prover for even 1MP images [12].

[1] For this fastest implementation, VerITAS reports a proving time of 10.25 minutes for proving knowledge of a pre-image under the linear hash function for a single-channel of a 30 MP image, and an additional 3.37 minutes for proving correct application of a grayscaling transformation on the same image.

[2] Table 3 in the VerITAS ePrint version (https://eprint.iacr.org/2024/1066.pdf) separately reports setup and proving times. As confirmed by their public codebase available at http://github.com/zk-VerITAS/VerITAS, the reported setup time primarily reflects the cost of generating the R1CS circuit and the extended private witness.

To address this, HyperVerITAS supports two modes, each tailored to different deployment settings:

- **Mode 1**: Designed for constrained devices (e.g., C2PA enabled cameras).
- **Mode 2**: Designed for compute-rich environments (e.g., ML servers such as those used by OpenAI).

Each mode uses a different hash function strategy that balances prover and signer efficiency.

Mode 1 (Camera setting): Linear Hash with Range Proof. In this mode, the image is hashed using a *lattice-based linear hash function* (e.g., Ajtai's hash [1]), which maps a high-dimensional image vector into a smaller field vector using a random matrix. The resulting hash is then signed using a conventional signature scheme such as RSA, Schnorr, or ECDSA.

This linear structure is highly amenable to efficient zero knowledge proofs as evidenced in prior work [6, 24]. VerITAS implements this using univariate IOPs and FFT-based polynomial commitments. In contrast, HyperVerITAS ports this approach to the multilinear setting using Boolean-hypercube encodings, avoiding FFTs and enabling low-memory proving even on laptops. Despite its simplicity, our evaluation shows that this yields significant gains in both speed and memory – e.g., we are able to prove knowledge of the hash pre-image for a 30 MP image in under 6.6 minutes (along with the image transformation proof) using just 27 GB RAM on a standard laptop (see Section 5.2.1).

However, Ajtai's hash is only collision-resistant for *low-norm* preimages, relying on the hardness of the Short Integer Solutions (SIS) problem. To maintain soundness, we include a *range proof* (inspired by HyperPlonk) that ensures the witness values fall within a small range (i.e., [0,255]).

Our design is also modular in its use of polynomial commitment schemes. We instantiate HyperVerITAS with five different schemes, with the two most optimal schemes being:

- The **PST** scheme [31] - a multilinear variant [31] of the pairing-based univariate KZG scheme [22], and
- A post-quantum, code-based scheme like **Brakedown** [17].

Surprisingly, despite the expensive pairing operations, our PST-based variant outperforms VerITAS, which uses FRI – a code-based FRI polynomial commitment [5]. We attribute these speedups to the elimination of expensive FFT operations. Further, the linear hash functions utilized by HyperVerITAS require lightweight computation, and it is feasible that a camera could run these operations. Note that this requires cameras which support linear hash functions and cryptographic operations in the hash-then-sign signature schemes. But this is not an unreasonable expectation, as major camera manufacturers, including Sony and Leica, are already integrating various hash-then-sign signature schemes into their devices [25, 36].

Mode 2 (ML Server setting): Commitment-as-Hash. In compute-rich environments, we explore using a *multilinear polynomial commitment (PC)* directly as the hash function. This approach treats the commitment (e.g., via Brakedown) as a cryptographic hash to the image data. Since these commitments are collision-resistant over arbitrary domains, we eliminate the need for range proofs.

Although generating such a commitment is computationally expensive, we find that the *memory footprint is significantly lower* than VerITAS. For example, computing the lattice-based hash in VerITAS for a 30 MP image reportedly consumes over 20 GB of RAM—just for hashing. Our PC-based hash avoids this by eschewing FFTs and enabling streamed encoding of the image.

Furthermore, once the commitment is generated, the proof only needs to demonstrate that the prover knows the committed image, reducing the signature verification to a trivial check. As we detail below, we fold this proof of knowledge into the image transformation proof itself, thereby eliminating most of the computational overhead and enabling significant speed-ups over Mode 1.

Proving Image Transformation Correctness. A central insight behind our design is that many common image transformations—such as cropping, redacting, and even some non-linear operations like grayscaling—can be expressed as *affine* functions over the image pixel space. Specifically, transformations like cropping or redacting can be represented as $I_T = \mathbf{A} \cdot I + e$, where $\mathbf{A}$ is a publicly known Boolean matrix that selects pixels to retain or modify, and e is a constant vector (e.g., with entries set to 255 to simulate white-out redactions). Even non-linear transformations like grayscaling, which involve rounding, can be approximated by such affine relations over the integers, where e captures the rounding error and remains low-norm. We formally define the class of supported affine functions in Section 3.2.

Building on this observation, we extend our linear pre-image proof system to build a proof of knowledge of pre-image of affine transformations. This approach enables substantial simplification compared to VerITAS, which translates high-level transformation logic into R1CS circuits—a process that dominates its prover runtime. By avoiding R1CS, HyperVerITAS significantly reduces both proving time and memory usage for common transformations.

For transformations like redaction, the offset vector e is fixed and can be made public. However, for operations like grayscaling, e encodes per-pixel rounding errors, which may leak information about the original image. In fact, revealing e alongside the transformed image may allow a malicious verifier to reconstruct the original image with high fidelity. To preserve privacy in such cases, we enhance our affine proof system to commit to the error vector rather than reveal it, and attach a range proof to demonstrate that the committed values lie within a small bounded range. In contrast, VerITAS reveals the error vector e, making it susceptible to the privacy attack described above.

Our affine approach not only yields significant performance improvements, but also introduces powerful composability. When an editing pipeline involves multiple sequential affine transformations-such as cropping followed by grayscaling-the composition of these operations remains affine. This allows HyperVerITAS to prove the correctness of an entire transformation sequence via a *single* affine proof. In contrast, a natural extension of VerITAS would require committing to each intermediate image and generating separate PLONK proofs for each transformation. This leads to substantial overhead in the form of additional polynomial commitments, multiple R1CS circuits, and larger proof sizes. Overall, by natively encoding image transformations as affine maps over the Boolean hypercube, HyperVerITAS delivers a more scalable, modular, and performant framework for transformation verification, especially in complex editing workflows.

Implementation. Our implementation builds on the HyperPlonk IOP primitives provided by the Plonkish library to perform key multilinear checks—such as sum-checks, range-checks, and product-checks—over the Boolean hypercube. To showcase the flexibility of our modular design, we instantiate HyperVerITAS using five distinct multilinear polynomial commitment schemes: the PST scheme [31], the code-based Brakedown scheme [17], Basefold [39], Basefold-Fri [39], and ZeromorphFri [23]. While Plonkish natively supports these various PCS, integrating them required us to reimplement some of HyperPlonk's IOP routines to interface with the Plonkish API. Specifically, we implemented product check, multset check, and range check. Our complete implementation—including the IOP reimplementation and PCS integration—is publicly available at [19].

Currently, the prototype supports two representative transformations: cropping (a public affine transformation) and grayscaling (a secret affine transformation). Thanks to the modular architecture, the system can be easily extended to support additional transformations as described earlier. In our implementation, each image with n pixels is represented as a $n \times 3$ matrix, where each row corresponds to the three values (red, green, and blue) that represent one pixel of the image, and each entry stores an integer in the range $[0, 255]$.

The public implementation of VerITAS provides only the core building block proofs such as grayscaling proof for all three channels, but support cropping proof and hash pre-image proof for a single channel. For a fair comparison, we extend the VerITAS codebase to handle three-channel (RGB) images and implement a complete end-to-end system that proves the relation defined in Equation (1).
Evaluation. We conduct a detailed evaluation of HyperVerITAS against VerITAS across two computing environments and for varying image sizes, focusing on the two representative transformations: crop and grayscale. Our experiments compare both systems' performance in terms of proof generation time, prover memory consumption, verification time, and proof size.

Commodity Hardware. We benchmark full-system implementations of VerITAS and HyperVerITAS on an Apple MacBook equipped with an M3 chip and 36 GB of RAM. This configuration better reflects the constraints of typical commodity hardware compared to the 131 GB RAM AWS server used in VerITAS's original evaluation.

By *full-system implementation*, we mean a setup that proves both knowledge of the signature and correctness of the image transformation across all three RGB channels. In summary, HyperVerITAS instantiated with the Brakedown commitment scheme (HV[BD-127]) successfully generates a proof for cropping a 33 MP image in just 6.6 minutes, while using only 27GB of memory (Table 3). In contrast, VerITAS using FRI-based commitments (V[FRI]) runs out of memory for images larger than 4 MP. Even at this smaller size, HyperVerITAS with the PST-based commitment scheme (HV[PST]) achieves roughly **50% faster** proving times compared to VerITAS-FRI. We discuss similar trends for grayscale in Table 7.

These results provide strong evidence that HyperVerITAS offers substantial improvements in both memory efficiency and speed on commodity hardware, primarily due to its use of multilinear encodings and avoidance of FFT-heavy operations.

High-Memory Server. We also evalute HyperVerITAS and VerITAS on an AWS EC2 *r5d.4xlarge* instance with 128 GiB RAM and 16 vCPUs. As illustrated in Figure 7 and Figure 9, HyperVerITAS – Brakedown consistently outperforms VerITAS in both full system crop and grayscale, across a variety of PCS. Notably, HyperVerITAS Brakedown (127) is able to generate crop and grayscale proofs for 33MP images, while VerITAS's prover runs out of memory already at 8MP for crop, and 4MP for grayscale. This demonstrates that the affine encoding approach of HyperVerITAS leads to substantial practical benefits even on high-end hardware.

Hashing Modes. We further compare the time and memory requirements of computing the Mode 2 hash function in both systems. HyperVerITAS supports five PCS - based hash functions—using PST, Brakedown, Basefold, BasefoldFri, and ZeromorphFri — which are substantially more memory-efficient than the FRI-based hash function used in VerITAS. As summarized in Table 4, HyperVerITAS – Brakedown reports a PCS hash time of 4.18 seconds and requires only 10.19 GB RAM, whereas VerITAS reports a time of 19.84 seconds with 18.90 GB RAM in their experiments [12]. This validates our design choice to avoid FFT-heavy primitives in favor of multilinear encodings over the Boolean hypercube.

Across both commodity and server-grade environments, HyperVerITAS demonstrates superior scalability, faster proof generation, and significantly reduced memory usage compared to VerITAS. These benefits stem from our system's native support for multilinear encodings, its avoidance of FFTs, and its efficient handling of affine image transformations. Full benchmarking results and reproducibility scripts are available in our open-source repository [19].
Summary. HyperVerITAS introduces a scalable and memory - efficient framework for zero-knowledge image provenance, leveraging multilinear polynomial encodings over the Boolean hypercube. Our contributions span theoretical design, practical engineering, and empirical validation:

- We present a **ZK-SNARK system optimized for memory efficiency**, built upon multilinear IOPs and Boolean-hypercube encodings, avoiding FFT-intensive operations common in prior systems like VerITAS.
- We support two deployment modes tailored to real-world use cases:
  - A **lightweight camera mode** that uses a lattice-based linear hash function along with range proofs for low-norm enforcement.
  - A **server-grade mode** where the hash is instantiated via succinct multilinear polynomial commitments, enabling stronger soundness guarantees without range proofs.
- Our design is **modular and commitment-agnostic**, supporting both pairing-based (PST) and post-quantum (Brakedown) polynomial commitment schemes, with minimal integration overhead.
- We enable **efficient affine transformation proofs**, reducing the need to encode transformations as R1CS circuits and allowing for composition of multiple edits within a single proof—significantly reducing complexity and proof size compared to VerITAS.
- Through our implementation and evaluation, we demonstrate **substantial practical gains**: lower memory consumption (even on commodity hardware), faster proof generation, and scalability to high-resolution (e.g., 30 MP) images—surpassing VerITAS across all key metrics.

We believe HyperVerITAS offers the first practical and scalable ZK framework for image transformation verification on commodity hardware, making content provenance cryptographically robust and broadly deployable.

1.2 Comparison with Other Related Work

Comparison with VIMz and TilesProof-MT. While the main focus of our comparison is with VerITAS, there have been other recent works that provide systems to prove image provenance for large images on commodity hardware. VIMz [14] utilizes folding-based ZK-SNARKS, where the transformation is proved in incremental steps, reducing the memory overhead. TilesProof-MT's [28] approach is to split the image into different regions (called tiles), and then the proofs are done with respect to these smaller tiles.

We provide a comprehensive comparison between HyperVerITAS, VerITAS, VIMz, and TilesProof-MT in Section 5. While HyperVerITAS outperforms VerITAS in almost all aspects, the comparison with VIMz and TilesProof-MT reveals some trade-offs. VIMz and TilesProof-MT offer even lower memory overheads than HyperVerITAS, while also providing much smaller proofs (KBs vs MBs). However, these approaches do suffer in that their prover is very slow compared to HyperVerITAS's prover. For 33MP images, HyperVerITAS can generate proofs in roughly 6 minutes, while VIMz takes nearly 2 hours, and TilesProof-MT takes over 30 minutes (for the same relation). So depending on the application, one would use HyperVerITAS over VIMz and TilesProof-MT, or vice versa.

Comparison to zkCNN. Although zkCNN [26] was built for a different application domain, the work proposed a sum-check based protocol to efficiently prove 2-D convolutions. Since certain image transformations, such as blurring, can be viewed as convolutions, the 2-D convolution protocol could be useful here as well. For images of size n, the protocol introduced in zkCNN exhibits prover time of $O(n)$, while verifier time and proof size both have a complexity of $O(\log^2 n)$. In comparison, our inner product protocol that we use to prove image transformations has the same prover runtime of $O(n)$, but the verifier time and proof size have a slightly better complexity of $O(\log n)$.

Organization. Our work is organized as follows: Section 2 introduces the syntax and relevant cryptographic tools used throughout the paper, Section 3 presents the design of HyperVerITAS and the protocols we developed, Section 4 details our implementation for HyperVerITAS, and Section 5 contains the results for our comprehensive set of experiments and comparisons.

2 Preliminaries

Basic Notation. We denote the security parameter by λ. We say that a function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ is negligible in λ if for every $k \in \mathbb{N}$, there exists a $\lambda_0 \in \mathbb{N}$ such that for every $\lambda \geq \lambda_0$, we have that $|f(\lambda)| \leq 1/\lambda^k$. We denote the output of a random process $A(\cdot)$ by $x \leftarrow A(\cdot)$. We denote the probability that an event A happens conditioned on an event B by $\Pr[A \mid B]$. We use $[k]$ to denote the set $\{0, 1, \ldots, k-1\}$ for any $k \in \mathbb{Z}^{\geq 0}$, and $[a : b]$ to denote the set $\{a, a+1, \ldots, b-1, b\}$ for any $a, b \in \mathbb{Z}$ with $a < b$. We use $\odot$ to denote matrix-vector or matrix-matrix multiplication. For any vectors $\mathbf{u}$ and $\mathbf{v}$, we denote the inner product of $\mathbf{u}$ and $\mathbf{v}$ by $\langle \mathbf{u}, \mathbf{v} \rangle$.

$\mathbf{I}$	Image in $[256]^{n \times 3}$
$\mathbf{I_t}$	Transformed image in $[256]^{m \times 3}$
T	Image transformation $\mathsf{T} : [256]^{n \times 3} \to [256]^{m \times 3}$
$\mathbf{A}$	Hash matrix $\mathbf{A} \in \mathbb{F}^{h \times n}$
$\odot$	Matrix-Matrix, Matrix-Vector multiplication
$\langle \mathbf{u}, \mathbf{v} \rangle$	Inner product between vectors $\mathbf{u}, \mathbf{v}$

Table 1: Frequently Used Notation

When we define various protocols, we will use the following syntax to denote public and private inputs. When we have a tuple of inputs $(a, b, c; d, e)$, the variables listed before the semicolon are public, and the ones after are private. We often work with different proof systems that are defined w.r.t. a relation which is a set of statement/witness pairs, typically denoted by R.

We use $\mathbb{F}$ to denote fields throughout the paper. Generally, we use lower case letters for scalar values, boldface lower case for vectors, and boldface uppercase for matrices.

Notation for Polynomials. A multilinear polynomial is a multivariate polynomial that is linear in each of its variables (i.e no variable has degree greater than two). The total degree of a polynomial is the maximum of the sums of the degrees of the variables in each monomial term. We will use $\mathbf{x}$ to denote vectors representing points used for evaluation, and $\mathbf{X}$ to denote vectors of indeterminates. We let $\mathbb{B}_\mu$ denote the boolean hypercube, that is, $\mathbb{B}_\mu := \{0, 1\}^\mu$.

Lemma 2.1 (Multilinear Extensions). *For every function $f : \mathbb{B}_\mu \to \mathbb{F}$, there is a unique multilinear polynomial $\hat{f} \in \mathbb{F}\left[X_1, \ldots, X_\mu\right]$ such that $\hat{f}(\mathbf{b}) = f(\mathbf{b})$ for all $\mathbf{b} \in \mathbb{B}_\mu$. We call $\hat{f}$ the multilinear extension of f, and $\hat{f}$ can be expressed as:*

$$\hat{f}(\mathbf{X}) = \textstyle\sum_{\mathbf{b} \in \mathbb{B}_\mu} f(\mathbf{b}) \cdot eq(\mathbf{b}, \mathbf{X}) \,,$$

where $eq(\mathbf{b}, \mathbf{X}) := \prod_{i=1}^{\mu} (\mathbf{b}_i \cdot X_i + (1 - \mathbf{b}_i)(1 - X_i))$.

Lemma 2.2 (Schwartz-Zippel Lemma). *Let $f \in \mathbb{F}\left[X_1, \ldots, X_\mu\right]$ be a non-zero polynomial of total degree d over a finite field $\mathbb{F}$. Let S be any finite subset of $\mathbb{F}$. Then we have that:*

$$\Pr\left[f(r_1, \ldots, r_\mu) = 0\right] \leq d/|S| \,,$$

where $(r_1, \ldots, r_\mu)$ are μ field elements selected independently and uniformly from the set S.

Notation for Images and Transformations. We define the frequently used notation for images and transformations in Table 1. We consider an image $\mathbf{I} \in [256]^{n \times 3}$ where the columns can be thought of as the three color channels of an image (red, green, blue), and n is the length of the image (in pixels). We use $\mathsf{T} : [256]^{n \times 3} \to [256]^{m \times 3}$ to denote an image transformation, and $\mathbf{I_t} \in [256]^{m \times 3}$ to denote a transformed image. Let $I_{k \times k}$ be the $k \times k$ identity matrix.

2.1 Basic Cryptographic Tools

Definition 2.3 (Hash Function). *A hash function $\mathcal{H} = (\mathsf{KGen}, \mathsf{Eval})$ is defined with respect to a keyspace $\mathcal{K}$, a message space $\mathcal{M}$, and range $\mathcal{Y}$. The two algorithms of $\mathcal{H}$ are defined as follows:*

- $\mathsf{k} \leftarrow \mathsf{KGen}(1^\lambda)$*: The key generation algorithm takes as input the security parameter, and outputs a key* $\mathsf{k} \in \mathcal{K}$.

- $y \leftarrow \mathsf{Eval}(\mathsf{k}, m)$: *The evaluation algorithm takes as input a key* $\mathsf{k} \in \mathcal{K}$ *and a message* $m \in \mathcal{M}$*, and outputs a hash* $y \in \mathcal{Y}$.

Note that $\mathsf{Eval}(\mathsf{k}, \cdot)$ *must be efficiently computable for all* $\mathsf{k} \in \mathcal{K}$*. In this work, we work with compressing functions where* $|\mathcal{Y}| << |\mathcal{M}|$*. We will use* $\mathsf{H}(\mathsf{k}, m)$ *as a short hand for* $\mathcal{H}.\mathsf{Eval}(\mathsf{k}, m)$.

We require a weaker variant of standard collision resistance which is restricted to pre-images with low norm.

Definition 2.4 (Low-Norm Collision-Resistant Hash Family). *For some integer* $B > 0$*, a hash function* $\mathcal{H}$ *is* B*-norm collision resistant if for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* ϵ *such that for all* $\lambda \in \mathbb{N}$*, we have that:*

$$\Pr\left[\begin{array}{c} m_1 \neq m_2 \wedge \\ \mathsf{H}(\mathsf{k}, m_1) = \mathsf{H}(\mathsf{k}, m_2) \wedge \\ ||m_1||_\infty \leq B \wedge ||m_2||_\infty \leq B \end{array} \middle| \begin{array}{c} \mathsf{k} \leftarrow\!\!\$\ \mathcal{H}.\mathsf{KGen}(1^\lambda) \\ (m_1, m_2) \leftarrow \mathcal{A}(\mathsf{k}) \end{array}\right] \leq \epsilon(\lambda) \ .$$

We use Ajtai's linear hash function whose security is based on the hardness of the short-integer solutions problem [2].

Definition 2.5 (Linear Hash Family). *The hash takes as input a data vector* $\mathbf{v} \in \mathbb{Z}_q^n$*. Given any matrix* $\mathbf{A} \in \mathbb{F}_q^{h \times n}$ *(where* q *is prime and* $h < n$*), we define the lattice hash as:*

$$\mathsf{H}_\mathbf{A}(\mathbf{v}) := \mathbf{A} \odot \mathbf{v} \pmod{q} \ . \tag{2}$$

This lattice hash function is low norm collision-resistant under the hardness of Short Integer Solution (SIS) problem [2]. An instance of the SIS problem is defined by a random matrix $\mathbf{A} \leftarrow \mathbb{F}_q^{h \times n}$*, and a set of parameters* $h, n, q, B \in \mathbb{N}$*, where* q *is prime and* $h < n$*. An adversary solves an instance of the SIS problem if they can find a non-zero vector* $\mathbf{v} \in \mathbb{Z}^n$ *such that* $||\mathbf{v}||_\infty < B$ *and* $\mathbf{A} \odot \mathbf{v} = 0 \pmod{q}$*. Note that for sufficiently large* $h \in \mathbb{N}$*, the SIS problem is conjectured to be hard for any* $n, q, b \in \mathbb{N}$ *such that* $q > b \cdot \mathrm{poly}(h)$ *and* $n > h \cdot \log_2(q)$.

We introduce another cryptographic primitive: digital signatures.

Definition 2.6 (Digital Signature). *A digital signature scheme is defined as* $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$*, where the three algorithms* KGen, Sign, *and* Vrfy *are defined as follows:*

- $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(1^\lambda)$: *The key generation algorithm takes as input the security parameter, and outputs a key pair* $(\mathsf{sk}, \mathsf{vk})$.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: *The signing algorithm takes as input the signing key* sk *and a message* m*, and outputs a signature* σ.
- $b := \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$: *The verification algorithm takes as input the verification key* vk*, a message* m*, and a signature* σ*, and outputs a bit* $b \in \{0, 1\}$.

A digital signature scheme is secure if it satisfies two properties: correctness *and* strong unforgeability.

Correctness. *A digital signature scheme* Σ *satisfies* correctness *if for every* $\lambda \in \mathbb{N}$*, and message* $m \in \{0,1\}^*$*, we have that:*

$$\Pr\left[\mathsf{Vrfy}(\mathsf{vk}, m, \sigma) = 1 \middle| \begin{array}{c} (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(1^\lambda), \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{array}\right] = 1 \ .$$

Strong Unforgeability. *A digital signature scheme* Σ *satisfies* strong unforgeability *if for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* ϵ *such that for all* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\mathsf{strongForge}_{\mathcal{A},\Sigma}(1^\lambda) = 1\right] \leq \epsilon(\lambda) \ ,$$

where $\mathsf{strongForge}_{\mathcal{A},\Sigma}$ *is defined in Figure 2.*

$\mathsf{strongForge}_{\mathcal{A},\Sigma}(1^\lambda)$

1: $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KGen}(1^\lambda), Q := \emptyset$
2: $(m^*, \sigma^*) \leftarrow \mathcal{A}^{O_s}(\mathsf{vk})$
3: **return** $((m^*, \sigma^*) \notin Q) \wedge \mathsf{Vrfy}(\mathsf{vk}, m^*, \sigma^*) = 1$

Oracle $O_s(m)$

1: $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$
2: $Q := Q \cup \{(m, \sigma)\}$
3: **return** σ

Figure 2: Strong Unforgeability for Digital Signature Schemes

Game $G_{real}(1^\lambda)$

1: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2: $(\mathsf{stmt}, \mathsf{wit}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}, z)$
3: **if** $((\mathsf{stmt}, \mathsf{wit}) \notin \mathsf{R})$: **return** $\perp$
4: **return** $\mathsf{view}_{\mathcal{A}_2}(\langle \mathcal{P}(\mathsf{pp}, \mathsf{stmt}, \mathsf{wit}), \mathcal{A}_2(\mathsf{stmt}, \mathsf{st})\rangle)$

Game $G_{ideal}(1^\lambda)$

1: $(\mathsf{pp}, \sigma) \leftarrow \mathcal{S}(1^\lambda)$
2: $(\mathsf{stmt}, \mathsf{wit}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}, z)$
3: **if** $((\mathsf{stmt}, \mathsf{wit}) \notin \mathsf{R})$: **return** $\perp$
4: **return** $\mathsf{view}_{\mathcal{A}_2}(\mathcal{S}^{\mathcal{A}_2}(\mathsf{pp}, \mathsf{stmt}, z, \sigma))$

Figure 3: Real and Ideal Games for Interactive Protocol

2.2 IOP Syntax, Security, Efficiency

Definition 2.7 (Interactive Argument of Knowledge). *An interactive protocol* $\prod = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *between a prover* $\mathcal{P}$ *and a verifier* $\mathcal{V}$ *is an argument of knowledge for a relation* R *if the following properties hold, where the common input is denoted by* stmt*, the prover's witness is denoted by* wit*, and the output of the verifier is denoted by the random variable* $\langle \mathcal{P}(\mathsf{pp}, \mathsf{stmt}, \mathsf{wit}), \mathcal{V}(\mathsf{pp}, \mathsf{stmt})\rangle$*:*

- *Perfect Completeness: for all* $(\mathsf{stmt}, \mathsf{wit}) \in \mathsf{R}$*, we have that:*

$$\Pr\left[\left\langle \mathcal{P}(\mathsf{pp}, \mathsf{stmt}, \mathsf{wit}), \mathcal{V}(\mathsf{pp}, \mathsf{stmt})\right\rangle = 1\right] = 1 \ ,$$

  where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
- *Knowledge Soundness: There exists an expected* PPT *oracle machine* $\mathcal{E}$ *called the extractor such that given oracle access to any pair of* PPT *adversarial prover algorithm* $(\mathcal{A}_1, \mathcal{A}_2)$ *the following holds:*

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}_2(\mathsf{pp}, \mathsf{stmt}, \mathsf{st}), \mathcal{V}(\mathsf{pp}, \mathsf{stmt})\rangle = 1 \\ \wedge\ (\mathsf{stmt}, \mathsf{wit}) \notin \mathsf{R} \end{array}\right] \leq \epsilon(\lambda) \ , \tag{3}$$

  where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{stmt}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp})$*, and* $\mathsf{wit} \leftarrow \mathcal{E}^{\mathcal{A}_2}(\mathsf{pp}, \mathsf{stmt})$.
- *Public Coin: An interactive protocol is public coin if* **all** *of* $\mathcal{V}$*'s messages (including the final output) can be computed as a deterministic function given a random public input.*
- *Zero knowledge: An interactive protocol* $\prod$ *is zero-knowledge if there is an expected* PPT *simulator* $\mathcal{S}$ *such that for every* PPT *adversary* $A = (\mathcal{A}_1, \mathcal{A}_2)$*, and auxiliary input* $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$*, we have that there exists a negligible function* $\epsilon(\lambda)$ *such that for all* PPT *distinguishers* $\mathcal{D}$ *and all* $\lambda \in \mathbb{N}$*:*

$$\left|\Pr\left[\mathcal{D}(G_{real}(1^\lambda)) = 1\right] - \Pr\left[\mathcal{D}(G_{ideal}(1^\lambda)) = 1\right]\right| \leq \epsilon(\lambda) \ ,$$

  where games G_{real} *and* G_{ideal} *are defined as in Figure 3.*

We say that $\langle \mathcal{P}, \mathcal{V} \rangle$ is honest-verifier zero knowledge (HVZK) if the adversary $\mathcal{A}_2$ honestly follows the verifier algorithm.

Definition 2.8 (Poly Interactive Oracle Proof (PIOP)). *A polynomial interactive oracle proof (PIOP) is a public-coin interactive protocol for a polynomial oracle relation* $\mathsf{R} = \{(\mathsf{stmt}, \mathsf{wit})\}$. *The relation is an oracle relation in that* stmt *can contain oracles to μ-variate polynomials over some field $\mathbb{F}$. The oracles specify μ and the degree in each variable. These oracles can be queried at arbitrary points in $\mathbb{F}^\mu$ to evaluate the polynomial at these points. The actual polynomials corresponding to the oracles are contained in the* pp *and the* wit, *respectively. We denote an oracle to a polynomial f by $[[f]]$. In every protocol message, the $\mathcal{P}$ sends multi-variate polynomial oracles. The verifier in every round sends a random challenge. We measure the following parameters for the complexity of a PIOP. The prover time measures the runtime of the prover. The verifier time measures the runtime of the verifier. The query complexity is the number of queries the verifier performs to the oracles. The round complexity measures the number of rounds. In our protocols, it is equivalent to the number of oracles sent. The communication size is the number of field elements sent by the prover to the verifier. The size of the proof oracles is the length of the transmitted polynomials. The size of the witness is the length of the witness polynomial.*

Looking ahead, HyperVerITAS utilizes two main PIOPs as subroutines: the Lookup PIOP from HyperPlonk [11], and the Inner Product PIOP (which can be proved using the classic sumcheck protocol from [27]). We define the corresponding relations and detail asymptotic efficiency on the prover time, verifier time, and proof size required by our work.

Definition 2.9 (Lookup Relation [11]). *The lookup relation R_{lk} is defined as:*

$$\mathsf{R}_{\mathsf{lk}} = \left\{ ((\mathbf{t}, [[f]]); f) : \forall \mathbf{x} \in \mathbb{B}_\mu, \mathbf{f}(\mathbf{x}) \in \mathbf{t} \right\},$$

where $\mathbf{t} \in \mathbb{F}^{2^\mu - 1}$, and f is a degree d multilinear polynomial with $f \in \mathbb{F}[X_1, \ldots, X_\mu]$.

HyperPlonk [11] built a PIOP for the look-up relation that is knowledge sound with error $O(\frac{2^\mu + d\mu}{|\mathbb{F}|})$. The prover runtime ($\mathsf{TP}_{\mathsf{lk}}$) is $O(d \log^2 d \cdot 2^\mu)\mathbb{F}$. The verifier runtime ($\mathsf{TV}_{\mathsf{lk}}$) is $O(\mu)\mathbb{F}$. The query complexity (Q_{lk}) is $\mu + 3$. The round complexity ($\mathsf{RC}_{\mathsf{lk}}$) is $\mu + 2$. The communication size ($\mathsf{CS}_{\mathsf{lk}}$) is $\mu\mathbb{F}$. The proof oracle size ($\mathsf{OS}_{\mathsf{lk}}$) is $2^{\mu+1} + O(2^\mu)$. The witness size ($\mathsf{WS}_{\mathsf{lk}}$) is $O(2^\mu)$.

Definition 2.10 (Inner Product Relation). *The inner product relation R_{ip} is defined as:*

$$\mathsf{R}_{\mathsf{ip}} = \left\{ (([[f]], g, c); f) : c = \textstyle\sum_{\mathbf{x} \in \mathbb{B}_\mu} f(\mathbf{x}) \cdot \mathbf{g}(\mathbf{x}) \right\},$$

where f, g are multilinear polynomials with $f, g \in \mathbb{F}[X_1, \ldots, X_\mu]$.

This relation can be proved using the classical sum-check protocol [27] for a degree-2 multivariate polynomial (i.e., $f \cdot g$). In our implementation, we use HyperPlonk's version of the sum-check protocol which was proven to be knowledge sound with error $\frac{2\mu}{|\mathbb{F}|}$, and reported the following asymptotics for degree 2 polynomials. The prover runtime ($\mathsf{TP}_{\mathsf{sc}}$) is $O(2^{\mu+1})\mathbb{F}$. The verifier runtime ($\mathsf{TV}_{\mathsf{sc}}$) is $O(\mu)\mathbb{F}$. The query complexity (Q_{sc}) is $\mu + 1$. The round complexity ($\mathsf{RC}_{\mathsf{sc}}$) is μ. The communication size ($\mathsf{CS}_{\mathsf{sc}}$) is $\mu\mathbb{F}$. The proof oracle size ($\mathsf{OS}_{\mathsf{sc}}$) is 2μ. The witness size ($\mathsf{WS}_{\mathsf{sc}}$) is $O(2^\mu)$.

Game $G_b(1^\lambda)$

1: $\mathsf{pp} \leftarrow \mathsf{C.Setup}(1^\lambda)$
2: $m_0, m_1 \leftarrow \mathcal{A}(\mathsf{pp})$
3: $(\mathsf{com}, d) \leftarrow \mathsf{C.Commit}(\mathsf{pp}, m_b)$
4: **return** $b' \leftarrow \mathcal{A}(\mathsf{com}, d)$

Figure 4: Hiding Game G_b for Commitment Schemes

2.3 Polynomial Commitments

Before we define polynomial commitment schemes, we first define the general notion of a commitment scheme.

Definition 2.11 (Commitment Scheme). *A commitment scheme* $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *is tuple of three algorithms* Setup, Commit, *and* Open, *defined as follows:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter 1^λ, and outputs the public parameters* pp.
- $(\mathsf{com}; d) \leftarrow \mathsf{Commit}(\mathsf{pp}, m)$*: The commit algorithm takes as input the public parameters* pp *and a message m, and outputs a commitment* com *and an decommitment string d.*
- $b \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, m, d)$*: The opening algorithm takes as input the public parameters* pp*, a commitment* com*, a message m, and a decommitment string d, and outputs a bit $b \in \{0, 1\}$.*

A commitment scheme is secure if it satisfies hiding and binding.

Hiding. *A commitment scheme is perfectly hiding if for all* PPT *adversaries $\mathcal{A}$, we have that:*

$$\left| \Pr[G_0 \text{ outs } 1] - \Pr[G_1 \text{ outs } 1] \right| = 0 \,,$$

where games G_0 and G_1 are defined as in Figure 4.

Binding. *A commitment scheme is computationally binding if for all* PPT *adversaries $\mathcal{A}$, there exists a negligible function ϵ such that for all $\lambda \in \mathbb{N}$, we have that:*

$$\Pr\begin{bmatrix} m_1 \neq m_2 \\ \wedge\ \mathsf{Open}(\mathsf{pp}, \mathsf{com}, m_1, d_1) = 1 \\ \wedge\ \mathsf{Open}(\mathsf{pp}, \mathsf{com}, m_2, d_2) = 1 \end{bmatrix} \leq \epsilon(\lambda) \,,$$

where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, and $(m_1, m_2, \mathsf{com}, d_1, d_2) \leftarrow \mathcal{A}(\mathsf{pp})$.

Now, we define polynomial commitment schemes.

Definition 2.12 (Polynomial Commitment Scheme). *A polynomial commitment scheme* $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ *allows one to commit to a multilinear polynomial $f \in \mathbb{F}[\mathbf{X}]$ (with n variables) and provide evaluation proofs for the polynomial at a point $\mathbf{x} \in \mathbb{F}^n$.* PC *is a tuple of four algorithms* Setup, Commit, Open, Verify *defined as follows:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$*: The setup algorithm takes as input the security parameter 1^λ and the number of variables n, and outputs the public parameters* pp.
- $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, f)$*: The commit algorithm takes as input the public parameters* pp *and a n-variable multilinear polynomial f, and outputs a commitment* com.
- $b \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f)$*: The opening algorithm takes as input the public parameters* pp*, a commitment* com*, and a n-variable multilinear polynomial f, and outputs a bit $b \in \{0, 1\}$.*
- $\langle \mathcal{P}, \mathcal{V} \rangle$ *is an interactive protocol between the prover $\mathcal{P}$ and verifier $\mathcal{V}$ that proves the relation* R_{ev}*:*

$$\mathsf{R}_{\mathsf{ev}} = \left\{ ((\mathsf{pp}, \mathsf{com}, \mathbf{x}, y); f) : \begin{array}{c} \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f) = 1 \\ \wedge\, f(\mathbf{x}) = y \end{array} \right\}$$

A polynomial commitment scheme is secure if it satisfies the property of extractability.

Extractability. *A polynomial commitment scheme satisfies extractability if the interactive protocol $\langle \mathcal{P}, \mathcal{V} \rangle$ is ϵ-knowledge sound (as in Theorem 2.7).*

Multilinear PCS Instantiations. In this work, we use the Brakedown [17] and PST [31] multilinear polynomial commitment schemes. However, note that HyperVerITAS is a flexible scheme, and can be instantiated with any other multilinear PCS, while still retaining its efficiency.

For PST[31], we have the following asymptotics for an opening proof, where the size of the committed polynomial is n, $\mathbb{G}$ denotes group scalar multiplications (and group elements for proof size), and $\mathbb{P}$ denotes pairings: prover time ($\mathsf{TP}^{\mathsf{PST}}_{\mathsf{op}}(n)$) is $O(n)\mathbb{G}$, verifier time ($\mathsf{TV}^{\mathsf{PST}}_{\mathsf{op}}(n)$) is $O(\log n)\mathbb{P}$, and proof size ($\pi^{\mathsf{PST}}_{\mathsf{op}}(n)$) is $O(\log n)\mathbb{G}$.

For Brakedown[17], we have the following asymptotics for an opening proof, where the size of the committed polynomial is n, and $\mathbb{F}$ denotes field operations (for proof size $\mathbb{F}$ represents field elements): prover time ($\mathsf{TP}^{\mathsf{BD}}_{\mathsf{op}}(n)$) is $O(n)\mathbb{F}$, verifier time ($\mathsf{TV}^{\mathsf{BD}}_{\mathsf{op}}(n)$) is $O(\sqrt{n})\mathbb{F}$, and proof size ($\pi^{\mathsf{BD}}_{\mathsf{op}}(n)$) is $O(\sqrt{n})\mathbb{F}$.

3 The Design of HyperVerITAS

We now formalize the relations and protocols for HyperVerITAS.

3.1 Overview

In this section, we provide the overall relation for HyperVerITAS. We describe it as an "overall relation" because rather than proving the relation in one shot, we prove components separately. We describe our overall relation for HyperVerITAS in Definition 3.1.

Definition 3.1 (Overall Relation for HyperVerITAS). *The overall relation $\mathsf{R}(\mathsf{pp}, \mathsf{vk}, \mathbf{A})$, where the relation is parameterized by the public parameters pp, the verification key vk of the digital signature scheme Σ, and a hash function A, is defined as:*

$$\mathsf{R} = \left\{ ((\mathbf{I}_{\mathbf{t}}, \mathsf{com}_{\mathbf{I}}, \mathbf{H}, \sigma, \mathsf{T}); \mathbf{I}) : \begin{array}{r} \mathsf{com}_{\mathbf{I}} = \mathsf{Commit}(\mathsf{pp}, \mathbf{I}) \\ \wedge\, \mathbf{I}_{\mathbf{t}} = \mathsf{T}(\mathbf{I}) \wedge\ \mathbf{H} = \mathbf{A} \odot \mathbf{I} \wedge \\ \mathbf{I} \in [256]^{n \times 3} \wedge \mathsf{Vrfy}(\mathsf{vk}, \mathbf{H}, \sigma) = 1 \end{array} \right\}$$

Note that the overall relation is very modular, and is broken down into four main components. The first condition ($\mathsf{com}_{\mathbf{I}} = \mathsf{Commit}(\mathbf{I})$) ensures that the commitment is correctly generated. The second condition ($\mathbf{I}_{\mathbf{t}} = \mathsf{T}(\mathbf{I})$) verifies that we have a valid transformed image. The next two conditions ($\mathbf{H} = \mathbf{A} \odot \mathbf{I} \wedge \mathbf{I} \in [256]^{n \times 3}$) check that the prover knows a low-norm pre-image of the hash. The last condition ($\mathsf{Vrfy}(\mathsf{vk}, m, \mathbf{H}, \sigma) = 1$) simply ensures that the signature is valid with respect to the message, verification key, and hash.

On Our Image Representation. Note that when we commit to an image, we commit to each channel separately. This choice was driven by prover efficiency, as committing to three smaller vectors is more efficient, and the time for the three commitments can be amortized by parallelization. As noted, some parallelization is indeed possible in the commitment and opening phases, but the benefit depends heavily on the underlying polynomial commitment scheme. For example, even in linear-time schemes like Brakedown[17], committing to a vector of size 2^{25} (33 MP) is nearly 5x faster than committing to a vector of size 2^{27} (which would represent a flattened 3-channel image of 2^{25} pixels, rounded to a power of 2). Although this approach increases proof sizes by a few group/field elements, depending on the underlying PCS, it decreases prover time by reducing expensive commitment computations.

Also, due to modular nature of the relation, we are able to prove different parts of the relation separately. In fact, we have tailored different proof systems for each condition of this relation. We describe the general proof system for the correctness of the transformed image in Section 3.2, and provide in text the specific instantiations for the redacting and grayscaling protocols. We describe the proof system for knowledge of a low-norm hash pre-image in Section 3.3. In the case where the hash is public, we can simply use the verification algorithm of the signature scheme Σ to perform the signature verification check.

3.2 Proving Correctness of Affine Transformations

Most image transformations can be viewed as affine transformations, and this is the case for those we consider in this paper (cropping, grayscaling, blurring, resizing). Rather than writing a separate relation definition for each transformation, we instead create a general relation for the correctness of an affine transformation.

We model an image transformation T as $\mathsf{T}(\mathbf{I}) = \mathbf{L} \odot \mathbf{I} \odot \mathbf{R} + \mathbf{E}$, for some $B \in \mathbb{Z}^{\geq 0}$, $\mathbf{L} \in \mathbb{Z}^{m \times n}$, $\mathbf{R} \in \mathbb{Z}^{3 \times 3}$, and $\mathbf{E} \in \mathbb{Z}^{m \times 3}$ such that $||\mathbf{E}||_\infty \leq B$. We note that in this case, we think of $\mathbf{L}$ as a transformation on the the channels individually (as in blurring, resizing), while $\mathbf{R}$ acts as transformation on the image that acts linearly on the interaction on the channels of images (as in grayscaling). Some transformations, such as redaction, utilize the constant term $\mathbf{E}$ as part of their definition, while other transformations, such as grayscaling or resizing utilize $\mathbf{E}$ as an error term.

Our protocol handles a specific set of affine transformations: those of the form $\mathsf{T}(\mathbf{I}) = \mathbf{L} \odot \mathbf{I} \odot \mathbf{R} + \mathbf{E}$, where $\mathbf{I}$ is modeled as an $n \times 3$ matrix (i.e., one column per channel). This includes a wide range of natural transformations in the C2PA standard including cropping, resizing, redacting, and grayscaling.

Remark 3.2. *Note that if one flattens the image matrix $\mathbf{I} \in \mathbb{Z}^{n \times 3}$ into a vector $\mathbf{i} \in \mathbb{Z}^{3n}$, a more general class of affine transformations becomes definable. That is, $\mathsf{T}(\mathbf{i}) = M \odot \mathbf{i}$, where $M \in \mathbb{Z}^{3m \times 3n}$ is a matrix which defines an affine transformation from an $3n$-sized input to an m-sized output. While our current formalism does not capture the arbitrary affine maps in this broader space (e.g., when the transformation matrix M is not a Kronecker product), our framework is extensible: with minor changes, we could commit to the image as a single $3n$-vector to support these cases. However, given that most real-world transformations fall within our current formalism, we chose this representation for its efficiency (see explaination in Section 3.1).*

With this representation in mind, we now introduce our general relation for affine transformations.

Definition 3.3 (Public Affine Relation). *The public affine relation $\mathsf{R}_{\mathsf{par}}$ is defined as:*

$$\mathsf{R}_{\mathsf{par}} = \left\{ ((\mathbf{I}_{\mathbf{t}}, \mathsf{com}_{\mathbf{I}}, \mathbf{L}, \mathbf{R}, \mathbf{E}); \mathbf{I}) : \begin{array}{r} \mathsf{com}_{\mathbf{I}} = \mathsf{Commit}(\mathsf{pp}, \mathbf{I}) \\ \wedge\, \mathbf{I}_{\mathbf{t}} = \mathbf{L} \odot \mathbf{I} \odot \mathbf{R} + \mathbf{E} \end{array} \right\}.$$

Public Affine Protocol $\left((\mathbf{I_t}, \mathsf{com_I}, \mathbf{L}, \mathbf{R}, \mathbf{E}); \mathbf{I}\right)$

1: $\mathcal{P}$ calculates commitments to each column of $\mathbf{I} \odot \mathbf{R}$, denote these as $\mathsf{com}_{(\mathbf{I}\odot\mathbf{R})_j}$ for $j \in [3]$.
2: $\mathcal{V}$ samples $\mathbf{r} \leftarrow\!\!\$\ \mathbb{F}^m$, and sends $\mathbf{r}$ to $\mathcal{P}$.
3: $\mathcal{P}$ and $\mathcal{V}$ compute $c_j := \langle \mathbf{r}, (\mathbf{I_t} - \mathbf{E})_j \rangle$ for $j \in [3]$.
4: $\mathcal{P}$ and $\mathcal{V}$ compute $\mathbf{v_0} := \mathbf{r} \odot \mathbf{L}$.
5: $\mathcal{P}$ and $\mathcal{V}$ engage in an Inner Product Protocol for relation $\left((\mathbf{v_0}, \mathsf{com}_{(\mathbf{I}\odot\mathbf{R})_j}, c_j); (\mathbf{I} \odot \mathbf{R})_j\right) \in \mathsf{R}_{\mathsf{ipE}}$ for $j \in [3]$.
6: $\mathcal{V}$ accepts if they accept the three inner product arguments.

Figure 5: Protocol for the Public Affine Relation

The relation $\mathsf{R}_{\mathsf{par}}$ can handle any image transformation that can be modeled as an affine transformation. Our protocol for proving an instance of the relation $\mathsf{R}_{\mathsf{par}}$ is depicted in Figure 5. In this protocol, the prover and verifier engage in inner product protocols to ensure correctness of the transformation, one channel at a time.

Theorem 3.4. *Suppose that the Inner Product Protocol is an Interactive Argument of Knowledge (Theorem 2.7). Then the Public Affine Protocol in Figure 5 is an Interactive Argument of Knowledge.*

Proof Sketch. Knowledge Soundness follows from the knowledge soundness of the the inner product protocol and Freivald's algorithm. Zero-knowledge follows from the zero-knowledge of the inner product protocol and the hiding property of the commitments. □

Complexity of Public Affine Protocol. We discuss the complexity of our public affine protocol below.

Theorem 3.5 (Complexity of Public Affine Protocol). *Let* PC *denote a polynomial commitment scheme. Let* $\mathsf{TP}^{\mathsf{PC}}_{\mathsf{op}}$, $\mathsf{TV}^{\mathsf{PC}}_{\mathsf{op}}$, *and* $\pi^{\mathsf{PC}}_{\mathsf{op}}$ *denote the prover runtime, verifier runtime, and proof size of an opening proof for the commitment scheme used. Let* $\pi^{\mathsf{PC}}_{\mathsf{com}}$ *denote the size of a commitment. Here, n is the size of the original image, and m is the size of the transformed image. We use* $\mathbb{F}$ *to denote field operations and* $\mathbb{H}$ *to denote hash computations. Then the asymptotic complexity of the Public Affine Protocol in Figure 5 is as follows:*

- $\mathsf{TP}^{\mathsf{PC}}_{\mathsf{par}}(n, m) := nm + O(n)\mathbb{F} + O(m)\mathbb{H} + 3\mathsf{TP}^{\mathsf{PC}}_{\mathsf{op}}(n)$
- $\mathsf{TV}^{\mathsf{PC}}_{\mathsf{par}}(n, m) := nm + O(m + \log n)\mathbb{F} + O(m)\mathbb{H} + 3\mathsf{TV}^{\mathsf{PC}}_{\mathsf{op}}(n)$
- $\pi^{\mathsf{PC}}_{\mathsf{par}}(n, m) := 3\log n\mathbb{F} + 3\pi^{\mathsf{PC}}_{\mathsf{com}} + 3\pi^{\mathsf{PC}}_{\mathsf{op}}$

Note that here we open each of the 3 *commitments on a point, so the opening complexity and proof size can be amortized by batching.*

Remark 3.6. *Technically speaking, the term* $O(n)$ *in our complexity analysis should be dominated by* $O(nm)$. *Despite this, we choose to be explicit about the* $O(n)$ *term as it helps us in asymptotic analysis of our gray-scaling and cropping protocol where the term nm decomposes to just* $O(n)$ *for sparse matrix-vector multiplications that arise in gray-scaling and cropping.*

Example: Redacting. We provide the concrete instantiation for one of the public image transformations, redacting. More specifically, we provide closed forms for the matrices $\mathbf{L}, \mathbf{R}, \mathbf{E}$ as in the relations $\mathsf{R}_{\mathsf{par}}$. These matrices describe the affine image transformations.

In the case of redacting, we want to remove some pixels, and replace them with some color $k = (k_1, k_2, k_3) \in [256]^3$. Let $S = \{i \in [n] | \text{pixel } i \text{ is redacted}\}$ be the set of redacted pixels. We view $\mathbf{L}$ as a diagonal matrix, that is, a matrix whose entries outside the diagonal are all zero. Along the diagonal, we set $\mathbf{L}_{i,i} := 0$ if $i \in S$, and $\mathbf{L}_{i,i} := 1$ otherwise. Notice that if we set $L_{i,i} = 0$, when we multiply $\mathbf{L} \odot \mathbf{I}$, we have that $(\mathbf{L} \odot \mathbf{I})_{i,0} = (\mathbf{L} \odot \mathbf{I})_{i,1} = (\mathbf{L} \odot \mathbf{I})_{i,2} = 0$ (that is, the pixel i is set to 0). To simplify notation, we view the diagonal as a vector $\mathbf{d} = (d_0, \ldots, d_{n-1})$, such that $d_i = 0$ if $i \in S$, and $d_i = 1$ otherwise. Let $\mathbf{e} = (e_0, \ldots, e_{n-1})$ such that $e_i = 1$ if $i \in S$, and $e_i = 0$ otherwise. We can view $\mathbf{e}$ as the vector adding the color over the redacted pixels (once multiplied by the desired color). Note that our error matrix has size $n \times 3$ since the transformed image has the same dimensions as the original image. With this in mind, we can then view the affine transformation for redacting as follows:

$$\mathbf{L} = \mathsf{diag}(\mathbf{d}), \mathbf{R} = I_{3\times 3}, \mathbf{E} = \begin{bmatrix} k_1 \cdot \mathbf{e}_0 & k_2 \cdot \mathbf{e}_0 & k_3 \cdot \mathbf{e}_0 \\ \vdots & \vdots & \vdots \\ k_1 \cdot \mathbf{e}_{n-1} & k_2 \cdot \mathbf{e}_{n-1} & k_3 \cdot \mathbf{e}_{n-1} \end{bmatrix}.$$

Although it is typically fine to leave the term $\mathbf{E}$ public, there some cases where revealing $\mathbf{E}$ leaks information about the original image. We now define the relation $\mathsf{R}_{\mathsf{sar}}$ for the secret version, where the term $\mathbf{E}$ is kept private.

Definition 3.7 (Secret Affine Relation). *The secret affine relation* $\mathsf{R}_{\mathsf{sar}}$ *is defined as:*

$$\mathsf{R}_{\mathsf{sar}} = \left\{ \left((\mathbf{I_t}, \mathsf{com_I}, \mathsf{com_E}, B, \mathbf{L}, \mathbf{R}); \mathbf{I}, \mathbf{E}\right) : \begin{array}{c} \mathsf{com_I} = \mathsf{Commit}(\mathsf{pp}, \mathbf{I}) \wedge \\ \mathsf{com_E} = \mathsf{Commit}(\mathsf{pp}, \mathbf{E}) \\ \wedge\ ||\mathbf{E}||_\infty \le B \\ \wedge\ \mathbf{I_t} = \mathbf{L} \odot \mathbf{I} \odot \mathbf{R} + \mathbf{E} \end{array} \right\}$$

Note that this relation is an extension of the public variant. The *secret* variant includes a commitment to $\mathbf{E}$ and a bound B in the statement, and has two new conditions. The protocol for the secret variant is the similar the public variant, but it needs to include an additional range check to ensure that $||\mathbf{E}||_\infty \le B$, that is, $\mathbf{E}$ has values within the range $[-B, B]$. In addition, the image transformation check is done using two inner product protocols (rather than one as done in the public protocol). We provide the protocol for proving an instance of the relation $\mathsf{R}_{\mathsf{sar}}$ in Figure 8 in Section B, along with its security proof and complexity analysis.

Example: Grayscaling. We provide the concrete instantiation for one of the private image transformations, grayscaling. More specifically, we provide closed forms for the matrices $\mathbf{L}, \mathbf{R}, \mathbf{E}$ as in the relations $\mathsf{R}_{\mathsf{sar}}$. These matrices describe the affine image transformations. The standard grayscale algorithm is as follows:

$$\mathbf{y} := \mathsf{Round}(0.30 \cdot \mathbf{r} + 0.59 \cdot \mathbf{g} + 0.11 \cdot \mathbf{b}),$$

where $\mathbf{r}, \mathbf{g}, \mathbf{b}$ are the vectors that denote the red, green, and blue channels of the image $\mathbf{I}$, respectively. This computed value, $\mathbf{y}$, will be used for all three channels in the grayscaled image. The necessity of the rounding function is due to the fact that the equation has non-integer coefficients, so we may end up with non-integer values for the grayscale channels. Thus, we round to the nearest integers as pixels can only have integer values. Performing this rounding introduces an error term relative to the linear equation; we define

Linear Hash Protocol $\left((\mathsf{com}_{\mathbf{I}}, \mathbf{A}, \mathbf{H}); \mathbf{I}\right)$

1: $\mathcal{P}$ and $\mathcal{V}$ engage in a Public Affine Transformation Protocol for relation $\left((\mathbf{H}, \mathsf{com}_{\mathbf{I}}, \mathbf{A}, I_{3\times 3}, [0]^{h\times 3}); \mathbf{I}\right) \in \mathsf{R}_{\mathsf{par}}$.

2: $\mathcal{P}$ and $\mathcal{V}$ engage in a Lookup Protocol for relation $\left([256], \mathsf{com}_{(\mathbf{I})_j}); (\mathbf{I})_j\right) \in \mathsf{R}_{\mathsf{rc}}$.

3: $\mathcal{V}$ accepts if they accept the public affine transformation argument and the three lookup arguments.

Figure 6: Linear Hash Protocol

that error as $\mathbf{e} := \mathbf{y} - (0.30\mathbf{r} + 0.59\mathbf{g} + 0.11\mathbf{b})$. Note that for every $i \in [n]$, we have that $-0.5 \leq \mathbf{e}_i < 0.5$ and $\mathbf{e}_i \cdot 100 \in \mathbb{Z}$, thus $100 \cdot \mathbf{y} = (30 \cdot \mathbf{r} + 59 \cdot \mathbf{g} + 11 \cdot \mathbf{b}) + 100 \cdot \mathbf{e}$. With this in mind, we can then view the affine transformation for grayscaling as follows:

$$\mathbf{L} = I_{m\times m}, \mathbf{R} = \begin{bmatrix} 30 & 30 & 30 \\ 59 & 59 & 59 \\ 11 & 11 & 11 \end{bmatrix}, \mathbf{E} = 100 \cdot \begin{bmatrix} \mathbf{e}_0 & \mathbf{e}_0 & \mathbf{e}_0 \\ \vdots & \vdots & \vdots \\ \mathbf{e}_{n-1} & \mathbf{e}_{n-1} & \mathbf{e}_{n-1} \end{bmatrix}.$$

3.3 Proving Knowledge of Low Norm Linear Hash Pre-Images

We next describe the relation and protocol for proving knowledge of low-norm hash pre-image, which is another component of the overall relation (Definition 3.1). We need that the pre-image is low-norm, as the short integer solution (SIS) problem only guarantees that our hash function is collision resistant for low-norm inputs. We now define the linear hash relation in Definition 3.8, which is a variant of the relation Datta et al [12] introduced for their hash pre-image proof.

Definition 3.8 (Linear Hash Relation). *The linear hash relation* R_{lh} *is defined as:*

$$\mathsf{R}_{\mathsf{lh}} = \left\{ \left((\mathsf{com}_{\mathbf{I}}, \mathbf{A}, \mathbf{H}); \mathbf{I}\right) : \begin{array}{c} \mathsf{com}_{\mathbf{I}} = \mathsf{Commit}(\mathsf{pp}, \mathbf{I}) \\ \wedge\, \mathbf{H} = \mathbf{A} \odot \mathbf{I} \wedge\ \mathbf{I} \in [256]^{n\times 3} \end{array} \right\}.$$

The linear hash protocol proposed by Datta et al. [12] utilizes general purpose ZK-SNARKs (Groth16) to prove the relation, which leads to costly proof generation times. Rather than utilizing expensive general purpose ZK-SNARKs, we develop a protocol that uses multilinear polynomial commitments. The protocol for the linear hash relation is depicted in Figure 6. Notice that the hash pre-image check can be modeled as an affine transformation, so we can use the protocol developed for $\mathsf{R}_{\mathsf{par}}$ in Figure 5 for the second check. We push the security proof and complexity analysis to Section C.

Optimizing Hash Pre-Image Prover Time. The hash pre-image check $\mathbf{H} = \mathbf{A} \odot \mathbf{I}$ can be optimized through the choice of $\mathbf{A}$. The natural way to generate $\mathbf{A}$ is to sample random values for each entry. Given that $\mathbf{A} \in \mathbb{F}^{h\times n}$, the computation $\mathbf{A} \odot \mathbf{I}$ will be expensive if $\mathbf{A}$ is filled with non-zero values. Since we will deal with large images, n will be large, meaning that $\mathbf{A}$ is a big matrix. Communicating and loading this full matrix $\mathbf{A}$ will be expensive, and so having a fixed way to generate $\mathbf{A}$ could be more optimal. VerITAS [12] instead samples $\mathbf{A}$ by using a linear congruential generator, which means that the prover and verifier can generate the matrix $\mathbf{A}$ when needed, rather than having to store and communicate the whole matrix.

We could also generate $\mathbf{A}$ as a block matrix, where the blocks along the diagonal have random values, and all other blocks are filled with zeroes. Naturally, this leads to a sparse matrix, meaning that $\mathbf{A} \odot \mathbf{I}$ can be computed more efficiently. However, in this case, security of SIS needs to be investigated.

Optimizing Hash Pre-Image Proof Size. The proof size can also be optimized. Since we have that $\mathbf{H} \in \mathbb{F}^{128\times 3}$, and we need 32 bytes to represent each element, the size of one column of $\mathbf{H}$ (i.e. hash of one channel) will be $128 \cdot 32 = 4096$ bytes. If we leave the hash size at 4096 bytes, it will be quite lengthy. We can shrink this hash size further by composing our hash function with another hash function such as Poseidon or SHA-256. VerITAS [12] uses Poseidon to shrink their hash down from 4096 bytes to just 32 bytes (per channel). However, to enable this solution, VerITAS employs a general purpose ZK-SNARK for the hash pre-image proof, which is expensive. As stated previously, we want to avoid using inefficient general purpose ZK-SNARKs, and instead opt to use multivariate methods for faster proof generation times.

Rather than composing the lattice hash with Poseidon, we could instead compose with SHA-256, and use a specialized system such as Binius [13] for the hash pre-image proof. The polynomial commitment scheme compatible with the Binius proof system is adapted from Brakedown [17], the primary polynomial commitment scheme of choice for HyperVerITAS. This is an interesting direction to explore for future work.

4 Implementation

We implemented HyperVerITAS in Rust. More specifically, we implemented HyperVerITAS with five different multilinear PCS: PST [31], Brakedown [17], Basefold [39], BasefoldFri [39], and ZeromorphFri [23]. We used HyperPlonk's [11] implementation for PST and their IOP protocol instantiations. We utilized their IOP instantiations to implement protocols for multset check, range check, and our affine transformation IOPs. For our PST variation of HyperVerITAS, we instantiate it using the arkworks [3] implementation of the 256-bit scalar field of the BLS12-381 curve.

We used the plonkish_basefold github repository [32] for their implementations of various multilinear PCS: Brakedown [17], Basefold [39], BasefoldFri [39], and ZeromorphFri [23]. Building on the available code, we created protocols for product check, multset check, range check, and our affine transformation IOPs. We used the Blake2s hash function when doing the Fiat-Shamir Transform to make our protocols non-interactive. For all variations of HyperVerITAS, we instantiate them over 127-bit fields to provide secure implementations. For the Brakedown variation, we also provide implementations over 64 and 256-bit fields to offer different levels of security. We chose to use the Goldilocks field and the Mersenne127 field implementations in plonkish_basefold repository for the 64-bit and 127-bit scenarios, respectively. We used halo2's[18] implementation of the scalar field of the Bn256 curve for the 256-bit scenario. For optimization, we implemented a batch opening protocol for the Brakedown PCS in the scenario where a single commitment is opened at multiple points. The protocol is a variant of the general batch opening protocol presented in HyperPlonk [11]. This optimization drastically reduces the proof size for HyperVerITAS when instantiated with Brakedown.

We utilized VerITAS's repository [37], VIMz's repository [38], and TilesProof-MT's repository [33] to perform comparisons. In the case of VerITAS, we implemented full system versions of crop and grayscale for both the KZG and FRI variants. Aligned with the suggestions in VerITAS's paper, to create the full system version, we run the single-channel hash pre-image proofs and image transformations proofs in parallel. With this implementation, we can perform a fair comparison between VerITAS and HyperVerITAS. As for VIMz and TilesProof-MT, we utilized their existing code to create circuits that worked for the images sizes we benchmark with in this work (i.e powers of 2). For TilesProof-MT, we use tiles with a size of 184756 pixels for cropping, and tiles with a size of 80000 pixels for grayscaling, as done in the original paper. Our implementation of HyperVerITAS, as well as our extensions of the other schemes, is available at [19].

4.1 Mode A: Camera

In this mode, recall that the image is hashed using a lattice based linear hash function which is low-norm collision resistant (under the hardness of the SIS problem). For a fair comparison with VerITAS, we use the same SIS parameters in our tests: $h = 128$, $B = 256$ (max pixel value desired is 255), and set q to be the bit-size of the prime field used for the PCS implementation. Note that this means for a hash $\mathbf{h} = \mathbf{A} \odot \mathbf{v}$, we have that $\mathbf{h} \in \mathbb{F}^{128}$, where $\mathbb{F}$ is a q-bit field.

In order to demonstrate the modularity of HyperVerITAS, we provide a comprehensive implementation over five different multilinear PCS: PST, Brakedown, Basefold, BasefoldFri, and ZeromorphFri. Note that HyperVerITAS could be extended to other PCS with minimal integration – only requiring a PCS implementation that links up with the plonkish_basefold repository. We compare these variants of HyperVerITAS to VerITAS instantiated with KZG and FRI. Through this comprehensive comparison, we demonstrate that HyperVerITAS is more efficient that VerITAS on nearly all metrics across a variety of PCS.

Further, we compare HyperVerITAS to schemes that employ folding and tiling techniques: VIMz and TilesProof-MT. Through this comparison, we provide a fair analysis of the various trade-offs that come with using HyperVerITAS. For all systems, we provide full system implementations for two image transformations: crop and grayscale. Note that cropping is a public affine transformation, while grayscale is a private affine transformation. Together, these two image transformations provide an accurate representation of HyperVerITAS's workload for any image transformation.

4.2 Mode B: ML Model

In the ML scenario, recall that rather than using the linear hash, we instead use multilinear commitment schemes to hash the images. Then, the prover can generate a proof that the transformed image was transformed from the image behind the commitment. This results in a significant reduction in overhead for the prover, as they no longer need to generate the expensive hash pre-image proof that they needed to in Mode A.

For HyperVerITAS, a prover in Mode B would need to generate the hash (i.e multilinear polynomial commitment to the image), and then do either a public or private affine transformation protocol to prove the image transformation. We report the time it takes for HyperVerITAS to generate a hash in Mode B in Section 5.2.3.

5 Evaluation and Comparison

In this section, we present a comprehensive comparison of HyperVerITAS (instantiated with five different PCS) with existing systems such as VerITAS [12], VIMz [14], and TilesProof-MT [28]. We present an asymptotic efficiency comparison in Section 5.1 and concrete experimental comparisons in Section 5.2.

5.1 Asymptotic Comparison

The asymptotics for prover runtime, verifier runtime, and proof size are included in Table 2. Notice that HyperVerITAS's prover is more efficient than VIMz and TilesProof-MT, as when instantiated with schemes like Brakedown, it avoids expensive group operations. VIMz does $O(n)\mathbb{G}$, and TilesProof-MT performs $O(n)\mathbb{E}$. On the other hand, HyperVerITAS performs primarily field operations, particularly when instantiated with schemes such as Brakedown. Notice that VerITAS performs $O(n \log n)\mathbb{F}$, while HyperVerITAS manages with only $O(n)\mathbb{F}$.

Notice that VIMz and TilesProof-MT have the smallest proof size. Since c_n is the number of columns in the original image, note that $\log(c_n) < \log(n)$, so the proof size of VIMz is very small. Similarly, TilesProof-MT incurs a constant number of group elements for each tile, and also provides $\log k$ hashes for each tile. TilesProof-MT opts to use tiles of size 184756 for cropping, so this results in roughly $k \approx 180$ tiles for a 33MP image, so the proof sizes are relatively small (compared to HyperVerITAS). Note that although VerITAS performs more openings than HyperVerITAS, the schemes utilized in VerITAS enable more efficient batching than our optimal scheme, Brakedown, admits. This results in HyperVerITAS having larger proof sizes when instantiated with schemes such as Brakedown. A similar analysis can be applied for verifier runtime, as similar trends hold there as well.

5.2 Experimental Comparison

We used two different environments to run tests. The first is a laptop with 36GB RAM and an Apple M3 Chip. The second enviroment is an AWS EC2 *r5d.4xlarge* instance with 128 GiB RAM and 16 vCPUs. We run full system implementations on both environments and benchmark all schemes over four metrics: prover runtime, verifier runtime, prover peak memory use, and proof size. Further, we use a variety of different image sizes to demonstrate HyperVerITAS's ability to handle any images efficiently. We perform experiments for two image transformations: crop and grayscale. Note that while we discuss the results for cropping in this section, reported numbers and analysis for grayscale is pushed to Section E (see Figure 9, Tables 7 to 9).

On the commodity laptop, we compare the full system implementation of HyperVerITAS with those of VerITAS, VIMz, and TilesProof-MT to demonstrate that HyperVerITAS is able to efficiently prove image transformations for large input sizes, even when computationally limited. For VerITAS, we record numbers for the FRI and KZG variants. For HyperVerITAS, we record numbers for the Brakedown (over 127 bit field) and PST variants. The results are detailed in Section 5.2.1.

	Prover Runtime			Verifier Runtime		Proof Size		
	$\mathbb{F}, \mathbb{G}, \mathbb{P}, \mathbb{E}, \mathbb{H}$	$\mathsf{TP}^{\mathsf{PC}}_{\mathsf{com}}(n)$	$\mathsf{TP}^{\mathsf{PC}}_{\mathsf{op}}(n)$	$\mathbb{F}, \mathbb{G}, \mathbb{P}, \mathbb{E}, \mathbb{H}$	$\mathsf{TV}^{\mathsf{PC}}_{\mathsf{op}}(n)$	$\mathbb{F}, \mathbb{G}, \mathbb{H}$	$\pi^{\mathsf{PC}}_{\mathsf{com}}(n)$	$\pi^{\mathsf{PC}}_{\mathsf{op}}(n)$
VIMz	$O(n)$ $\mathbb{G}$, $(n+m)$ $\mathbb{H}$	—	—	$O(1)$ $\mathbb{G}$	—	$O(\log(c_n+c_m))$ $\mathbb{G}$	—	—
TilesProof-MT	$O(n)$ $\mathbb{E}$, $O(k)$ $\mathbb{H}$	—	—	$O(k)$ $\mathbb{P}$, $O(m)$ $\mathbb{E}$, $O(k \log k)$ $\mathbb{H}$	—	$O(k)$ $\mathbb{G}$, $O(k \log(k))$ $\mathbb{H}$	—	—
VerITAS	$O(n \log n)$ $\mathbb{F}$, $O(m)$ $\mathbb{E}$	13	65	n $\mathbb{F}$, $O(1)$ $\mathbb{P}$, $O(1)$ $\mathbb{E}$	65	$O(1)$ $\mathbb{F}$, $O(1)$ $\mathbb{G}$	13	65
This Work (crop)	$O(n)$ $\mathbb{F}$, $O(m)$ $\mathbb{H}$	12	42	$O(m)$ $\mathbb{F}$, $O(m)$ $\mathbb{H}$	42	$9 \log n$ $\mathbb{F}$	12	42
This Work (gray)	$O(n)$ $\mathbb{F}$, $O(n)$ $\mathbb{H}$	16	55	$O(n)$ $\mathbb{F}$, $O(n)$ $\mathbb{H}$	55	$11 \log n$ $\mathbb{F}$	16	55

Table 2: Asymptotic comparison (dominant operations) of proof systems. $\mathbb{F}$=field operations, $\mathbb{G}$=group scalar multiplications, $\mathbb{P}$=pairing operations, $\mathbb{E}$=group exponentiations, and $\mathbb{H}$=hash computations. PC **denotes a polynomial commitment scheme,** com **denotes commit complexities, and** op **denotes opening complexities. n = size of original image, m = size of transformed image. c_n/c_m = number of columns in the original/transformed image. k = number of tiles used. Note that the opening complexity can be amortized via batching (depending on** PC**).**

We also provide a more comprehensive comparison between HyperVerITAS and VerITAS on the laptop. Specifically, we extend our benchmarks for HyperVerITAS to include five multiliner PCS: PST [31], Brakedown(over 64, 127, and 256 bit fields) [17], Basefold [39], BasefoldFri [39], and ZeromorphFri [23]. Through this comparison, we show that HyperVerITAS is more efficient than VerITAS over a variety of different PCS. The results are detailed in Section 5.2.1. In order to demonstrate that HyperVerITAS remains more efficient than VerITAS when more computational resources are available, we replicate this comparison on a high-memory AWS server. The results are detailed in Section 5.2.2.

Lastly, to demonstrate HyperVerITAS's efficiency in Mode B (i.e the ML scenario), we also provide benchmarks for the runtime and memory usage of hashing via multilinear polynomial commitments. We provide results for all the PCS that we instantiate HyperVerITAS with. The results are detailed in Section 5.2.3.

5.2.1 Full System Comparison on Commodity Laptop. We first provide a comprehensive comparison between HyperVerITAS, VerITAS, VIMz, and TilesProof-MT to examine the various trade-offs between the schemes. We considered four metrics: prover time (T_p), verifier time (T_v), prover peak memory (M_p), and proof size (π). Table 3 contains results for the full proof system, where we crop 50% of the original image. Notice that HyperVerITAS Brakedown has the fastest prover time for all input sizes. For images of size 2^{25} (∼33MP) images, HyperVerITAS Brakedown's prover is over 16x faster than VIMz and 4.5x faster than TilesProof-MT. Note that HyperVerITAS Brakedown is able to generate a proof for 33MP images on a laptop, while VerITAS is only able to handle images up to size 2^{22} (∼4MP). For the largest input size VerITAS can handle, HyperVerITAS Brakedown is ∼10x faster than VerITAS FRI, and over 20x faster than VerITAS KZG.

We provide a more comprehensive comparison between HyperVerITAS and VerITAS in the top graph of Figure 7. Notice that all variants of HyperVerITAS have faster proof generation times than VerITAS FRI and KZG (with the exception of HyperVerITAS ZeromorphFri at 2^{21}). This demonstrates how the design choices of HyperVerITAS drastically reduce proving time and memory usage, as compared to VerITAS. Table 5 (in Section D) contains the corresponding verifier runtime and proof size metrics for the schemes presented in Figure 7. Notice that HyperVerITAS Brakedown's verifier is comparable to VerITAS FRI's for most input sizes, and is over 2x faster at 2^{22}. On the other hand, HyperVerITAS Brakedown has much worse proof sizes than VerITAS FRI. However, HyperVerITAS PST offers much smaller proof sizes than VerITAS, while maintaining faster prover time and lower memory usage.

Although HyperVerITAS outperforms VerITAS in all aspects but proof size, the comparison with VIMz and TilesProof-MT reveals the trade-offs for using HyperVerITAS. Depending on the application, one might prefer VIMz and TilesProof-MT for their minimal memory usage and smaller proof sizes. VIMz and TilesProof-MT achieve the lowest prover peak memory, needing only 9.4GB and 3.1GB RAM for 33MP images, respectively. Further, VIMz generates proofs with constant size of roughly 11KB, and TilesProof-MT generates a proof of size 128KB for 33MP images. On the other hand, HyperVerITAS Brakedown proofs are on the order of MB, even for the smallest image size considered (2^{19}). Further, VIMz's verifier runs in under a second for all sizes, the fastest of all schemes. However, to generate a proof for a 33MP image, VIMz takes about 1 hour and 50 minutes, and TilesProof-MT takes roughly 30 minutes, while HyperVerITAS takes roughly 6.6 minutes.

Note that we have implemented optimizations to help reduce the proof size of HyperVerITAS Brakedown as much as possible. The most significant optimization implemented was the batch opening. The BatchEval protocol from HyperPlonk[11] enables a prover to open multiple distinct commitments at multiple points via a *single* opening proof. However, in order to leverage the full efficiency of this protocol, the commitment scheme used needs to be homomorphic. Since Brakedown is not homomorphic, we were unable to directly use this protocol. Therefore, we implemented a slight variant of this protocol where a prover opens a single commitment at multiple points. The correctness and security of our variant follows from the original protocol. This significantly decreased the number of opening proofs, which drastically reduced the proof size.

Notice that the large proof sizes of HyperVerITAS-Brakedown are not inherent to HyperVerITAS as a scheme, as our proof sizes for HyperVerITAS PST are between 49-55KB. There are two main ways in which we can further optimize HyperVerITAS Brakedown, which we leave for future work. The first direction is to develop a new

T_P (s)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	69.4	139.4	286.3	621.6	—	—	—
V[FRI]	24.6	46.7	98.5	267.8	—	—	—
HV[PST]	16.1	31.9	67.1	143.8	327.8	—	—
HV[BD-127]	3.6	7.1	14.2	27.9	58.2	130.9	402.7
VIMz	150.9	293.9	490.1	971.9	1763.0	3530.5	6729.9
TP-MT	30.2	60.3	120.6	231.2	462.4	914.8	1829.5

M_P (GB)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	9.8	18.0	23.4	28.7	—	—	—
V[FRI]	10.9	19.0	23.9	27.4	—	—	—
HV[PST]	6.4	12.3	23.9	24.5	29.3	—	—
HV[BD-127]	2.0	3.7	6.8	14.2	22.4	26.3	27.4
VIMz	1.6	1.6	2.6	2.7	4.8	5.0	9.4
TP-MT	3.1	3.1	3.1	3.1	3.1	3.1	3.1

T_V (s)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	12.9	25.9	51.1	105.9	—	—	—
V[FRI]	1.5	2.8	5.2	19.0	—	—	—
HV[PST]	3.1	6.1	12.4	24.6	49.5	—	—
HV[BD-127]	1.3	2.5	4.5	8.6	16.4	32.9	62.9
VIMz	0.1	0.1	0.2	0.2	0.3	0.3	0.5
TP-MT	4.9	9.8	19.6	37.5	74.9	148.3	296.7

π (KB)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	461.7	488.7	526.1	544.7	—	—	—
V[FRI]	780.5	821.9	892.9	940.3	—	—	—
HV[PST]	49.1	50.7	52.3	53.9	55.5	—	—
HV[BD-127]	65034	85351	108422	147614	193756	270696	362980
VIMz	10.1	10.1	10.4	10.4	10.7	10.7	11.0
TP-MT	2.1	4.2	8.5	16.3	32.5	64.3	128.7

Table 3: Full System Crop (50% of image) prover runtime (top left), prover peak memory (top right), verifier runtime (bottom left), and proof size (bottom right) for VerITAS, HyperVerITAS, VIMz and TilesProof-MT (TP-MT) on a commodity laptop.

code based multilinear PCS that has better proof sizes. The second direction is to optimize the hash pre-image proof size as discussed in Section 3.3. Although the second direction will result in larger prover time and memory usage, it will decrease the verification time and proof size.

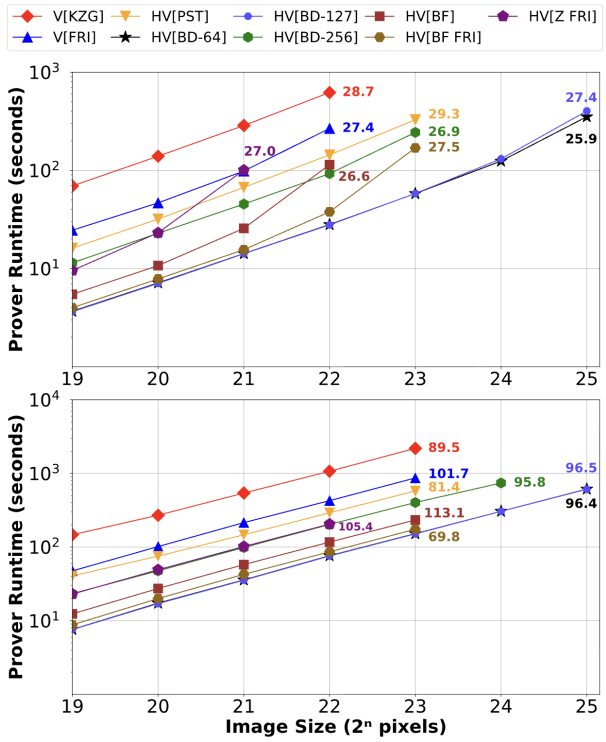

Figure 7: Prover runtime for full system crop (50% of original image) on laptop (top) and AWS (bottom). We display the peak memory of the prover (in GB RAM) for the final data point by writing the value next it. Note the y-axis is log scaled.

5.2.2 Full System Comparison on High-Memory Server. We also compare the performance of HyperVerITAS and VerITAS on a high-memory server similar to the one used in VerITAS's paper to see how the schemes compare when given more resources. Again, we compare HyperVerITAS instantiated with PST, Brakedown (over 64, 127, and 256 bit fields), Basefold, BasefoldFri, and ZeromorphFri to VerITAS instantiated with FRI and KZG. Here, we consider the full system implementation where we crop 50% of the original image. The results for prover runtime and memory are displayed in the bottom graph of Figure 7, while Table 6 (in Section D) contains the corresponding results for verifier runtime and proof size. Notice that all variants of HyperVerITAS have faster proof generation times than VerITAS FRI and KZG on all input sizes. On the server, VerITAS is able to generate proofs for images of size 2^{23}, one power greater than on the laptop. HyperVerITAS Brakedown instantiated over a 256-bit field is even able to generate proofs for larger images than VerITAS FRI (which uses the quadratic extension of a 64-bit field), further showing the efficiency of HyperVerITAS.

5.2.3 Hashing with Polynomial Commitments. Recall that in Mode B, the prover hashes by constructing a multilinear polynomial commitment to the image. In the case of HyperVerITAS, we generate three total commitments - one for each channel (red, green, blue). We have provided prover runtime and peak memory usage for computing these three commitments in Table 4. Notice that for Brakedown, we achieve a PCS hash time of 4.18 seconds and use only 10.19 GB RAM. For the FRI scheme, VerITAS reports a PCS hash time of 19.84 seconds and use 18.90 GB RAM. These results further demonstrate the efficiency of using multilinear PCS rather than univariate ones.

PCS	Commit Time (s)	Peak Memory (GB)
PST	85.58	29.74
Brakedown	4.18	10.19

Table 4: Commitment Scheme Hash Times

Acknowledgments

Garrett Greiner and Pratik Soni were supported in part by a Price College of Engineering and the Vice President for Research (VPR) Office Seed Award and a Google Academic Research Award. The authors used generative AI-based tools to revise the text, improve flow and correct any typos, grammatical errors, and awkward phrasing.

References

[1] M. Ajtai. 1996. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 99–108. https://doi.org/10.1145/237814.237838

[2] M. Ajtai. 1996. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 99–108. https://doi.org/10.1145/237814.237838

[3] arkworks contributors. 2022. `arkworks` zkSNARK ecosystem. https://arkworks.rs

[4] Laasya Bangalore, Rishabh Bhadauria, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. 2022. On Black-Box Constructions of Time and Space Efficient Sublinear Arguments from Symmetric-Key Primitives. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part I* (Chicago, IL, USA). Springer-Verlag, Berlin, Heidelberg, 417–446. https://doi.org/10.1007/978-3-031-22318-1_15

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 14:1–14:17. https://doi.org/10.4230/LIPIcs.ICALP.2018.14

[6] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Scalable Zero Knowledge via Cycles of Elliptic Curves. Cryptology ePrint Archive, Paper 2014/595. https://eprint.iacr.org/2014/595

[7] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. 2020. Public-Coin Zero-Knowledge Arguments with (almost) Minimal Time and Space Overheads. In *Theory of Cryptography*. 168–197.

[8] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. 2021. Time- and Space-Efficient Arguments from Groups of Unknown Order. In *Advances in Cryptology – CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV*. Springer-Verlag, Berlin, Heidelberg, 123–152. https://doi.org/10.1007/978-3-030-84259-8_5

[9] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrú. 2022. Gemini: Elastic SNARKs for Diverse Environments. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part II* (Trondheim, Norway). Springer-Verlag, Berlin, Heidelberg, 427–457. https://doi.org/10.1007/978-3-031-07085-3_15

[10] Jan Camenisch and Markus Stadler. 1997. Efficient group signature schemes for large groups. In *Advances in Cryptology — CRYPTO '97*, Burton S. Kaliski (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 410–424.

[11] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In *Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part II* (Lyon, France). Springer-Verlag, Berlin, Heidelberg, 499–530. https://doi.org/10.1007/978-3-031-30617-4_17

[12] Trisha Datta, Binyi Chen, and Dan Boneh. 2025. VerITAS: Verifying Image Transformations at Scale . In *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 4606–4623. https://doi.org/10.1109/SP61157.2025.00097

[13] Benjamin E. Diamond and Jim Posen. 2025. Succinct Arguments over Towers of Binary Fields. In *Advances in Cryptology – EUROCRYPT 2025*, Serge Fehr and Pierre-Alain Fouque (Eds.). Springer Nature Switzerland, Cham, 93–122.

[14] Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. 2024. VIMz: Private Proofs of Image Manipulation using Folding-based zkSNARKs. Cryptology ePrint Archive, Paper 2024/1063. https://eprint.iacr.org/2024/1063

[15] Coalition for Content Provenance and Authenticity. 2022. *Coalition for Content Provenance and Authenticity (C2PA)*.

[16] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953. https://eprint.iacr.org/2019/953

[17] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. 2023. Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS. In *Advances in Cryptology – CRYPTO 2023*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer Nature Switzerland, Cham, 193–226.

[18] halo2. 2025. https://github.com/zcash/halo2

[19] HyperVerITAS. 2025. https://github.com/glgreiner/HyperVerITAS

[20] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The Elliptic Curve Digital Signature Algorithm (ECDSA). *Int. J. Inf. Secur.* 1, 1 (Aug. 2001), 36–63. https://doi.org/10.1007/s102070100002

[21] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. 2022. ZK-IMG: Attested Images via Zero-Knowledge Proofs to Fight Disinformation. arXiv:2211.04775 [cs.CR] https://arxiv.org/abs/2211.04775

[22] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, Masayuki Abe (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 177–194.

[23] Tohru Kohrita and Patrick Towa. 2024. Zeromorph: Zero-Knowledge Multilinear-Evaluation Proofs from Homomorphic Univariate Commitments. *Journal of Cryptology* 37, 4 (08 Oct 2024), 38. https://doi.org/10.1007/s00145-024-09519-0

[24] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*. 839–858. https://doi.org/10.1109/SP.2016.55

[25] Leica. 2025. *Partnership for greater trust in digital photography: Leica and Content Authenticity Initiative*. https://leica-camera.com/en-US/news/partnership-greater-trust-digital-photography-leica-and-content-authenticity-initiative

[26] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 2968–2985. https://doi.org/10.1145/3460120.3485379

[27] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic methods for interactive proof systems. *J. ACM* 39, 4 (Oct. 1992), 859–868. https://doi.org/10.1145/146585.146605

[28] Pierpaolo Della Monica, Ivan Visconti, Andrea Vitaletti, and Marco Zecchini. 2025. Trust Nobody: Privacy-Preserving Proofs for Edited Photos with Your Laptop. In *2025 IEEE Symposium on Security and Privacy (SP)*. 4624–4642. https://doi.org/10.1109/SP61157.2025.00014

[29] Assa Naveh and Eran Tromer. 2016. PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations. In *2016 IEEE Symposium on Security and Privacy (SP)*. 255–271. https://doi.org/10.1109/SP.2016.23

[30] OpenAI. 2025. *C2PA in ChatGPT Images*. https://help.openai.com/en/articles/8912793-c2pa-in-chatgpt-images

[31] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. 2011. Signatures of Correct Computation. Cryptology ePrint Archive, Paper 2011/587. https://eprint.iacr.org/2011/587

[32] plonkish_basefold. 2025. https://github.com/hadasz/plonkish_basefold

[33] Privacy-PreservingProofs4EditedPhotos. 2025. https://github.com/pierpaolodm/Privacy-PreservingProofs4EditedPhotos

[34] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* 21, 2 (feb 1978), 120–126. https://doi.org/10.1145/359340.359342

[35] C. P. Schnorr. 1991. Efficient Signature Generation by Smart Cards. *J. Cryptol.* 4, 3 (jan 1991), 161–174.

[36] Sony. 2022. *Sony's Forgery-Detection Digital Signature Solutions*. https://pro.sony/ue_US/solutions/forgery-detection

[37] VerITAS. 2025. https://github.com/zk-VerITAS/VerITAS

[38] VIMz. 2025. https://github.com/zero-savvy/vimz/

[39] Hadas Zeilberger, Binyi Chen, and Ben Fisch. 2023. BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes. Cryptology ePrint Archive, Paper 2023/1705. https://eprint.iacr.org/2023/1705

A Supplementary Definitions

A.1 Virtual Oracles and Commitments

Given multiple polynomial oracles, we can construct virtual oracles to the functions of these polynomials. An oracle to $g([f_1], ..., [f_k])$ for some function g is simply the list of oracles $\{[f_1], ..., [f_k]\}$ as well as a description of g. In order to evaluate $g([f_1], ..., [f_k])$ at some point $\mathbf{x}$ we can compute $y_i = f_i(\mathbf{x}) \forall i \in [k]$ and output $g(y_1, ..., y_k)$. Equivalently given commitments to polynomials, we can construct a virtual commitment to a function of these polynomials in the same manner. If g is an additive function and the polynomial commitment is additively homomorphic, then we can use the homomorphism to

Secret Affine Protocol $\left((\mathbf{I_t}, \mathsf{com_I}, \mathsf{com_E}, B, \mathbf{L}, \mathbf{R}); (\mathbf{I}, \mathbf{E})\right)$

1: $\mathcal{P}$ calculates commitments to each column of $\mathbf{I} \odot \mathbf{R}$, denote these as $\mathsf{com}_{(\mathbf{I}\odot\mathbf{R})_j}$ for $j \in [3]$.
2: $\mathcal{P}$ calculates commitments to each column of $(\mathbf{I_t} - \mathbf{E})$, denote these as $\mathsf{com}_{(\mathbf{I_t}-\mathbf{E})_j}$ for $j \in [3]$.
3: $\mathcal{V}$ samples $\mathbf{r} \leftarrow\!\$\ \mathbb{F}^m$ and sends $\mathbf{r}$ to $\mathcal{P}$.
4: $\mathcal{P}$ computes $c_j := \langle \mathbf{r}, (\mathbf{I_t} - \mathbf{E})_j \rangle$, sends c_j to $\mathcal{V}$, for $j \in [3]$.
5: $\mathcal{P}$ and $\mathcal{V}$ engage in an Inner Product Protocol for relation $\left((\mathbf{r}, \mathsf{com}_{(\mathbf{I_t}-\mathbf{E})_j}, c_j); (\mathbf{I_t} - \mathbf{E})_j\right) \in \mathsf{R_{ipE}}$ for $j \in [3]$.
6: $\mathcal{P}$ and $\mathcal{V}$ compute $\mathbf{v_0} := \mathbf{r} \odot \mathbf{L}$.
7: $\mathcal{P}$ and $\mathcal{V}$ engage in an Inner Product Protocol for relation $\left((\mathbf{v_0}, \mathsf{com}_{(\mathbf{I}\odot\mathbf{R})_j}, c_j); (\mathbf{I} \odot \mathbf{R})_j\right) \in \mathsf{R_{ipE}}$ for $j \in [3]$.
8: $\mathcal{P}$ and $\mathcal{V}$ engage in a Lookup Protocol for relation $\left(([B], \mathsf{com_E}); \mathbf{E}\right) \in \mathsf{R_{rc}}$.
9: $\mathcal{V}$ accepts if they accept the six inner product arguments (in steps 5 and 7) and the lookup argument (in step 8).

Figure 8: Protocol for the Secret Affine Relation

do the evaluation. A common example is that given additive commitments C_f, C_g to $f(\mathbf{x}), g(\mathbf{x})$, we want to construct a commitment to $(1-Y)f + Yg$. Then (C_f, C_g) serves as such a commitment and we can evaluate it at $(y, \mathbf{x})$ by evaluating $(1-y)C_f + y \cdot C_g$ at $\mathbf{x}$.

B Secret Affine Protocol

Here, we include the protocol for proving the Secret Affine Relation $\mathsf{R_{sar}}$ as defined in Theorem 3.7. We also include the security proof and the complexity analysis.

Theorem B.1. *Suppose that the Lookup and Inner Product Protocols are Interactive Arguments of Knowledge (Theorem 2.7). Then the Secret Affine Protocol in Figure 8 is an Interactive Argument of Knowledge.*

Proof Sketch. Knowledge Soundness follows from the knowledge soundness of the lookup protocol, the inner product protocol, and Freivald's algorithm. Zero-knowledge comes from the zero-knowledge of the lookup protocol, the inner product protocol, and the hiding property of the commitments. □

Complexity of Secret Affine Protocol. We discuss the complexity of our secret affine protocol below.

Theorem B.2 (Complexity of Secret Affine Protocol). *Let* PC *denote a polynomial commitment scheme. Let* $\mathsf{TP}^{\mathsf{PC}}_{\mathsf{op}}$, $\mathsf{TV}^{\mathsf{PC}}_{\mathsf{op}}$, *and* $\pi^{\mathsf{PC}}_{\mathsf{op}}$ *denote the prover runtime, verifier runtime, and proof size of an opening proof for the commitment scheme used. Let* $\pi^{\mathsf{PC}}_{\mathsf{com}}$ *denote the size of a commitment. Here,* n *is the size of the original image, and* m *is the size of the transformed image. We use* $\mathbb{F}$ *to denote field operations and* $\mathbb{H}$ *to denote hash computations. Then the asymptotic complexity of the Secret Affine Protocol in Figure 8 is as follows:*

- $\mathsf{TP}^{\mathsf{PC}}_{\mathsf{sar}} := nm + O(n)\mathbb{F} + O(m)\mathbb{H} + 3\mathsf{TP}^{\mathsf{PC}}_{\mathsf{op}}(n) + 39\mathsf{TP}^{\mathsf{PC}}_{\mathsf{op}}(m)$
- $\mathsf{TV}^{\mathsf{PC}}_{\mathsf{sar}} := nm + O(m + \log n)\mathbb{F} + O(m)\mathbb{H} + 3\mathsf{TV}^{\mathsf{PC}}_{\mathsf{op}}(n) + 39\mathsf{TV}^{\mathsf{PC}}_{\mathsf{op}}(m)$
- $\pi^{\mathsf{PC}}_{\mathsf{sar}} := (3\log n + 6\log m)\mathbb{F} + 3\pi^{\mathsf{PC}}_{\mathsf{com}}(n) + 12\pi^{\mathsf{PC}}_{\mathsf{com}}(m) + 3\pi^{\mathsf{PC}}_{\mathsf{op}}(n) + 39\pi^{\mathsf{PC}}_{\mathsf{op}}(m)$

Note that here we are opening 15 *commitments on a total of* 42 *points, so the opening complexity and proof size can be amortized by batching. The reduction in complexity varies by PCS (depending on the kind of batching allowed).*

C Linear Hash Protocol

Here, we include the security proof and complexity analysis for the linear hash protocol in Figure 6.

Theorem C.1. *Suppose that the Lookup and Inner Product Protocols are Interactive Arguments of Knowledge (Theorem 2.7). Then the Linear Hash Protocol in Figure 6 is an Interactive Argument of Knowledge.*

Proof Sketch. Knowledge Soundness follows from the knowledge soundness of the lookup protocol, the inner product protocol, and Freivald's algorithm. Zero-knowledge comes from the zero-knowledge of the lookup protocol, the inner product protocol, and the hiding property of the commitments. □

Complexity of Linear Hash Protocol. We describe the efficiency of our linear hash protocol below.

Theorem C.2 (Complexity of Linear Hash Protocol). *Let* PC *denote a polynomial commitment scheme. Let* $\mathsf{TP}^{\mathsf{PC}}_{\mathsf{op}}$, $\mathsf{TV}^{\mathsf{PC}}_{\mathsf{op}}$, *and* $\pi^{\mathsf{PC}}_{\mathsf{op}}$ *denote the prover runtime, verifier runtime, and proof size of an opening proof for the commitment scheme used. Let* $\pi^{\mathsf{PC}}_{\mathsf{com}}$ *denote the size of a commitment. We use* $\mathbb{F}$ *to denote field operations and* $\mathbb{H}$ *to denote hash computations. Then the asymptotic complexity of the Linear Hash Protocol in Figure 6 is as follows:*

- $\mathsf{TP}^{\mathsf{PC}}_{\mathsf{lh}} := \mathsf{TP}^{\mathsf{PC}}_{\mathsf{par}}(n, h) + 3\mathsf{TP}^{\mathsf{PC}}_{\mathsf{lk}}(n)$
- $\mathsf{TV}^{\mathsf{PC}}_{\mathsf{lh}} := \mathsf{TV}^{\mathsf{PC}}_{\mathsf{par}}(n, h) + 3\mathsf{TV}^{\mathsf{PC}}_{\mathsf{lk}}(n)$
- $\pi^{\mathsf{PC}}_{\mathsf{lh}} := \pi^{\mathsf{PC}}_{\mathsf{par}}(n, h) + 3\mathsf{CS}_{\mathsf{lk}}(n) + 9\pi^{\mathsf{PC}}_{\mathsf{com}}(n) + 36\pi^{\mathsf{PC}}_{\mathsf{op}}(n)$

Note that here we are opening 12 *commitments on a total of* 39 *points (we get three additional commitments from public affine relation, and open each one on a point), so the opening complexity and proof size can be amortized by batching. The reduction in complexity varies by PCS (depending on the kind of batching allowed).*

D Crop Experiments

Here, we include the tables for verifier time and proof size for the comparisons between HyperVerITAS and VerITAS on both the laptop (Table 5) and AWS (Table 6).

E Grayscale Experiments

Here, we include the results for the grayscale counterparts to the various experiments included in Section 5.2. More specifically, in Section E.1, we provide the results for the grayscale experiments on a commodity laptop (same set of experiments done for crop in Section 5.2.1). Further, in Section E.2, we provide the results for grayscale experiments on the AWS server (same set of experiments done for crop in Section 5.2.2).

T_V (s)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	12.9	25.9	51.1	105.9	—	—	—
V[FRI]	1.5	2.8	5.2	19.0	—	—	—
HV[PST]	3.1	6.1	12.4	24.6	49.5	—	—
HV[BD-64]	1.4	2.4	4.4	8.1	15.8	29.9	59.4
HV[BD-127]	1.3	2.5	4.6	8.6	16.4	32.9	62.9
HV[BD-256]	7.2	14.1	27.3	53.9	106.5	—	—
HV[BF]	1.1	2.0	3.9	7.8	—	—	—
HV[BF FRI]	1.0	1.9	3.7	7.4	14.9	—	—
HV[Z FRI]	1.2	2.4	5.2	—	—	—	—

π (KB)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	461.7	488.7	526.1	544.7	—	—	—
V[FRI]	780.5	821.9	892.9	940.3	—	—	—
HV[PST]	49.1	50.7	52.3	53.9	55.5	—	—
HV[BD-64]	71325	94397	121005	167148	218922	311206	413312
HV[BD-127]	65034	85351	108422	147614	193756	270696	362980
HV[BD-256]	94414	121023	167166	218942	311226	—	—
HV[BF]	52153	59387	67882	77852	—	—	—
HV[BF FRI]	28843	30473	32129	33810	35517	—	—
HV[Z FRI]	5584	5874	6168	—	—	—	—

Table 5: Full System Crop (50% of original image) verifier runtime (left) and proof size (right) for various instantiations of VerITAS and HyperVerITAS on a commodity laptop.

T_V (s)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	13.8	27.5	54.9	109.7	220.6	—	—
V[FRI]	2.2	4.4	8.8	17.7	33.5	—	—
HV[PST]	5.3	10.8	21.9	43.4	87.0	—	—
HV[BD-64]	2.4	3.9	7.5	13.7	26.8	50.4	99.9
HV[BD-127]	2.4	4.4	7.7	15.0	27.9	55.0	105.0
HV[BD-256]	11.6	22.7	43.5	86.2	168.9	333.9	—
HV[BF]	1.7	3.3	6.3	12.2	24.7	—	—
HV[BF FRI]	1.6	3.1	6.1	12.3	24.9	—	—
HV[Z FRI]	1.9	3.9	7.9	16.0	—	—	—

π (KB)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	461.7	488.7	526.1	544.7	563.2	—	—
V[FRI]	780.5	821.9	892.9	940.3	983.1	—	—
HV[PST]	49.1	50.7	52.3	53.9	55.5	—	—
HV[BD-64]	71325	94397	121005	167148	218922	311206	413312
HV[BD-127]	65034	85351	108422	147614	193756	270696	362980
HV[BD-256]	94414	121023	167166	218942	311226	413333	—
HV[BF]	52153	59387	67882	77852	89902	—	—
HV[BF FRI]	28843	30473	32129	33810	35517	—	—
HV[Z FRI]	5584	5874	6168	6466	—	—	—

Table 6: Full System Crop (50% of original image) verifier runtime (left) and proof size (right) for various instantiations of VerITAS and HyperVerITAS on the AWS server.

T_P (s)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	73.1	146.9	316.4	—	—	—	—
V[FRI]	26.3	52.5	123.1	—	—	—	—
HV[PST]	19.1	39.1	77.4	171.7	—	—	—
HV[BD-127]	4.4	8.6	17.0	33.6	71.1	184.9	—
VIMz	217.1	439.9	773.9	1548.0	2901.4	5785.3	10336.3
TP-MT	109.8	219.5	423.4	831.1	1646.5	3293.1	6586.2

M_P (GB)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	10.3	19.2	24.1	—	—	—	—
V[FRI]	10.2	19.3	29.5	—	—	—	—
HV[PST]	7.8	15.1	27.0	27.2	—	—	—
HV[BD-127]	2.5	4.8	9.1	18.3	27.4	28.4	—
VIMz	2.2	2.5	4.3	4.3	8.5	9.0	12.6
TP-MT	4.5	4.5	4.5	4.5	4.5	4.5	4.5

T_V (s)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	13.9	27.8	59.4	—	—	—	—
V[FRI]	1.7	3.6	8.4	—	—	—	—
HV[PST]	2.6	5.1	10.4	20.6	—	—	—
HV[BD-127]	1.4	2.6	4.3	8.3	14.9	29.7	—
VIMz	0.2	0.2	0.3	0.3	0.5	0.5	1.1
TP-MT	4.5	9.1	17.5	34.3	68.0	136.0	272.0

π (KB)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	203.3	210.6	217.9	—	—	—	—
V[FRI]	516.9	543.8	590.0	—	—	—	—
HV[PST]	46.7	48.1	49.5	50.8	—	—	—
HV[BD-127]	86710	113799	144562	196817	258340	360927	—
VIMz	10.4	10.4	10.7	10.7	11.0	11.0	11.3
TP-MT	4.9	9.8	19.0	37.3	73.8	147.6	295.3

Table 7: Full System Grayscale prover runtime (top left), prover peak memory (top right), verifier runtime (bottom left), and proof size (bottom right) for VerITAS, HyperVerITAS, VIMz and TilesProof-MT (TP-MT) on a commodity laptop.

E.1 Full System Comparison on Commodity Laptop

We first provide a comprehensive comparison between HyperVerITAS, VerITAS, VIMz, and TilesProof-MT to examine the various trade-offs between the schemes. We considered four metrics: prover time (T_p), verifier time (T_v), prover peak memory (M_p), and proof size (π). Table 7 contains results for the full proof system, where we apply a grayscale to the original image. Notice that HyperVerITAS Brakedown has the fastest prover time for all input sizes up to 2^{24}. For images of size 2^{24}, HyperVerITAS Brakedown's prover is over 30x faster than VIMz and 17x faster than TilesProof-MT. Note that HyperVerITAS Brakedown is not able to generate a proof for 33MP images on a laptop, while VIMz and TilesProof-MT are able to do so. Meanwhile, VerITAS is only able to handle images of size 2^{21} (~2MP) for both the KZG and FRI instantiations. For the largest input size VerITAS can handle, HyperVerITAS Brakedown is 7x faster than VerITAS FRI, and 18x faster than VerITAS KZG.

We provide a more comprehensive comparison between HyperVerITAS and VerITAS in the top graph of Figure 9. Notice that all variants of HyperVerITAS have faster proof generation times than both VerITAS FRI and VerITAS KZG. This demonstrates how the design choices of HyperVerITAS drastically reduce proving time and

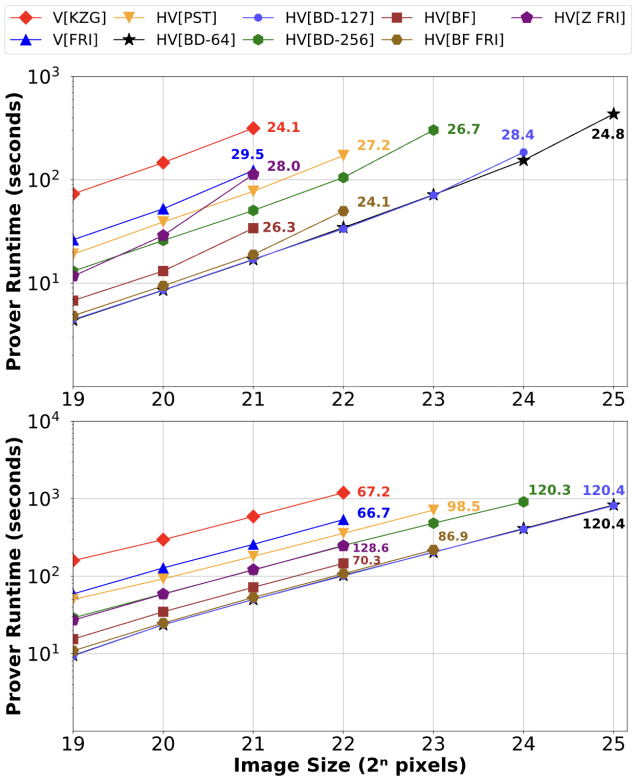

Figure 9: Prover runtime for full system grayscale on a commodity laptop (top) and AWS (bottom). For each scheme, we display the peak memory of the prover (in GB RAM) for the final data point by writing the value next it. Note the y-axis is log scaled.

memory usage, as compared to VerITAS. Table 8 contains the corresponding verifier runtime and proof size metrics for the schemes presented in the top graph of Figure 9. Notice that HyperVerITAS Brakedown's verifier is comparable to VerITAS FRI's for most input sizes, and is about 2x faster on input size 2^{21}. On the other hand, as we saw with crop, HyperVerITAS Brakedown has much worse proof sizes than VerITAS FRI. However, HyperVerITAS PST offers much smaller proof sizes than VerITAS, while maintaining faster prover time and lower memory usage.

E.2 Full System Comparison on High-Memory Server

We also compare the performance of HyperVerITAS and VerITAS on a high-memory server to see how the schemes compare when given more resources. As with crop, we compare HyperVerITAS instantiated with PST, Brakedown (over 64, 127, and 256 bit fields), Basefold, BasefoldFri, and ZeromorphFri to VerITAS instantiated with FRI and KZG. Here, we consider the full system implementation where we grayscale the original image. The results for prover runtime and memory are displayed in the bottom graph of Figure 9, while Table 9 contains the corresponding results for verifier runtime and proof size. Notice that all variants of HyperVerITAS have faster proof generation times than VerITAS FRI and KZG on all input sizes. On the server, VerITAS is able to generate proofs for images of size 2^{22}, one power greater than on the laptop. Even HyperVerITAS Brakedown instantiated over a 256-bit field is able to generate proofs for larger images than VerITAS FRI (which uses the quadratic extension of a 64-bit field), further showing the efficiency of HyperVerITAS.

T_V (s)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	13.9	27.8	59.4	—	—	—	—
V[FRI]	1.7	3.6	8.4	—	—	—	—
HV[PST]	2.6	5.1	10.4	20.6	—	—	—
HV[BD-64]	1.5	2.4	4.3	7.8	14.9	27.8	54.6
HV[BD-127]	1.4	2.6	4.3	8.3	14.9	29.7	—
HV[BD-256]	7.3	14.2	27.1	53.5	105.0	—	—
HV[BF]	1.0	1.8	3.5	—	—	—	—
HV[Bf FRI]	0.9	1.7	3.3	6.6	—	—	—
HV[Z FRI]	1.1	2.2	4.6	—	—	—	—

π (KB)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	203.3	210.6	217.9	—	—	—	—
V[FRI]	516.9	543.8	590.0	—	—	—	—
HV[PST]	46.7	48.1	49.5	50.8	—	—	—
HV[BD-64]	95099	125861	161339	222862	291894	414939	551080
HV[BD-127]	86710	113799	144562	196817	258340	360927	—
HV[BD-256]	125882	161361	222885	291919	414965	—	—
HV[BF]	65052	73935	84353	—	—	—	—
HV[BF FRI]	36017	37980	39970	41985	—	—	—
HV[Z FRI]	6876	7220	7569	—	—	—	—

Table 8: Full System Grayscale verifier runtime (left) and proof size (right) for various instantiations of VerITAS and HyperVerITAS on a commodity laptop.

T_V (s)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	13.9	27.7	55.6	111.1	—	—	—
V[FRI]	2.4	4.7	9.3	20.1	—	—	—
HV[PST]	4.1	8.3	16.6	33.3	66.2	—	—
HV[BD-64]	2.9	4.4	8.2	14.3	27.3	50.3	100.0
HV[BD-127]	2.5	4.7	7.8	15.1	26.7	53.9	100.7
HV[BD-256]	11.9	23.3	43.5	86.1	161.9	330.1	—
HV[BF]	1.6	2.9	5.6	10.8	—	—	—
HV[BF FRI]	1.5	2.8	5.5	10.7	21.5	—	—
HV[Z FRI]	1.8	3.5	7.0	14.4	—	—	—

π (KB)	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
V[KZG]	203.3	210.6	217.9	225.3	—	—	—
V[FRI]	516.9	543.8	590.0	619.1	—	—	—
HV[PST]	46.7	48.1	49.5	50.8	52.2	—	—
HV[BD-64]	95099	125861	161339	222862	291894	414939	551080
HV[BD-127]	86710	113799	144562	196817	258340	360927	483972
HV[BD-256]	125882	161361	222885	291919	414965	551107	—
HV[BF]	65052	73935	84353	96566	—	—	—
HV[BF FRI]	36017	37980	39970	41985	44026	—	—
HV[Z FRI]	6876	7220	7569	7922	—	—	—

Table 9: Full System Grayscale verifier runtime (left) and proof size (right) for various instantiations of VerITAS and HyperVerITAS on the AWS server.